

 eBook Gratuit

APPRENEZ

Angular 2

eBook gratuit non affilié créé à partir des
contributeurs de Stack Overflow.

#angular2

Table des matières

À propos.....	1
Chapitre 1: Démarrer avec Angular 2.....	2
Remarques.....	2
Versions.....	2
Exemples.....	3
Installez angular2 avec un angle angulaire.....	3
Conditions préalables:.....	3
Pour configurer un nouveau projet.....	3
Ajouter à un projet existant.....	4
Lancer le projet localement.....	4
Composants, directives, tuyaux et services générateurs.....	4
Premiers pas avec Angular 2 sans clics angulaires.....	6
Étape 1.....	6
Étape 2.....	6
Étape 3.....	8
Étape 5.....	9
Étape 6.....	9
Étape 7.....	10
Étape 8.....	11
Et maintenant?.....	11
Garder Visual Studios en phase avec les mises à jour NPM et NODE.....	11
Passer à travers ce proxy d'entreprise embêtant.....	12
Premiers pas avec Angular 2 avec backend node.js / expressjs (exemple http inclus).....	13
Conditions préalables.....	13
Feuille de route.....	13
Étape 1.....	13
Étape 2.....	13
Étape 3.....	14
Plongeons dans Angular 4!.....	18

Chapitre 2: Ajout dynamique de composants à l'aide de ViewContainerRef.createComponent	24
Exemples	24
Un composant wrapper qui ajoute des composants dynamiques de manière déclarative	24
Ajouter dynamiquement un composant sur un événement spécifique (cliquez sur)	25
Rendu le tableau de composants créé dynamiquement sur le modèle HTML dans Angular2	26
Chapitre 3: Angulaire - ForLoop	30
Syntaxe	30
Remarques	30
Exemples	30
Angulaire 2 pour boucle	30
NgFor - Markup For Loop	31
* ngPour les lignes de table	31
* ngPour le composant	31
* ngPour X quantité d'éléments par ligne	32
Chapitre 4: Angulaire 2 - rapporteur	33
Exemples	33
Test du routage Navbar avec Protractor	33
Angular2 Protractor - Installation	34
Chapitre 5: Angulaire-cli	36
Introduction	36
Exemples	36
Créer une application Angular2 vide avec des angles angulaires	36
Composants, directives, tuyaux et services générateurs	36
Ajout de bibliothèques tierces	36
construire avec cli angulaire	37
Nouveau projet avec scss / sass comme feuille de style	37
Définir le fil comme gestionnaire de paquets par défaut pour @ angular / cli	37
Exigences	38
Chapitre 6: Angular 2 Détection de changement et déclenchement manuel	39
Exemples	39
Exemple de base	39
Chapitre 7: Angular 2 formes de données pilotées	41

Remarques.....	41
Exemples.....	41
Formulaire piloté par les données.....	42
Chapitre 8: Angular2 Animations.....	44
Introduction.....	44
Exemples.....	44
Animation de base - Transpose un élément entre deux états pilotés par un attribut de modèle.....	44
Chapitre 9: Angular2 CanActivate.....	46
Exemples.....	46
Angular2 CanActivate.....	46
Chapitre 10: Angular2 Databinding.....	47
Exemples.....	47
@Contribution().....	47
Composant parent: Initialiser les listes d'utilisateurs.....	47
Chapitre 11: Angular2 en utilisant webpack.....	49
Exemples.....	49
Configuration angulaire de webpack 2.....	49
Chapitre 12: Angular2 Entrée () sortie ().....	53
Exemples.....	53
Contribution().....	53
Composant parent: Initialiser les listes d'utilisateurs.....	53
Exemple simple de propriétés d'entrée.....	54
Chapitre 13: Angular2 fournit des données externes à App avant le bootstrap.....	55
Introduction.....	55
Exemples.....	55
Via l'injection de dépendance.....	55
Chapitre 14: Angular2 Validations personnalisées.....	56
Paramètres.....	56
Exemples.....	56
Exemples de validateurs personnalisés:.....	56
Utiliser des validateurs dans le FormBuilder.....	56

get / set Les paramètres de contrôles FormBuilder.....	57
Chapitre 15: Animation.....	58
Exemples.....	58
Transition entre des états nuls.....	58
Animation entre plusieurs états.....	58
Chapitre 16: API Web Angular2 In Memory.....	60
Remarques.....	60
Exemples.....	60
Configuration de base.....	60
Configuration de plusieurs routes API de test.....	61
Chapitre 17: Baril.....	63
Introduction.....	63
Exemples.....	63
En utilisant baril.....	63
Chapitre 18: Bootstrap Module vide en angulaire 2.....	64
Exemples.....	64
Un module vide.....	64
Un module avec la mise en réseau sur le navigateur Web.....	64
Amorçage de votre module.....	64
Module racine d'application.....	65
Amorçage statique avec les classes d'usine.....	65
Chapitre 19: Chargement paresseux d'un module.....	66
Exemples.....	66
Exemple de chargement paresseux.....	66
Chapitre 20: CommandePar Pipe.....	68
Introduction.....	68
Exemples.....	68
Le tuyau.....	68
Chapitre 21: Comment utiliser ngfor.....	71
Introduction.....	71
Exemples.....	71

Exemple de liste non ordonnée.....	71
Exemple de modèle plus complexe.....	71
Suivi de l'exemple d'interaction en cours.....	71
Angular2 aliasé valeurs exportées.....	71
* ngPour la pipe.....	72
Chapitre 22: Comment utiliser ngif.....	73
Introduction.....	73
Syntaxe.....	73
Exemples.....	73
Afficher un message de chargement.....	73
Afficher le message d'alerte à condition.....	74
Pour exécuter une fonction au début ou à la fin de * ngFor loop using * ngIf.....	74
Utilisez * ngIf avec * ngFor.....	74
Chapitre 23: Compilation AOT avec Angular 2.....	76
Exemples.....	76
1. Installez les dépendances Angular 2 avec le compilateur.....	76
2. Ajoutez `angularCompilerOptions` à votre fichier `tsconfig.json`.....	76
3. Exécutez ngc, le compilateur angulaire.....	76
4. Modifiez le fichier `main.ts` pour utiliser NgFactory et le navigateur de plate-forme s.....	76
Pourquoi avons-nous besoin d'une compilation, d'une compilation de flux d'événements et d'.....	77
Utilisation de la compilation AoT avec la CLI angulaire.....	78
Chapitre 24: Composants.....	79
Introduction.....	79
Exemples.....	79
Un composant simple.....	79
Modèles et styles.....	79
Passer le modèle en tant que chemin de fichier.....	79
Passer un modèle en tant que code en ligne.....	80
Passer un tableau de chemins de fichiers.....	80
Passer un tableau de codes en ligne.....	80
Test d'un composant.....	80
Composants d'imbrication.....	82

Chapitre 25: Conception de matériau angulaire	83
Exemples	83
Md2Select	83
Md2Tooltip	83
Md2Toast	83
Md2Datepicker	84
Md2Accordion et Md2Collapse	84
Chapitre 26: Configuration de l'application ASP.net Core pour qu'elle fonctionne avec Angu	86
Introduction	86
Exemples	86
Asp.Net Core + Angular2 + Gulp	86
[Graine] Asp.Net Core + Angular2 + Gulp sur Visual Studio 2017	90
MVC <-> Angular 2	90
Chapitre 27: couverture de test angulaire-cli	92
Introduction	92
Exemples	92
Une simple couverture de test de base de commande angulaire-cli	92
Rapport détaillé de couverture de test graphique des composants individuels	92
Chapitre 28: Créer un paquet Angular 2+ NPM	94
Introduction	94
Exemples	94
Forfait le plus simple	94
Fichiers de configuration	94
.gitignore	94
.npmignore	94
gulpfile.js	94
index.d.ts	95
index.js	95
package.json	95
dist / tsconfig.json	96
src / angular-x-minimal-npm-package.component.ts	97

src / angular-x-minimal-npm-package.component.html.....	97
src / angular-x-data-table.component.css.....	97
src / angular-x-minimal-npm-package.module.ts.....	97
Construire et compiler.....	97
Publier.....	98
Chapitre 29: Créer une bibliothèque Angular npm.....	99
Introduction.....	99
Exemples.....	99
Module minimal avec classe de service.....	99
Structure de fichier.....	99
Service et module.....	99
Compilation.....	100
Paramètres NPM.....	101
Intégration continue.....	102
Chapitre 30: Crochets de cycle de vie.....	104
Remarques.....	104
Disponibilité des événements.....	104
Ordre des événements.....	104
Lectures complémentaires.....	104
Exemples.....	104
OnInit.....	104
OnDestroy.....	105
Modifications.....	105
AfterContentInit.....	105
AfterContentChecked.....	106
AfterViewInit.....	106
AfterViewChecked.....	107
DoCheck.....	107
Chapitre 31: CRUD dans Angular2 avec API Restful.....	108
Syntaxe.....	108
Exemples.....	108

Lecture d'une API Restful dans Angular2.....	108
Chapitre 32: Débogage de l'application typographique Angular2 à l'aide du code Visual Stud...	110
Exemples.....	110
Configuration de Launch.json pour votre espace de travail.....	110
Chapitre 33: Détection des événements de redimensionnement.....	112
Exemples.....	112
Un composant écoutant la fenêtre redimensionne l'événement.....	112
Chapitre 34: Directives.....	113
Syntaxe.....	113
Remarques.....	113
Exemples.....	113
Directive attribut.....	113
Le composant est une directive avec un modèle.....	113
Directives structurelles.....	113
Directive personnalisée.....	113
* ngPour.....	114
Directive Copier dans le Presse-papiers.....	115
Tester une directive personnalisée.....	116
Chapitre 35: Directives d'attribut affectant la valeur des propriétés sur le noeud hôte à	119
Exemples.....	119
@HostBinding.....	119
Chapitre 36: Directives et composants: @Input @Output.....	120
Syntaxe.....	120
Exemples.....	120
Exemple de saisie.....	120
Angular2 @Input et @Output dans un composant imbriqué.....	121
Angular2 @Input avec des données asynchrones.....	122
Composant parent avec appel asynchrone à un noeud final.....	122
Composant enfant ayant des données asynchrones en entrée.....	123
Chapitre 37: Directives et services généralement intégrés.....	125
Introduction.....	125

Exemples.....	125
Classe d'emplacement.....	125
AsyncPipe.....	125
Affichage de la version angulaire2 actuelle utilisée dans votre projet.....	126
Tuyau de monnaie.....	126
Chapitre 38: Dropzone dans Angular2.....	128
Exemples.....	128
Zone de largage.....	128
Chapitre 39: Exemple pour des routes telles que / route / subroute pour les URL statiques.....	130
Exemples.....	130
Exemple de route de base avec arbre de sous-routes.....	130
Chapitre 40: Exemples de composants avancés.....	131
Remarques.....	131
Exemples.....	131
Sélecteur d'image avec aperçu.....	131
Filtrer les valeurs de table par l'entrée.....	132
Chapitre 41: Http Interceptor.....	134
Remarques.....	134
Exemples.....	134
Classe simple Extension de la classe Http angulaire.....	134
Utiliser notre classe à la place de Http d'Angular.....	135
Simple HttpClient AuthToken Interceptor (Angular 4.3+).....	136
Chapitre 42: Ignorer la désinfection pour les valeurs de confiance.....	137
Paramètres.....	137
Remarques.....	137
SUPER IMPORTANT!.....	137
DÉSACTIVER LA DÉSINFECTION VOUS PERMET DE RISQUE DE XSS (Cross-Site Scripting) ET D'AUTRES.....	137
Exemples.....	137
Bypassing Sanitizing with pipes (pour réutiliser le code).....	137
Chapitre 43: Installer des plugins tiers avec angular-cli@1.0.0-beta.10.....	141
Remarques.....	141

Exemples.....	141
Ajout de la bibliothèque jquery dans un projet angular-cli.....	141
Ajouter une bibliothèque tierce qui n'a pas de typage.....	143
Chapitre 44: Interactions entre composants.....	145
Syntaxe.....	145
Paramètres.....	145
Exemples.....	145
Parent - Interaction enfant utilisant les propriétés @Input & @Output.....	145
Parent - Interaction enfant avec ViewChild.....	146
Interaction bidirectionnelle parent-enfant via un service.....	147
Chapitre 45: Interactions entre composants.....	150
Introduction.....	150
Exemples.....	150
Passer des données de parent à enfant avec une liaison d'entrée.....	150
Chapitre 46: Le routage.....	158
Exemples.....	158
Routage de base.....	158
Itinéraires enfant.....	160
ResolveData.....	161
Routage avec des enfants.....	164
Chapitre 47: Mise à jour angulaire de 2 formulaires.....	166
Remarques.....	166
Exemples.....	166
Formulaire de modification de mot de passe simple avec validation multi-contrôle.....	166
pw-change.template.html.....	166
pw-change.component.ts.....	167
pw-validators.ts.....	167
Angular 2: Formulaires pilotés par un modèle.....	168
Angular 2 Form - Validation personnalisée du courrier électronique / mot de passe.....	169
Angular 2: Reactive Forms (alias les formes réactives).....	170
formulaire d'inscription.component.ts.....	170
formulaire d'inscription.html.....	171

Angular 2 Forms (Reactive Forms) avec formulaire d'inscription et confirmation de la valid.....	171
app.module.ts.....	171
app.component.ts.....	171
app.component.html.....	172
validateurs.ts.....	173
Angular2 - Créateur de formulaire.....	173
Chapitre 48: Mise à jour des typings.....	175
Exemples.....	175
Mettre à jour les typages quand: typings WARN est obsolète.....	175
Chapitre 49: Mise à niveau de la force brutale.....	176
Introduction.....	176
Remarques.....	176
Exemples.....	176
Echafaudage d'un nouveau projet CLI angulaire.....	176
Chapitre 50: Mocking @ ngx / Store.....	177
Introduction.....	177
Paramètres.....	177
Remarques.....	177
Exemples.....	178
Observer Mock.....	178
Test unitaire pour composant avec Mock Store.....	178
Test d'unité pour le composant d'espionnage en magasin.....	179
Angular 2 - Mock Observable (composante service +).....	180
Magasin simple.....	183
Chapitre 51: Modèles.....	186
Introduction.....	186
Exemples.....	186
Angular 2 Modèles.....	186
Chapitre 52: Modules.....	188
Introduction.....	188
Exemples.....	188

Un module simple.....	188
Modules d'imbrication.....	188
Chapitre 53: Modules de fonctionnalités.....	190
Exemples.....	190
Un module de fonctionnalités.....	190
Chapitre 54: ngrx.....	191
Introduction.....	191
Exemples.....	191
Exemple complet: Connexion / Déconnexion d'un utilisateur.....	191
1) Définir IUser Interface.....	191
2) Déclarez les actions pour manipuler l' User.....	192
3) Définir l'état initial du UserReducer.....	193
4) Créez le réducteur UserReducer.....	193
Rappel: un réducteur doit être initialisé à un moment donné.....	194
5) Importer notre UserReducer dans notre module principal pour construire le Store.....	194
6) Utiliser les données de la Store pour afficher des informations à notre avis.....	195
Chapitre 55: npx-bootstrap personnalisé datepicker + entrée.....	198
Exemples.....	198
npx-bootstrap datepicker personnalisé.....	198
Chapitre 56: Optimisation du rendu à l'aide de ChangeDetectionStrategy.....	201
Exemples.....	201
Défaut vs OnPush.....	201
Chapitre 57: Ouvrier.....	203
Introduction.....	203
Exemples.....	203
Ajouter Service Worker à notre application.....	203
Chapitre 58: Pipes.....	206
Introduction.....	206
Paramètres.....	206
Remarques.....	206
Exemples.....	206

Chaining Pipes	206
Tuyaux personnalisés	207
Tuyaux Intégrés	207
Angular2 est livré avec quelques tuyaux intégrés:.....	207
Exemple	208
hotel-reservation.component.ts	208
hotel-reservation.template.html	208
Sortie	208
Déboguer avec JsonPipe	208
Code	209
Sortie	209
Pipe personnalisée disponible à l'échelle mondiale	209
Créer un tuyau personnalisé	209
Déballez les valeurs asynchrones avec un tuyau asynchrone	210
Extension d'un tuyau existant	210
Pipes Stateful	211
Tuyau dynamique	212
Tester un tuyau	214
Chapitre 59: redux angulaire	215
Exemples	215
De base	215
Obtenir l'état actuel	216
changer d'état	216
Ajouter l'outil chrome redux	217
Chapitre 60: Routage (3.0.0+)	218
Remarques	218
Exemples	218
Bootstrap	218
Configuration du routeur	218
Changer d'itinéraires (à l'aide de modèles et de directives)	219
Définir les itinéraires	220
Contrôle de l'accès à ou à partir d'une route	221

Comment fonctionnent les gardes de route	221
Interfaces de garde de route	221
Gardes de route synchrone vs asynchrone	221
Garde de route synchrone.....	222
Garde de route asynchrone.....	222
Ajouter un garde à la configuration de l'itinéraire.....	223
Utilisez Guard dans l'application bootstrap.....	223
Utiliser des résolveurs et des gardes.....	223
Chapitre 61: Service EventEmitter	226
Exemples.....	226
Aperçu de la classe.....	226
Composant de classe.....	226
Événements émouvants.....	226
Attraper l'événement.....	226
Exemple en direct.....	227
Chapitre 62: Services et injection de dépendance	228
Exemples.....	228
Exemple de service.....	228
Exemple avec Promise.resolve.....	229
Tester un service.....	230
Chapitre 63: Sujets et observables angulaires RXJS avec requêtes API	233
Remarques.....	233
Exemples.....	233
Demande de base.....	233
Encapsulation des requêtes API.....	233
Attendez plusieurs demandes.....	234
Chapitre 64: Test d'une application angulaire 2	236
Exemples.....	236
Installation du framework de test Jasmine.....	236
Installer	236
Vérifier	236

Mise en place de tests avec Gulp, Webpack, Karma et Jasmine	236
Test du service HTTP	241
Test des composants angulaires - De base	243
Chapitre 65: test unitaire	245
Exemples	245
Test élémentaire	245
fichier de composant	245
Chapitre 66: Tester ngModel	247
Introduction	247
Exemples	247
Test de base	247
Chapitre 67: Titre de la page	249
Introduction	249
Syntaxe	249
Exemples	249
changer le titre de la page	249
Chapitre 68: Utiliser des composants Web natifs dans Angular 2	250
Remarques	250
Exemples	250
Inclure le schéma des éléments personnalisés dans votre module	250
Utilisez votre composant web dans un modèle	250
Chapitre 69: Utiliser des librairies tierces comme jQuery dans Angular 2	251
Introduction	251
Exemples	251
Configuration en utilisant des angles angulaires	251
MNP	251
Dossier Actifs	251
Remarque	251
Utiliser jQuery dans les composants Angular 2.x	251
Chapitre 70: Zone.js	253
Exemples	253

Faire référence à NgZone.....	253
Utiliser NgZone pour effectuer plusieurs requêtes HTTP avant d'afficher les données.....	253
Crédits	255

À propos

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: [angular-2](#)

It is an unofficial and free Angular 2 ebook created for educational purposes. All the content is extracted from [Stack Overflow Documentation](#), which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official Angular 2.

The content is released under Creative Commons BY-SA, and the list of contributors to each chapter are provided in the credits section at the end of this book. Images may be copyright of their respective owners unless otherwise specified. All trademarks and registered trademarks are the property of their respective company owners.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Chapitre 1: Démarrer avec Angular 2

Remarques

Cette section fournit une vue d'ensemble de la façon d'installer et de configurer Angular2 + pour une utilisation dans divers environnements et IDE à l'aide d'outils tels que le système [angi-cli](#) développé par la communauté.

La version précédente de Angular est [AngularJS](#) ou encore Angular 1. Voir ici la [documentation](#) .

Versions

Version	Date de sortie
4.3.3	2017-08-02
4.3.2	2017-07-26
4.3.1	2017-07-19
4.3.0	2017-07-14
4.2.0	2017-06-08
4.1.0	2017-04-26
4.0.0	2017-03-23
2.3.0	2016-12-08
2.2.0	2016-11-14
2.1.0	2016-10-13
2.0.2	2016-10-05
2.0.1	2016-09-23
2.0.0	2016-09-14
2.0.0-rc.7	2016-09-13
2.0.0-rc.6	2016-08-31
2.0.0-rc.5	2016-08-09
2.0.0-rc.4	2016-06-30

Version	Date de sortie
2.0.0-rc.3	2016-06-21
2.0.0-rc.2	2016-06-15
2.0.0-rc.1	2016-05-03
2.0.0-rc.0	2016-05-02

Exemples

Installez angular2 avec un angle angulaire

Cet exemple est une configuration rapide de Angular 2 et comment générer un exemple de projet rapide.

Conditions préalables:

- [Node.js v4](#) ou supérieur.
- [npm v3](#) ou plus ou [fil](#) .

Ouvrez un terminal et exécutez les commandes une par une:

```
npm install -g @angular/cli
```

ou

```
yarn global add @angular/cli
```

en fonction de votre choix de gestionnaire de paquets.

La commande précédente installe globalement **@ angular / cli** , en ajoutant l'exécutable `ng` à `PATH`.

Pour configurer un nouveau projet

Naviguez avec le terminal dans un dossier où vous souhaitez configurer le nouveau projet.

Exécutez les commandes:

```
ng new PROJECT_NAME  
cd PROJECT_NAME  
ng serve
```

C'est tout, vous avez maintenant un exemple de projet simple réalisé avec Angular 2. Vous

pouvez maintenant naviguer jusqu'au lien affiché dans le terminal et voir ce qu'il est en train d'exécuter.

Ajouter à un projet existant

Accédez à la racine de votre projet en cours.

Exécutez la commande:

```
ng init
```

Cela ajoutera l'échafaudage nécessaire à votre projet. Les fichiers seront créés dans le répertoire en cours, assurez-vous donc de l'exécuter dans un répertoire vide.

Lancer le projet localement

Pour voir et interagir avec votre application pendant son exécution dans le navigateur, vous devez démarrer un serveur de développement local hébergeant les fichiers de votre projet.

```
ng serve
```

Si le serveur a démarré avec succès, il doit afficher une adresse sur laquelle le serveur est exécuté. Est habituellement ceci:

```
http://localhost:4200
```

Ce serveur de développement local est prêt à être utilisé avec Hot Module Reloading. Ainsi, toute modification apportée au code HTML, texte dactylographié ou CSS, déclenchera le rechargement automatique du navigateur (mais peut être désactivé si vous le souhaitez).

Composants, directives, tuyaux et services générateurs

La commande `ng generate <scaffold-type> <name>` (ou simplement `ng g <scaffold-type> <name>`) vous permet de générer automatiquement des composants angulaires:

```
# The command below will generate a component in the folder you are currently at
ng generate component my-generated-component
# Using the alias (same outcome as above)
ng g component my-generated-component
```

Il existe plusieurs types d'échafaudages possibles:

Type d'échafaudage	Usage
Module	<code>ng g module my-new-module</code>
Composant	<code>ng g component my-new-component</code>
Directif	<code>ng g directive my-new-directive</code>
Tuyau	<code>ng g pipe my-new-pipe</code>
Un service	<code>ng g service my-new-service</code>
Classe	<code>ng g class my-new-class</code>
Interface	<code>ng g interface my-new-interface</code>
Enum	<code>ng g enum my-new-enum</code>

Vous pouvez également remplacer le nom du type par sa première lettre. Par exemple:

`ng gm my-new-module` pour générer un nouveau module ou `ng gc my-new-component` pour créer un composant.

Bâtiment / groupement

Lorsque vous avez fini de créer votre application Web Angular 2 et que vous souhaitez l'installer sur un serveur Web comme Apache Tomcat, il vous suffit d'exécuter la commande de génération avec ou sans l'indicateur de production. La production minimisera le code et optimisera pour un environnement de production.

```
ng build
```

ou

```
ng build --prod
```

Recherchez ensuite dans le répertoire racine des projets un dossier `/dist` contenant la version.

Si vous souhaitez bénéficier des avantages d'une offre de production plus petite, vous pouvez également utiliser la compilation de modèles Ahead-of-Time, qui supprime le compilateur de modèle de la version finale:

```
ng build --prod --aot
```

Test d'unité

Angular 2 fournit des tests unitaires intégrés et chaque élément créé par angular-cli génère un test élémentaire de base, qui peut être étendu. Les tests unitaires sont écrits en jasmine et exécutés via Karma. Pour lancer le test, exécutez la commande suivante:

```
ng test
```

Cette commande exécute tous les tests du projet et les ré-exécute chaque fois qu'un fichier source change, qu'il s'agisse d'un test ou d'un code de l'application.

Pour plus d'infos, visitez également: la page [angular-cli github](#)

Premiers pas avec Angular 2 sans clics angulaires.

Angulaire 2.0.0-rc.4

Dans cet exemple, nous allons créer un "Hello World!" application avec un seul composant racine (`AppComponent`) pour des raisons de simplicité.

Conditions préalables:

- [Node.js](#) v5 ou version ultérieure
- npm v3 ou plus tard

Remarque: Vous pouvez vérifier les versions en exécutant `node -v` et `npm -v` dans la console / le terminal.

Étape 1

Créez et entrez un nouveau dossier pour votre projet. Appelons cela `angular2-example`.

```
mkdir angular2-example
cd angular2-example
```

Étape 2

Avant de commencer à écrire notre code d'application, nous ajouterons les 4 fichiers fournis ci-dessous: `package.json`, `tsconfig.json`, `typings.json` et `systemjs.config.js`.

Avertissement: Les mêmes fichiers peuvent être trouvés dans le [Quickstart officiel de 5 minutes](#).

`package.json` - Nous permet de télécharger toutes les dépendances avec npm et fournit une exécution de script simple pour faciliter la vie des projets simples. (Vous devriez envisager d'utiliser quelque chose comme [Gulp](#) à l'avenir pour automatiser les tâches).

```
{
  "name": "angular2-example",
  "version": "1.0.0",
  "scripts": {
    "start": "tsc && concurrently \"npm run tsc:w\" \"npm run lite\" ",
    "lite": "lite-server",
    "postinstall": "typings install",
    "tsc": "tsc",
```

```

    "tsc:w": "tsc -w",
    "typings": "typings"
  },
  "license": "ISC",
  "dependencies": {
    "@angular/common": "2.0.0-rc.4",
    "@angular/compiler": "2.0.0-rc.4",
    "@angular/core": "2.0.0-rc.4",
    "@angular/forms": "0.2.0",
    "@angular/http": "2.0.0-rc.4",
    "@angular/platform-browser": "2.0.0-rc.4",
    "@angular/platform-browser-dynamic": "2.0.0-rc.4",
    "@angular/router": "3.0.0-beta.1",
    "@angular/router-deprecated": "2.0.0-rc.2",
    "@angular/upgrade": "2.0.0-rc.4",
    "systemjs": "0.19.27",
    "core-js": "^2.4.0",
    "reflect-metadata": "^0.1.3",
    "rxjs": "5.0.0-beta.6",
    "zone.js": "^0.6.12",
    "angular2-in-memory-web-api": "0.0.14",
    "bootstrap": "^3.3.6"
  },
  "devDependencies": {
    "concurrently": "^2.0.0",
    "lite-server": "^2.2.0",
    "typescript": "^1.8.10",
    "typings": "^1.0.4"
  }
}

```

tscconfig.json - Configure le transpiler TypeScript.

```

{
  "compilerOptions": {
    "target": "es5",
    "module": "commonjs",
    "moduleResolution": "node",
    "sourceMap": true,
    "emitDecoratorMetadata": true,
    "experimentalDecorators": true,
    "removeComments": false,
    "noImplicitAny": false
  }
}

```

typings.json - Rend TypeScript reconnaît les bibliothèques que nous utilisons.

```

{
  "globalDependencies": {
    "core-js": "registry:dt/core-js#0.0.0+20160602141332",
    "jasmine": "registry:dt/jasmine#2.2.0+20160621224255",
    "node": "registry:dt/node#6.0.0+20160621231320"
  }
}

```

systemjs.config.js - Configure [SystemJS](#) (vous pouvez également utiliser [webpack](#)).


```

/**
 * System configuration for Angular 2 samples
 * Adjust as necessary for your application's needs.
 */
(function(global) {
  // map tells the System loader where to look for things
  var map = {
    'app': 'app', // 'dist',
    '@angular': 'node_modules/@angular',
    'angular2-in-memory-web-api': 'node_modules/angular2-in-memory-web-api',
    'rxjs': 'node_modules/rxjs'
  };
  // packages tells the System loader how to load when no filename and/or no extension
  var packages = {
    'app': { main: 'main.js', defaultExtension: 'js' },
    'rxjs': { defaultExtension: 'js' },
    'angular2-in-memory-web-api': { main: 'index.js', defaultExtension: 'js' },
  };
  var ngPackageNames = [
    'common',
    'compiler',
    'core',
    'forms',
    'http',
    'platform-browser',
    'platform-browser-dynamic',
    'router',
    'router-deprecated',
    'upgrade',
  ];
  // Individual files (~300 requests):
  function packIndex(pkgName) {
    packages['@angular/' + pkgName] = { main: 'index.js', defaultExtension: 'js' };
  }
  // Bundled (~40 requests):
  function packUmd(pkgName) {
    packages['@angular/' + pkgName] = { main: '/bundles/' + pkgName + '.umd.js',
    defaultExtension: 'js' };
  }
  // Most environments should use UMD; some (Karma) need the individual index files
  var setPackageConfig = System.packageWithIndex ? packIndex : packUmd;
  // Add package entries for angular packages
  ngPackageNames.forEach(setPackageConfig);
  var config = {
    map: map,
    packages: packages
  };
  System.config(config);
})(this);

```

Étape 3

Installons les dépendances en tapant

```
npm install
```

dans la console / le terminal.

Étape 4

Créez `index.html` intérieur du dossier `angular2-example` .

```
<html>
  <head>
    <title>Angular2 example</title>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <!-- 1. Load libraries -->
    <!-- Polyfill(s) for older browsers -->
    <script src="node_modules/core-js/client/shim.min.js"></script>
    <script src="node_modules/zone.js/dist/zone.js"></script>
    <script src="node_modules/reflect-metadata/Reflect.js"></script>
    <script src="node_modules/systemjs/dist/system.src.js"></script>
    <!-- 2. Configure SystemJS -->
    <script src="systemjs.config.js"></script>
    <script>
      System.import('app').catch(function(err){ console.error(err); });
    </script>
  </head>
  <!-- 3. Display the application -->
  <body>
    <my-app></my-app>
  </body>
</html>
```

Votre application sera rendue entre les balises `my-app` .

Cependant, Angular ne sait toujours pas *quoi* rendre. Pour le dire, nous allons définir `AppComponent` .

Étape 5

Créez un sous-dossier appelé `app` où nous pouvons définir les composants et les **services** qui composent notre application. (Dans ce cas, il vous reste plus qu'à contenir le `AppComponent` code et `main.ts` .)

```
mkdir app
```

Étape 6

Créez le fichier `app/app.component.ts`

```
import { Component } from '@angular/core';

@Component({
  selector: 'my-app',
  template: `
    <h1>{{title}}</h1>
    <ul>
      <li *ngFor="let message of messages">
        {{message}}
      </li>
    </ul>
  `
})
```

```

        </li>
    </ul>
    ,
  })
  export class AppComponent {
    title = "Angular2 example";
    messages = [
      "Hello World!",
      "Another string",
      "Another one"
    ];
  }
}

```

Que ce passe-t-il? Premièrement, nous importons le décorateur `@Component` que nous utilisons pour donner à Angular la balise HTML et le modèle pour ce composant. Ensuite, nous créons la classe `AppComponent` avec des `AppComponent` de `title` et de `messages` que nous pouvons utiliser dans le modèle.

Regardons maintenant ce modèle:

```

<h1>{{title}}</h1>
<ul>
  <li *ngFor="let message of messages">
    {{message}}
  </li>
</ul>

```

Nous affichons la variable `title` dans une balise `h1`, puis faisons une liste affichant chaque élément du tableau de `messages` en utilisant la directive `*ngFor`. Pour chaque élément du tableau, `*ngFor` crée une variable de `message` que nous utilisons dans l'élément `li`. Le résultat sera:

```

<h1>Angular 2 example</h1>
<ul>
  <li>Hello World!</li>
  <li>Another string</li>
  <li>Another one</li>
</ul>

```

Étape 7

Maintenant, nous créons un fichier `main.ts`, qui sera le premier fichier que Angular regarde.

Créez le fichier `app/main.ts`

```

import { bootstrap } from '@angular/platform-browser-dynamic';
import { AppComponent } from './app.component';

bootstrap(AppComponent);

```

Nous importons la fonction `bootstrap` et la classe `AppComponent`, puis utilisons `bootstrap` pour indiquer à Angular quel composant utiliser comme racine.

Étape 8

Il est temps de lancer votre première application. Type

```
npm start
```

dans votre console / terminal. Cela exécutera un script préparé à partir de `package.json` qui démarre `lite-server`, ouvre votre application dans une fenêtre de navigateur et exécute le transpiler TypeScript en mode montre (les fichiers `.ts` seront transpilés et le navigateur s'actualisera lorsque vous enregistrez les modifications) .

Et maintenant?

Consultez [le guide officiel Angular 2](#) et les autres rubriques de [la documentation de StackOverflow](#)

Vous pouvez également modifier `AppComponent` pour utiliser des modèles externes, des styles ou ajouter / modifier des variables de composant. Vous devriez voir vos modifications immédiatement après avoir enregistré des fichiers.

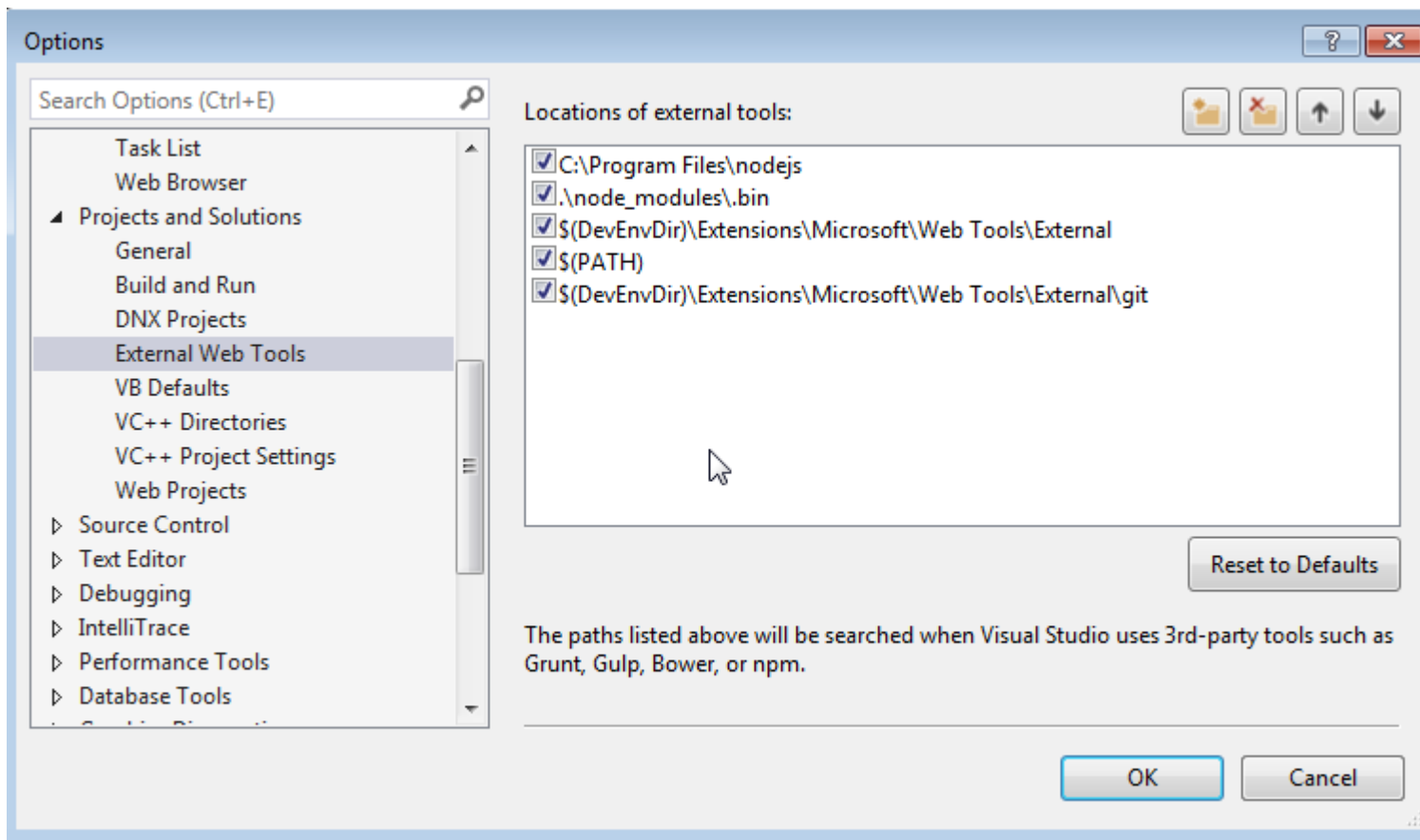
Garder Visual Studios en phase avec les mises à jour NPM et NODE

Étape 1: Recherchez votre téléchargement de Node.js, généralement installé sous `C: / program files / nodejs`

Étape 2: Ouvrez Visual Studios et accédez à "Outils> Options"

Étape 3: Dans la fenêtre des options, accédez à "Projets et solutions> Outils Web externes"

Étape 4: Ajoutez une nouvelle entrée avec votre emplacement de fichier Node.js (`C: / program files / nodejs`), IMPORTANT utilisez les boutons fléchés du menu pour déplacer votre référence vers le haut de la liste.



Étape 5: Redémarrez Visual Studios et exécutez une installation npm, contre votre projet, à partir de la fenêtre de commande npm

Passer à travers ce proxy d'entreprise embêtant

Si vous tentez de faire fonctionner un site Angular2 sur votre ordinateur Windows au XYZ MegaCorp, il est probable que vous rencontriez des problèmes pour accéder au proxy de la société.

Il y a (au moins) deux gestionnaires de paquets qui doivent passer par le proxy:

1. MNP
2. Dactylographie

Pour NPM, vous devez ajouter les lignes suivantes au fichier `.npmrc` :

```
proxy=http://[DOMAIN]%5C[USER]:[PASS]@[PROXY]:[PROXYPORT] /
https-proxy=http://[DOMAIN]%5C[USER]:[PASS]@[PROXY]:[PROXYPORT] /
```

Pour Typings, vous devez ajouter les lignes suivantes au fichier `.typingsrc` :

```
proxy=http://[DOMAIN]%5C[USER]:[PASS]@[PROXY]:[PROXYPORT] /
https-proxy=http://[DOMAIN]%5C[USER]:[PASS]@[PROXY]:[PROXYPORT] /
rejectUnauthorized=false
```

Ces fichiers n'existent probablement pas encore, vous pouvez donc les créer en tant que fichiers

texte vierges. Ils peuvent être ajoutés à la racine du projet (au même endroit que `package.json` ou vous pouvez les mettre dans `%HOMEPATH%` et ils seront disponibles pour tous vos projets).

Le bit qui n'est pas évident et la principale raison pour laquelle les gens pensent que les paramètres de proxy ne fonctionnent pas est le `%5C` qui est le code URL du `\` pour séparer les noms de domaine et d'utilisateur. Merci à Steve Roberts pour celui-ci: [Utiliser npm derrière le proxy d'entreprise .pac](#)

Premiers pas avec Angular 2 avec backend node.js / expressjs (exemple http inclus)

Nous allons créer un simple "Hello World!" app avec Angular2 2.4.1 (changement `@NgModule`) avec un backend node.js (expressjs).

Conditions préalables

- [Node.js](#) v4.xx ou supérieur
- [npm](#) v3.xx ou supérieur ou [fil](#)

Ensuite, exécutez `npm install -g typescript` ou `yarn global add typescript` pour installer les typescript globalement

Feuille de route

Étape 1

Créez un nouveau dossier (et le répertoire racine de notre back-end) pour notre application. Appelons cela `Angular2-express`.

ligne de commande :

```
mkdir Angular2-express
cd Angular2-express
```

Étape 2

Créez le `package.json` (pour les dépendances) et `app.js` (pour le bootstrap) pour notre application `node.js`

package.json:

```
{
  "name": "Angular2-express",
  "version": "1.0.0",
  "description": "",
```

```

"scripts": {
  "start": "node app.js"
},
"author": "",
"license": "ISC",
"dependencies": {
  "body-parser": "^1.13.3",
  "express": "^4.13.3"
}
}

```

app.js:

```

var express = require('express');
var app = express();
var server = require('http').Server(app);
var bodyParser = require('body-parser');

server.listen(process.env.PORT || 9999, function(){
  console.log("Server connected. Listening on port: " + (process.env.PORT || 9999));
});

app.use(bodyParser.json());
app.use(bodyParser.urlencoded({extended: true}));

app.use(express.static(__dirname + '/front'));

app.get('/test', function(req,res){ //example http request receiver
  return res.send(myTestVar);
});

//send the index.html on every page refresh and let angular handle the routing
app.get('/*', function(req, res, next) {
  console.log("Reloading");
  res.sendFile('index.html', { root: __dirname });
});

```

Ensuite, exécutez une `npm install` ou un `yarn npm install` pour installer les dépendances.

Maintenant, notre structure back-end est terminée. Passons au front-end.

Étape 3

Notre `Angular2-express` doit se trouver dans un dossier nommé `front` dans notre dossier `Angular2-express`.

ligne de commande:

```

mkdir front
cd front

```

Tout comme nous l'avons fait avec notre back-end, notre serveur frontal a également besoin des fichiers de dépendance. Allons de l'avant et créons les fichiers suivants: `package.json`, `systemjs.config.js`, `tsconfig.json`

package.json :

```
{
  "name": "Angular2-express",
  "version": "1.0.0",
  "scripts": {
    "tsc": "tsc",
    "tsc:w": "tsc -w"
  },
  "licenses": [
    {
      "type": "MIT",
      "url": "https://github.com/angular/angular.io/blob/master/LICENSE"
    }
  ],
  "dependencies": {
    "@angular/common": "~2.4.1",
    "@angular/compiler": "~2.4.1",
    "@angular/compiler-cli": "^2.4.1",
    "@angular/core": "~2.4.1",
    "@angular/forms": "~2.4.1",
    "@angular/http": "~2.4.1",
    "@angular/platform-browser": "~2.4.1",
    "@angular/platform-browser-dynamic": "~2.4.1",
    "@angular/platform-server": "^2.4.1",
    "@angular/router": "~3.4.0",
    "core-js": "^2.4.1",
    "reflect-metadata": "^0.1.8",
    "rxjs": "^5.0.2",
    "systemjs": "0.19.40",
    "zone.js": "^0.7.4"
  },
  "devDependencies": {
    "@types/core-js": "^0.9.34",
    "@types/node": "^6.0.45",
    "typescript": "2.0.2"
  }
}
```

systemjs.config.js:

```
/**
 * System configuration for Angular samples
 * Adjust as necessary for your application needs.
 */
(function (global) {
  System.config({
    defaultJSExtensions:true,
    paths: {
      // paths serve as alias
      'npm:': 'node_modules/'
    },
    // map tells the System loader where to look for things
    map: {
      // our app is within the app folder
      app: 'app',
      // angular bundles
      '@angular/core': 'npm:@angular/core/bundles/core.umd.js',
      '@angular/common': 'npm:@angular/common/bundles/common.umd.js',
```



```

    '@angular/compiler': 'npm:@angular/compiler/bundles/compiler.umd.js',
    '@angular/platform-browser': 'npm:@angular/platform-browser/bundles/platform-
browser.umd.js',
    '@angular/platform-browser-dynamic': 'npm:@angular/platform-browser-
dynamic/bundles/platform-browser-dynamic.umd.js',
    '@angular/http': 'npm:@angular/http/bundles/http.umd.js',
    '@angular/router': 'npm:@angular/router/bundles/router.umd.js',
    '@angular/forms': 'npm:@angular/forms/bundles/forms.umd.js',
    // other libraries
    'rxjs': 'npm:rxjs',
    'angular-in-memory-web-api': 'npm:angular-in-memory-web-api',
  },
  // packages tells the System loader how to load when no filename and/or no extension
  packages: {
    app: {
      main: './main.js',
      defaultExtension: 'js'
    },
    rxjs: {
      defaultExtension: 'js'
    }
  }
});
})(this);

```

tsconfig.json:

```

{
  "compilerOptions": {
    "target": "es5",
    "module": "commonjs",
    "moduleResolution": "node",
    "sourceMap": true,
    "emitDecoratorMetadata": true,
    "experimentalDecorators": true,
    "removeComments": false,
    "noImplicitAny": false
  },
  "compileOnSave": true,
  "exclude": [
    "node_modules/*"
  ]
}

```

Ensuite, exécutez une `npm install` ou un `yarn npm install` pour installer les dépendances.

Maintenant que nos fichiers de dépendance sont complets. Passons à notre `index.html` :

index.html:

```

<html>
  <head>
    <base href="/">
    <title>Angular2-express</title>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <!-- 1. Load libraries -->
    <!-- Polyfill(s) for older browsers -->

```

```

<script src="node_modules/core-js/client/shim.min.js"></script>
<script src="node_modules/zone.js/dist/zone.js"></script>
<script src="node_modules/reflect-metadata/Reflect.js"></script>
<script src="node_modules/systemjs/dist/system.src.js"></script>
<!-- 2. Configure SystemJS -->
<script src="systemjs.config.js"></script>
<script>
  System.import('app').catch(function(err){ console.error(err); });
</script>

</head>
<!-- 3. Display the application -->
<body>
  <my-app>Loading...</my-app>
</body>
</html>

```

Nous sommes maintenant prêts à créer notre premier composant. Créez un dossier nommé `app` dans notre dossier `front`.

ligne de commande:

```

mkdir app
cd app

```

Faisons les fichiers suivants nommés `main.ts`, `app.module.ts`, `app.component.ts`

main.ts:

```

import { platformBrowserDynamic } from '@angular/platform-browser-dynamic';

import { AppModule } from './app.module';

const platform = platformBrowserDynamic();
platform.bootstrapModule(AppModule);

```

app.module.ts:

```

import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { HttpClientModule } from "@angular/http";

import { AppComponent } from './app.component';

@NgModule({
  imports: [
    BrowserModule,
    HttpClientModule
  ],
  declarations: [
    AppComponent
  ],
  providers: [],
  bootstrap: [ AppComponent ]
})
export class AppModule {}

```

app.component.ts:

```
import { Component } from '@angular/core';
import { Http } from '@angular/http';

@Component({
  selector: 'my-app',
  template: 'Hello World!',
  providers: []
})
export class AppComponent {
  constructor(private http: Http){
    //http get example
    this.http.get('/test')
      .subscribe((res)=>{
        console.log(res);
      });
  }
}
```

Après cela, compilez les fichiers typographiques en fichiers javascript. Allez à 2 niveaux du répertoire actuel (dans le dossier Angular2-express) et exécutez la commande ci-dessous.

ligne de commande:

```
cd ..
cd ..
tsc -p front
```

Notre structure de dossiers devrait ressembler à:

```
Angular2-express
├─ app.js
├─ node_modules
├─ package.json
├─ front
│   ├─ package.json
│   ├─ index.html
│   ├─ node_modules
│   ├─ systemjs.config.js
│   ├─ tsconfig.json
│   └─ app
│       ├─ app.component.ts
│       ├─ app.component.js.map
│       ├─ app.component.js
│       ├─ app.module.ts
│       ├─ app.module.js.map
│       ├─ app.module.js
│       ├─ main.ts
│       ├─ main.js.map
│       └─ main.js
```

Enfin, dans le dossier Angular2-express, exécutez la commande `node app.js` dans la ligne de commande. Ouvrez votre navigateur préféré et vérifiez `localhost:9999` pour voir votre application.

Plongeons dans Angular 4!

Angular 4 est maintenant disponible! En fait, Angular utilise un demi-point depuis Angular 2, ce qui nécessite d'augmenter le nombre majeur lors de l'introduction des changements de rupture. L'équipe Angular a reporté les fonctionnalités qui provoquent des modifications brisées, qui seront publiées avec Angular 4. Angular 3 a été ignoré pour pouvoir aligner les numéros de version des modules principaux, car le routeur possédait déjà la version 3.

Selon l'équipe Angular, les applications Angular 4 vont consommer moins d'espace et être plus rapides qu'auparavant. Ils ont séparé le package d'animation du package @angular/core. Si quelqu'un n'utilise pas de package d'animation, l'espace supplémentaire du code ne se retrouvera pas dans la production. La syntaxe de liaison de modèle prend désormais en charge la syntaxe de style if / else. Angular 4 est maintenant compatible avec la version la plus récente de Typescript 2.1 et 2.2. Donc, Angular 4 va être plus excitant.

Maintenant, je vais vous montrer comment configurer Angular 4 dans votre projet.

Commençons l'installation angulaire de trois manières différentes:

Vous pouvez utiliser Angular-CLI (Interface de ligne de commande), cela installera toutes les dépendances pour vous.

- Vous pouvez migrer de Angular 2 à Angular 4.
- Vous pouvez utiliser github et cloner le code Angular4. (C'est le plus facile)
- Configuration angulaire à l'aide de la CLI angulaire (interface de ligne de commande).

Avant de commencer à utiliser Angular-CLI, assurez-vous que vous avez un nœud installé sur votre machine. Ici, j'utilise le nœud v7.8.0. Maintenant, ouvrez votre terminal et tapez la commande suivante pour Angular-CLI.

```
npm install -g @angular/cli
```

ou

```
yarn global add @angular/cli
```

selon le gestionnaire de paquets que vous utilisez.

Installons Angular 4 en utilisant Angular-CLI.

```
ng new Angular4-boilerplate
```

cd Angular4-ironplate Nous sommes tous prêts pour Angular 4. Sa méthode assez simple et facile.

Configuration angulaire en migrant de Angular 2 à Angular 4

Maintenant, voyons la deuxième approche. Je vais vous montrer comment migrer Angular 2 vers Angular 4. Pour cela, vous devez cloner un projet Angular 2 et mettre à jour les dépendances

Angular 2 avec la dépendance Angular 4 dans votre package.json comme suit:

```
"dependencies": {
  "@angular/animations": "^4.1.0",
  "@angular/common": "4.0.2",
  "@angular/compiler": "4.0.2",
  "@angular/core": "^4.0.1",
  "@angular/forms": "4.0.2",
  "@angular/http": "4.0.2",
  "@angular/material": "^2.0.0-beta.3",
  "@angular/platform-browser": "4.0.2",
  "@angular/platform-browser-dynamic": "4.0.2",
  "@angular/router": "4.0.2",
  "typescript": "2.2.2"
}
```

Ce sont les principales dépendances pour Angular 4. Maintenant, vous pouvez installer npm, puis npm pour lancer l'application. Pour référence mon package.json.

Configuration angulaire du projet github

Avant de commencer cette étape, assurez-vous d'avoir installé git sur votre machine. Ouvrez votre terminal et clonez la grille angulaire4 en utilisant la commande ci-dessous:

```
git@github.com:CypherTree/angular4-boilerplate.git
```

Ensuite, installez toutes les dépendances et exécutez-le.

```
npm install
npm start
```

Et vous avez terminé avec la configuration Angular 4. Toutes les étapes sont très simples, vous pouvez donc choisir l'une d'elles.

Structure du répertoire de la plaque angulaire4

```
Angular4-boilerplate
-karma
-node_modules
-src
  -mocks
  -models
    -loginform.ts
    -index.ts
  -modules
    -app
      -app.component.ts
    -app.component.html
    -login
      -login.component.ts
      -login.component.html
      -login.component.css
    -widget
      -widget.component.ts
```

```
-widget.component.html
-widget.component.css
.....
-services
  -login.service.ts
  -rest.service.ts
-app.routing.module.ts
-app.module.ts
-bootstrap.ts
-index.html
-vendor.ts
-typings
-webpack
-package.json
-tsconfig.json
-tslint.json
-typings.json
```

Compréhension de base pour la structure d'annuaire:

Tout le code réside dans le dossier src.

mocks folder est utilisé pour les données factices utilisées dans les tests.

Le dossier modèle contient la classe et l'interface utilisées dans le composant.

Le dossier Modules contient la liste des composants tels que l'application, le login, le widget, etc. Tous les composants contiennent des fichiers dactylographiés, HTML et CSS. index.ts sert à exporter toute la classe.

Le dossier services contient la liste des services utilisés dans l'application. J'ai séparé le service de repos et le service de composants différents. In rest service contient différentes méthodes http. Le service de connexion fonctionne en tant que médiateur entre le composant de connexion et le service de repos.

Le fichier app.routing.ts décrit toutes les routes possibles pour l'application.

app.module.ts décrit le module d'application comme composant racine.

bootstrap.ts exécutera toute l'application.

Le dossier webpack contient le fichier de configuration webpack.

Le fichier package.json est destiné à toutes les dépendances.

Le karma contient la configuration du karma pour le test unitaire.

node_modules contient la liste des paquets groupés.

Commençons avec le composant Login. Dans login.component.html

```
<form>Dreamfactory - Addressbook 2.0
  <label>Email</label> <input id="email" form="" name="email" type="email" />
  <label>Password</label> <input id="password" form="" name="password"
```

```
type="password" />
  <button form="">Login</button>
</form>
```

Dans login.component.ts

```
import { Component } from '@angular/core';
import { Router } from '@angular/router';
import { Form, FormGroup } from '@angular/forms';
import { LoginForm } from '../models';
import { LoginService } from '../services/login.service';

@Component({
  selector: 'login',
  template: require('./login.component.html'),
  styles: [require('./login.component.css')]
})
export class LoginComponent {

  constructor(private loginService: LoginService, private router: Router, form: LoginForm) {
  }

  getLogin(form: LoginForm): void {
    let username = form.email;
    let password = form.password;
    this.loginService.getAuthenticate(form).subscribe(() => {
      this.router.navigate(['/calender']);
    });
  }
}
```

Nous devons exporter ce composant dans index.ts.

```
export * from './login/login.component';
```

nous devons définir des itinéraires pour la connexion à app.routes.ts

```
const appRoutes: Routes = [
  {
    path: 'login',
    component: LoginComponent
  },
  .....
  {
    path: '',
    pathMatch: 'full',
    redirectTo: '/login'
  }
];
```

Dans le composant racine, le fichier app.module.ts, il vous suffit d'importer ce composant.

```
.....
import { LoginComponent } from './modules';
.....
@NgModule({
```

```
bootstrap: [AppComponent],
declarations: [
  LoginComponent
  .....
  .....
]
.....
})
export class AppModule { }
```

et après cette installation npm et npm commencent. Voici! Vous pouvez vérifier l'écran de connexion dans votre localhost. En cas de difficulté, vous pouvez vous référer à la grille angulaire4.

Fondamentalement, je peux sentir moins de construction de package et une réponse plus rapide avec l'application Angular 4 et bien que j'ai trouvé Exactement similaire à Angular 2 dans le codage.

Lire Démarrer avec Angular 2 en ligne: <https://riptutorial.com/fr/angular2/topic/789/demarrer-avec-angular-2>

Chapitre 2: Ajout dynamique de composants à l'aide de ViewContainerRef.createComponent

Exemples

Un composant wrapper qui ajoute des composants dynamiques de manière déclarative

Un composant personnalisé qui prend le type d'un composant en entrée et crée une instance de ce type de composant à l'intérieur de lui-même. Lorsque l'entrée est mise à jour, le composant dynamique ajouté précédemment est supprimé et le nouveau composant ajouté.

```
@Component({
  selector: 'dcl-wrapper',
  template: `<div #target></div>`
})
export class DclWrapper {
  @ViewChild('target', {
    read: ViewContainerRef
  }) target;
  @Input() type;
  cmpRef: ComponentRef;
  private isViewInitialized: boolean = false;

  constructor(private resolver: ComponentResolver) {}

  updateComponent() {
    if (!this.isViewInitialized) {
      return;
    }
    if (this.cmpRef) {
      this.cmpRef.destroy();
    }
    this.resolver.resolveComponent(this.type).then((factory: ComponentFactory < any > ) => {
      this.cmpRef = this.target.createComponent(factory)
      // to access the created instance use
      // this.cmpRef.instance.someProperty = 'someValue';
      // this.cmpRef.instance.someOutput.subscribe(val => doSomething());
    });
  }

  ngOnChanges() {
    this.updateComponent();
  }

  ngAfterViewInit() {
    this.isViewInitialized = true;
    this.updateComponent();
  }

  ngOnDestroy() {
```

```

    if (this.cmpRef) {
      this.cmpRef.destroy();
    }
  }
}

```

Cela vous permet de créer des composants dynamiques comme

```
<dcl-wrapper [type]="someComponentType"></dcl-wrapper>
```

Exemple Plunker

Ajouter dynamiquement un composant sur un événement spécifique (cliquez sur)

Fichier principal:

```

//our root app component
import {Component, NgModule, ViewChild, ViewContainerRef, ComponentFactoryResolver,
ComponentRef} from '@angular/core'
import {BrowserModule} from '@angular/platform-browser'
import {ChildComponent} from './childComp.ts'

@Component({
  selector: 'my-app',
  template: `
    <div>
      <h2>Hello {{name}}</h2>
      <input type="button" value="Click me to add element" (click) = addElement()> // call the
function on click of the button
      <div #parent> </div> // Dynamic component will be loaded here
    </div>
  `,
})
export class App {
  name:string;

  @ViewChild('parent', {read: ViewContainerRef}) target: ViewContainerRef;
  private componentRef: ComponentRef<any>;

  constructor(private componentFactoryResolver: ComponentFactoryResolver) {
    this.name = 'Angular2'
  }

  addElement(){
    let childComponent =
this.componentFactoryResolver.resolveComponentFactory(ChildComponent);
    this.componentRef = this.target.createComponent(childComponent);
  }
}

```

childComp.ts:

```
import{Component} from '@angular/core';
```

```

@Component({
  selector: 'child',
  template: `
    <p>This is Child</p>
  `,
})
export class ChildComponent {
  constructor() {

  }
}

```

app.module.ts:

```

@NgModule({
  imports: [ BrowserModule ],
  declarations: [ App, ChildComponent ],
  bootstrap: [ App ],
  entryComponents: [ChildComponent] // define the dynamic component here in module.ts
})
export class AppModule {}

```

Exemple Plunker

Rendu le tableau de composants créé dynamiquement sur le modèle HTML dans Angular2

Nous pouvons créer un composant dynamique et obtenir les instances du composant dans un tableau, puis le rendre sur un modèle.

Par exemple, nous pouvons prendre en compte deux composants de widget, ChartWidget et PatientWidget, qui ont étendu la classe WidgetComponent que je voulais ajouter dans le conteneur.

ChartWidget.ts

```

@Component({
  selector: 'chart-widget',
  templateUrl: 'chart-widget.component.html',
  providers: [{provide: WidgetComponent, useExisting: forwardRef(() => ChartWidget) }]
})

export class ChartWidget extends WidgetComponent implements OnInit {
  constructor(ngEl: ElementRef, renderer: Renderer) {
    super(ngEl, renderer);
  }
  ngOnInit() {}
  close() {
    console.log('close');
  }
  refresh() {
    console.log('refresh');
  }
  ...
}

```

chart-widget.component.html (en utilisant primeng Panel)

```
<p-panel [style]="{'margin-bottom':'20px'}">
  <p-header>
    <div class="ui-helper-clearfix">
      <span class="ui-panel-title" style="font-size:14px;display:inline-block;margin-top:2px">Chart Widget</span>
      <div class="ui-toolbar-group-right">
        <button pButton type="button" icon="fa-window-minimize"
(click)="minimize()"></button>
        <button pButton type="button" icon="fa-refresh" (click)="refresh()"></button>
        <button pButton type="button" icon="fa-expand" (click)="expand()" ></button>
        <button pButton type="button" (click)="close()" icon="fa-window-close"></button>
      </div>
    </div>
  </p-header>
  some data
</p-panel>
```

DataWidget.ts

```
@Component({
  selector: 'data-widget',
  templateUrl: 'data-widget.component.html',
  providers: [{provide: WidgetComponent, useExisting: forwardRef(() =>DataWidget) }]
})

export class DataWidget extends WidgetComponent implements OnInit {
  constructor(ngEl: ElementRef, renderer: Renderer) {
    super(ngEl, renderer);
  }
  ngOnInit() {}
  close(){
    console.log('close');
  }
  refresh(){
    console.log('refresh');
  }
  ...
}
```

data-widget.component.html (identique au widget graphique utilisant primeng Panel)

WidgetComponent.ts

```
@Component({
  selector: 'widget',
  template: '<ng-content></ng-content>'
})
export class WidgetComponent{
}
```

Nous pouvons créer des instances de composants dynamiques en sélectionnant les composants préexistants. Par exemple,

```
@Component({
```

```

    selector: 'dynamic-component',
    template: `<div #container><ng-content></ng-content></div>`
  })
  export class DynamicComponent {
    @ViewChild('container', {read: ViewContainerRef}) container: ViewContainerRef;

    public addComponent(ngItem: Type<WidgetComponent>): WidgetComponent {
      let factory = this.compFactoryResolver.resolveComponentFactory(ngItem);
      const ref = this.container.createComponent(factory);
      const newItem: WidgetComponent = ref.instance;
      this._elements.push(newItem);
      return newItem;
    }
  }
}

```

Enfin, nous l'utilisons dans le composant `app.component.ts`

```

@Component({
  selector: 'app-root',
  templateUrl: './app/app.component.html',
  styleUrls: ['./app/app.component.css'],
  entryComponents: [ChartWidget, DataWidget],
})

export class AppComponent {
  private elements: Array<WidgetComponent>=[];
  private WidgetClasses = {
    'ChartWidget': ChartWidget,
    'DataWidget': DataWidget
  }
  @ViewChild(DynamicComponent) dynamicComponent:DynamicComponent;

  addComponent(widget: string ): void{
    let ref= this.dynamicComponent.addComponent(this.WidgetClasses[widget]);
    this.elements.push(ref);
    console.log(this.elements);

    this.dynamicComponent.resetContainer();
  }
}

```

`app.component.html`

```

<button (click)="addComponent('ChartWidget')">Add ChartWidget</button>
<button (click)="addComponent('DataWidget')">Add DataWidget</button>

<dynamic-component [hidden]="true" ></dynamic-component>

<hr>
Dynamic Components
<hr>
<widget *ngFor="let item of elements">
  <div>{{item}}</div>
  <div [innerHTML]="item._ngEl.nativeElement.innerHTML | sanitizeHtml">
  </div>
</widget>

```

<https://plnkr.co/edit/lugU2pPsSBd3XhPHiUP1?p=preview>

Quelques modifications de @yurzui pour utiliser l'événement mouse sur les widgets

view.directive.ts

```
import {ViewRef, Directive, Input, ViewContainerRef} de '@ angular / core';
```

```
@Directive({
  selector: '[view]'
})
export class ViewDirective {
  constructor(private vcRef: ViewContainerRef) {}

  @Input()
  set view(view: ViewRef) {
    this.vcRef.clear();
    this.vcRef.insert(view);
  }

  ngOnDestroy() {
    this.vcRef.clear()
  }
}
```

app.component.ts

```
private elements: Array<{ view: ViewRef, component: WidgetComponent}> = [];

...
addComponent(widget: string ): void{
  let component = this.dynamicComponent.addComponent(this.WidgetClasses[widget]);
  let view: ViewRef = this.dynamicComponent.container.detach(0);
  this.elements.push({view, component});

  this.dynamicComponent.resetContainer();
}
```

app.component.html

```
<widget *ngFor="let item of elements">
  <ng-container *view="item.view"></ng-container>
</widget>
```

<https://plnkr.co/edit/JHpIHR43SvJd0OxJVMfV?p=preview>

Lire Ajout dynamique de composants à l'aide de ViewContainerRef.createComponent en ligne:
<https://riptutorial.com/fr/angular2/topic/831/ajout-dynamique-de-composants-a-l-aide-de-viewcontainerref-createcomponent>

Chapitre 3: Angulaire - ForLoop

Syntaxe

1. `<div *ngFor = "laisser un élément d'élément; laisser i = index"> {{i}} {{item}} </div>`

Remarques

La directive structurale `*ngFor` s'exécute en tant que boucle dans une collection et répète un morceau de code HTML pour chaque élément d'une collection.

@View décorateur @View est maintenant obsolète. Les développeurs doivent utiliser les propriétés `template` ou `'templateUrl'` pour le décorateur @Component .

Exemples

Angulaire 2 pour boucle

Pour vivre, [cliquez sur ...](#)

```
<!doctype html>
<html>
<head>
  <title>ng for loop in angular 2 with ES5.</title>
  <script type="text/javascript" src="https://code.angularjs.org/2.0.0-alpha.28/angular2.sfx.dev.js"></script>
  <script>
    var ngForLoop = function () {
      this.msg = "ng for loop in angular 2 with ES5.";
      this.users = ["Anil Singh", "Sunil Singh", "Sushil Singh", "Aradhya", 'Reena'];
    };

    ngForLoop.annotations = [
      new angular.Component({
        selector: 'ngforloop'
      }),
      new angular.View({
        template: '<H1>{{msg}}</H1>' +
          '<p> User List : </p>' +
          '<ul>' +
          '<li *ng-for="let user of users">' +
          '{{user}}' +
          '</li>' +
          '</ul>',
        directives: [angular.NgFor]
      })
    ];

    document.addEventListener("DOMContentLoaded", function () {
      angular.bootstrap(ngForLoop);
    });
  </script>
```

```

</head>
<body>
  <ngforloop></ngforloop>
  <h2>
    <a href="http://www.code-sample.com/" target="_blank">For more detail...</a>
  </h2>
</body>
</html>

```

NgFor - Markup For Loop

La directive **NgFor** instancie un modèle une fois par article à partir d'une itération. Le contexte de chaque modèle instancié hérite du contexte externe, la variable de boucle donnée étant définie sur l'élément actuel à partir de l'itérable.

Pour personnaliser l'algorithme de suivi par défaut, NgFor prend en **charge l'**option **trackBy** . **trackBy** prend une fonction qui a deux arguments: index et item. Si **trackBy** est donné, les pistes angulaires sont **modifiées** par la valeur de retour de la fonction.

```

<li *ngFor="let item of items; let i = index; trackBy: trackByFn">
  {{i}} - {{item.name}}
</li>

```

Options supplémentaires : NgFor fournit plusieurs valeurs exportées pouvant être associées à des variables locales:

- **index** sera défini sur l'itération de boucle en cours pour chaque contexte de modèle.
- **La** valeur booléenne indique si l'élément est le premier de l'itération.
- **last** sera défini sur une valeur booléenne indiquant si l'élément est le dernier de l'itération.
- **even** sera mis à une valeur booléenne indiquant si cet élément a un index pair.
- **odd** sera mis à une valeur booléenne indiquant si cet élément a un index impair.

* ngPour les lignes de table

```

<table>
  <thead>
    <th>Name</th>
    <th>Index</th>
  </thead>
  <tbody>
    <tr *ngFor="let hero of heroes">
      <td>{{hero.name}}</td>
    </tr>
  </tbody>
</table>

```

* ngPour le composant

```

@Component({
  selector: 'main-component',
  template: '<example-component

```



```

        *ngFor="let hero of heroes"
        [hero]="hero"></example-component>'
    })

@Component({
  selector: 'example-component',
  template: '<div>{{hero?.name}}</div>'
})

export class ExampleComponent {
  @Input() hero : Hero = null;
}

```

* ngPour X quantité d'éléments par ligne

L'exemple montre 5 articles par ligne:

```

<div *ngFor="let item of items; let i = index">
  <div *ngIf="i % 5 == 0" class="row">
    {{ item }}
    <div *ngIf="i + 1 < items.length">{{ items[i + 1] }}</div>
    <div *ngIf="i + 2 < items.length">{{ items[i + 2] }}</div>
    <div *ngIf="i + 3 < items.length">{{ items[i + 3] }}</div>
    <div *ngIf="i + 4 < items.length">{{ items[i + 4] }}</div>
  </div>
</div>

```

Lire Angulaire - ForLoop en ligne: <https://riptutorial.com/fr/angular2/topic/6543/angulaire---forloop>

Chapitre 4: Angulaire 2 - rapporteur

Exemples

Test du routage Navbar avec Protractor

Commençons par créer base navbar.html avec 3 options. (Accueil, Liste, Créer)

```
<nav class="navbar navbar-default" role="navigation">
<ul class="nav navbar-nav">
  <li>
    <a id="home-navbar" routerLink="/home">Home</a>
  </li>
  <li>
    <a id="list-navbar" routerLink="/create" >List</a>
  </li>
  <li>
    <a id="create-navbar" routerLink="/create">Create</a>
  </li>
</ul>
```

second permet de créer navbar.e2e-spec.ts

```
describe('Navbar', () => {

  beforeEach(() => {
    browser.get('home'); // before each test navigate to home page.
  });

  it('testing Navbar', () => {
    browser.sleep(2000).then(function() {
      checkNavbarTexts();
      navigateToListPage();
    });
  });

  function checkNavbarTexts() {
    element(by.id('home-navbar')).getText().then(function(text) { // Promise
      expect(text).toEqual('Home');
    });

    element(by.id('list-navbar')).getText().then(function(text) { // Promise
      expect(text).toEqual('List');
    });

    element(by.id('create-navbar')).getText().then(function(text) { // Promise
      expect(text).toEqual('Create');
    });
  }

  function navigateToListPage() {
    element(by.id('list-home')).click().then(function() { // first find list-home a tag and
    than click
      browser.sleep(2000).then(function() {
        browser.getCurrentUrl().then(function(actualUrl) { // promise
```

```

        expect(actualUrl.indexOf('list') !== -1).toBeTruthy(); // check the current url is
list
        });
    });

});
}
});

```

Angular2 Protractor - Installation

exécuter les commandes suivantes à cmd

- npm install -g protractor
- webdriver-manager update
- webdriver-manager start

**** créer le fichier protractor.conf.js dans la racine de l'application principale.**

très important de déclarer useAllAngular2AppRoots: true

```

const config = {
  baseUrl: 'http://localhost:3000/',

  specs: [
    './dev/**/*.e2e-spec.js'
  ],

  exclude: [],
  framework: 'jasmine',

  jasmineNodeOpts: {
    showColors: true,
    isVerbose: false,
    includeStackTrace: false
  },

  directConnect: true,

  capabilities: {
    browserName: 'chrome',
    shardTestFiles: false,
    chromeOptions: {
      'args': ['--disable-web-security ', '--no-sandbox', 'disable-extensions', 'start-
maximized', 'enable-crash-reporter-for-testing']
    }
  },

  onPrepare: function() {
    const SpecReporter = require('jasmine-spec-reporter');
    // add jasmine spec reporter
    jasmine.getEnv().addReporter(new SpecReporter({ displayStackTrace: true }));

    browser.ignoreSynchronization = false;
  },
  useAllAngular2AppRoots: true
};

```

```
if (process.env.TRAVIS) {
  config.capabilities = {
    browserName: 'firefox'
  };
}

exports.config = config;
```

créer un test de base dans le répertoire dev.

```
describe('basic test', () => {

  beforeEach(() => {
    browser.get('http://google.com');
  });

  it('testing basic test', () => {
    browser.sleep(2000).then(function() {
      browser.getCurrentUrl().then(function(actualUrl) {
        expect(actualUrl.indexOf('google') !== -1).toBeTruthy();
      });
    });
  });
});
```

courir dans cmd

```
protractor conf.js
```

Lire Angular 2 - rapporteur en ligne: <https://riptutorial.com/fr/angular2/topic/8900/angular2---rapporteur>

Chapitre 5: Angulaire-cli

Introduction

Ici vous trouverez comment commencer avec angular-cli, générer un nouveau composant / service / pipe / module avec angular-cli, ajouter 3 party comme bootstrap, construire un projet angulaire.

Exemples

Créer une application Angular2 vide avec des angles angulaires

Exigences:

- NodeJS: [page de téléchargement](#)
 - [npm](#) ou [fil](#)
-

Exécutez les commandes suivantes avec cmd du nouveau dossier répertoire:

1. `npm install -g @angular/cli` **OU** `yarn global add @angular/cli`
 2. `ng new PROJECT_NAME`
 3. `cd PROJECT_NAME`
 4. `ng serve`
-

Ouvrez votre navigateur sur localhost: 4200

Composants, directives, tuyaux et services générateurs

utilisez simplement votre cmd: Vous pouvez utiliser la commande `ng generate` (ou juste `ng g`) pour générer des composants angulaires:

- **Composant:** `ng g component my-new-component`
- **Directive:** `ng g directive my-new-directive`
- **Pipe:** `ng g pipe my-new-pipe`
- **Service:** `ng g service my-new-service`
- **Classe:** `ng g class my-new-classt`
- **Interface:** `ng g interface my-new-interface`
- **Enum:** `ng g enum my-new-enum`
- **Module:** `ng g module my-module`

Ajout de bibliothèques tierces

Dans `angular-cli.json`, vous pouvez modifier la configuration de l'application.

Si vous voulez ajouter ng2-bootstrap par exemple:

1. `npm install ng2-bootstrap --save` **OU** `yarn add ng2-bootstrap`
2. Dans `angular-cli.json`, ajoutez simplement le chemin du bootstrap sur `node-modules`.

```
"scripts": [  
  "../node_modules/jquery/dist/jquery.js",  
  "../node_modules/bootstrap/dist/js/bootstrap.js"  
]
```

construire avec cli angulaire

Dans la clé `angular-cli.json` at `outDir`, vous pouvez définir votre répertoire de construction;

ceux-ci sont équivalents

```
ng build --target=production --environment=prod  
ng build --prod --env=prod  
ng build --prod
```

et ainsi sont-ils

```
ng build --target=development --environment=dev  
ng build --dev --e=dev  
ng build --dev  
ng build
```

Lors de la construction, vous pouvez modifier la balise de base (`<base>`) dans votre `index.html` avec l'option `--base-href your-url`.

Définit la balise de base href sur `/ myUrl /` dans votre `index.html`

```
ng build --base-href /myUrl/  
ng build --bh /myUrl/
```

Nouveau projet avec scss / sass comme feuille de style

Les fichiers de style par défaut générés et compilés par `@angular/cli` sont **css** .

Si vous souhaitez utiliser **scss** à la place, générez votre projet avec:

```
ng new project_name --style=scss
```

Si vous souhaitez utiliser **sass** , générez votre projet avec:

```
ng new project_name --style=sass
```

Définir le fil comme gestionnaire de paquets par défaut pour `@ angular / cli`

Yarn est une alternative à npm, le gestionnaire de paquets par défaut sur @ angular / cli. Si vous voulez utiliser yarn comme gestionnaire de paquet pour @ angular / cli, suivez ces étapes:

Exigences

- [fil](#) (`npm install --global yarn` ou voir la [page d'installation](#))
- [@ angular / cli](#) (`npm install -g @angular/cli` ou `yarn global add @angular/cli`)

Pour définir le fil comme gestionnaire de paquet @ angular / cli:

```
ng set --global packageManager=yarn
```

Pour remettre npm en tant que @ angular / cli package manager:

```
ng set --global packageManager=npm
```

Lire Angulaire-cli en ligne: <https://riptutorial.com/fr/angular2/topic/8956/angulaire-cli>

Chapitre 6: Angular 2 Détection de changement et déclenchement manuel

Exemples

Exemple de base

Composant parent:

```
import {Component} from '@angular/core';

@Component({
  selector: 'parent-component',
  templateUrl: './parent-component.html'
})
export class ParentComponent {
  users : Array<User> = [];
  changeUsersActivation(user : User){
    user.changeButtonState();
  }
  constructor(){
    this.users.push(new User('Narco', false));
    this.users.push(new User('Bombasto', false));
    this.users.push(new User('Celeritas', false));
    this.users.push(new User('Magneta', false));
  }
}

export class User {
  firstName : string;
  active : boolean;

  changeButtonState(){
    this.active = !this.active;
  }
  constructor(_firstName :string, _active : boolean){
    this.firstName = _firstName;
    this.active = _active;
  }
}
```

Parent HTML:

```
<div>
  <child-component [usersDetails]="users"
    (changeUsersActivation)="changeUsersActivation($event)" >
  </child-component>
</div>
```

composant enfant:


```

import {Component, Input, EventEmitter, Output} from '@angular/core';
import {User} from "../parent.component";

@Component({
  selector: 'child-component',
  templateUrl: './child-component.html',
  styles: [`
    .btn {
      height: 30px;
      width: 100px;
      border: 1px solid rgba(0, 0, 0, 0.33);
      border-radius: 3px;
      margin-bottom: 5px;
    }
  `]
})
export class ChildComponent {
  @Input() usersDetails : Array<User> = null;
  @Output() changeUsersActivation = new EventEmitter();

  triggerEvent(user : User) {
    this.changeUsersActivation.emit(user);
  }
}

```

HTML enfant:

```

<div>
  <div>
    <table>
      <thead>
        <tr>
          <th>Name</th>
          <th></th>
        </tr>
      </thead>
      <tbody *ngIf="user !== null">
        <tr *ngFor="let user of usersDetails">
          <td>{{user.firstName}}</td>
          <td><button class="btn" (click)="triggerEvent(user)">{{user.active}}</button></td>
        </tr>
      </tbody>
    </table>
  </div>
</div>

```

Lire Angular 2 Détection de changement et déclenchement manuel en ligne:

<https://riptutorial.com/fr/angular2/topic/8971/angular-2-detection-de-changement-et-declenchement-manuel>

Chapitre 7: Angular 2 formes de données pilotées

Remarques

```
this.myForm = this.formBuilder.group
```

crée un objet de formulaire avec la configuration de l'utilisateur et l'affecte à la variable `this.myForm`.

```
'loginCredentials': this.formBuilder.group
```

méthode crée un groupe de contrôles composé d'un **formControlName**, par exemple. `login` et value `['', Validators.required]`, où le premier paramètre est la valeur initiale de l'entrée du formulaire et le secons est un validateur ou un tableau de validateurs comme dans `'email': ['', [Validators.required, customValidator]]`,

```
'hobbies': this.formBuilder.array
```

Crée un tableau de groupes où l'index du groupe est **formGroupName** dans le tableau et est accessible comme:

```
<div *ngFor="let hobby of myForm.find('hobbies').controls; let i = index">
  <div formGroupName="{i}">...</div>
</div>
```

```
onAddHobby() {
  (<FormArray>this.myForm.find('hobbies')).push(new FormGroup({
    'hobby': new FormControl('', Validators.required)
  }))
}
```

Cet exemple de méthode ajoute un nouveau `FormGroup` au tableau. L'accès actuel nécessite de spécifier le type de contrôle auquel nous voulons accéder. Dans cet exemple, ce type est:

```
<FormArray>
```

```
removeHobby(index: number) {
  (<FormArray>this.myForm.find('hobbies')).removeAt(index);
}
```

les mêmes règles que ci-dessus s'appliquent à la suppression d'un contrôle de formulaire spécifique du tableau

Exemples

Formulaire piloté par les données

Composant

```
import {Component, OnInit} from '@angular/core';
import {
  FormGroup,
  FormControl,
  FORM_DIRECTIVES,
  REACTIVE_FORM_DIRECTIVES,
  Validators,
  FormBuilder,
  FormArray
} from "@angular/forms";
import {Control} from "@angular/common";

@Component({
  moduleId: module.id,
  selector: 'app-data-driven-form',
  templateUrl: 'data-driven-form.component.html',
  styleUrls: ['data-driven-form.component.css'],
  directives: [FORM_DIRECTIVES, REACTIVE_FORM_DIRECTIVES]
})
export class DataDrivenFormComponent implements OnInit {
  myForm: FormGroup;

  constructor(private formBuilder: FormBuilder) {}

  ngOnInit() {
    this.myForm = this.formBuilder.group({
      'loginCredentials': this.formBuilder.group({
        'login': ['', Validators.required],
        'email': ['', [Validators.required, customValidator]],
        'password': ['', Validators.required]
      }),
      'hobbies': this.formBuilder.array([
        this.formBuilder.group({
          'hobby': ['', Validators.required]
        })
      ])
    });
  }

  removeHobby(index: number) {
    (<FormArray>this.myForm.find('hobbies')).removeAt(index);
  }

  onAddHobby() {
    (<FormArray>this.myForm.find('hobbies')).push(new FormGroup({
      'hobby': new FormControl('', Validators.required)
    }));
  }

  onSubmit() {
    console.log(this.myForm.value);
  }
}

function customValidator(control: Control): {[s: string]: boolean} {
  if(!control.value.match("[a-z0-9!#$$%&'+/=/?^_`{|}~-]+(?:\\. [a-z0-9!#$$%&'+/=/?^_`{|}~-"))
```

```

] +) * @ (?: [a-z0-9] (?: [a-z0-9-]* [a-z0-9])? \. ) + [a-z0-9] (?: [a-z0-9-]* [a-z0-9])? ") ) {
    return {error: true}
}
}
}

```

Balisateur HTML

```

<h3>Register page</h3>
<form [formGroup]="myForm" (ngSubmit)="onSubmit()">
  <div formGroupName="loginCredentials">
    <div class="form-group">
      <div>
        <label for="login">Login</label>
        <input id="login" type="text" class="form-control" formControlName="login">
      </div>
      <div>
        <label for="email">Email</label>
        <input id="email" type="text" class="form-control" formControlName="email">
      </div>
      <div>
        <label for="password">Password</label>
        <input id="password" type="text" class="form-control" formControlName="password">
      </div>
    </div>
  </div>
  <div class="row" >
    <div formGroupName="hobbies">
      <div class="form-group">
        <label>Hobbies array:</label>
        <div *ngFor="let hobby of myForm.find('hobbies').controls; let i = index">
          <div formGroupName="{{i}}">
            <input id="hobby_{{i}}" type="text" class="form-control" formControlName="hobby">
            <button *ngIf="myForm.find('hobbies').length > 1"
(click)="removeHobby(i)">x</button>
          </div>
        </div>
        <button (click)="onAddHobby()">Add hobby</button>
      </div>
    </div>
    <button type="submit" [disabled]="!myForm.valid">Submit</button>
  </form>

```

Lire Angular 2 formes de données pilotées en ligne:

<https://riptutorial.com/fr/angular2/topic/6463/angular-2-formes-de-donnees-pilotees>

Chapitre 8: Angular2 Animations

Introduction

Le système d'animation d'Angular vous permet de créer des animations qui s'exécutent avec le même type de performances natives que les animations CSS pures. Vous pouvez également intégrer étroitement votre logique d'animation au reste de votre code d'application pour en faciliter le contrôle.

Exemples

Animation de base - Transpose un élément entre deux états pilotés par un attribut de modèle.

app.component.html

```
<div>
  <div>
    <div *ngFor="let user of users">
      <button
        class="btn"
        [@buttonState]="user.active"
        (click)="user.changeButtonState()">{{user.firstName}}</button>
    </div>
  </div>
</div>
```

app.component.ts

```
import {Component, trigger, state, transition, animate, style} from '@angular/core';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styles: [`
    .btn {
      height: 30px;
      width: 100px;
      border: 1px solid rgba(0, 0, 0, 0.33);
      border-radius: 3px;
      margin-bottom: 5px;
    }
  `],
  animations: [
    trigger('buttonState', [
      state('true', style({
        background: '#04b104',
        transform: 'scale(1)'
      })),
    ]),
  ],
})
```

```

    state('false', style({
      background: '#e40202',
      transform: 'scale(1.1)'
    })),
    transition('true => false', animate('100ms ease-in')),
    transition('false => true', animate('100ms ease-out'))
  ])
]
})
export class AppComponent {
  users : Array<User> = [];
  constructor(){
    this.users.push(new User('Narco', false));
    this.users.push(new User('Bombasto', false));
    this.users.push(new User('Celeritas', false));
    this.users.push(new User('Magneta', false));
  }
}

export class User {
  firstName : string;
  active : boolean;

  changeButtonState(){
    this.active = !this.active;
  }
  constructor(_firstName :string, _active : boolean){
    this.firstName = _firstName;
    this.active = _active;
  }
}

```

Lire Angular2 Animations en ligne: <https://riptutorial.com/fr/angular2/topic/8970/angular2-animations>

Chapitre 9: Angular2 CanActivate

Exemples

Angular2 CanActivate

Implémenté dans un routeur:

```
export const MainRoutes: Route[] = [{
  path: '',
  children: [ {
    path: 'main',
    component: MainComponent ,
    canActivate : [CanActivateRoute]
  } ]
}];
```

Le fichier `canActivateRoute` :

```
@Injectable()
export class CanActivateRoute implements CanActivate{
  constructor(){}
  canActivate(next: ActivatedRouteSnapshot, state: RouterStateSnapshot): boolean {
    return true;
  }
}
```

Lire Angular2 CanActivate en ligne: <https://riptutorial.com/fr/angular2/topic/8899/angular2-canactivate>

Chapitre 10: Angular2 Databinding

Exemples

@Contribution()

Composant parent: Initialiser les listes d'utilisateurs.

```
@Component ({
  selector: 'parent-component',
  template: '<div>
    <child-component [users]="users"></child-component>
  </div>'
})
export class ParentComponent implements OnInit {
  let users : List<User> = null;

  ngOnInit() {
    users.push(new User('A', 'A', 'A@gmail.com'));
    users.push(new User('B', 'B', 'B@gmail.com'));
    users.push(new User('C', 'C', 'C@gmail.com'));
  }
}
```

Le composant enfant obtient l'utilisateur du composant parent avec Input ()

```
@Component ({
  selector: 'child-component',
  template: '<div>
    <table *ngIf="users !== null">
      <thead>
        <th>Name</th>
        <th>FName</th>
        <th>Email</th>
      </thead>
      <tbody>
        <tr *ngFor="let user of users">
          <td>{{user.name}}</td>
          <td>{{user.fname}}</td>
          <td>{{user.email}}</td>
        </tr>
      </tbody>
    </table>

  </div>',
})
export class ChildComponent {
  @Input() users : List<User> = null;
}

export class User {
  name : string;
```



```
fname : string;
email : string;

constructor(_name : string, _fname : string, _email : string){
  this.name = _name;
  this.fname = _fname;
  this.email = _email;
}
}
```

Lire Angular2 Databinding en ligne: <https://riptutorial.com/fr/angular2/topic/9036/angular2-databinding>

Chapitre 11: Angular2 en utilisant webpack

Exemples

Configuration angulaire de webpack 2

webpack.config.js

```
const webpack = require("webpack")
const helpers = require('./helpers')
const path = require("path")
const WebpackNotifierPlugin = require('webpack-notifier');

module.exports = {

  // set entry point for your app module
  "entry": {
    "app": helpers.root("app/main.module.ts"),
  },

  // output files to dist folder
  "output": {
    "filename": "[name].js",
    "path": helpers.root("dist"),
    "publicPath": "/",
  },

  "resolve": {
    "extensions": ['.ts', '.js'],
  },

  "module": {
    "rules": [
      {
        "test": /\.ts$/,
        "loaders": [
          {
            "loader": 'awesome-typescript-loader',
            "options": {
              "configFileName": helpers.root("./tsconfig.json")
            }
          },
          "angular2-template-loader"
        ]
      }
    ],
  },

  "plugins": [

    // notify when build is complete
    new WebpackNotifierPlugin({title: "build complete"}),

    // get reference for shims
    new webpack.DllReferencePlugin({
      "context": helpers.root("src/app"),
    })
  ]
}
```

```

    "manifest": helpers.root("config/polyfills-manifest.json")
  }},
  // get reference of vendor DLL
  new webpack.DllReferencePlugin({
    "context": helpers.root("src/app"),
    "manifest": helpers.root("config/vendor-manifest.json")
  }),
  // minify compiled js
  new webpack.optimize.UglifyJsPlugin(),
],
}

```

vendor.config.js

```

const webpack = require("webpack")
const helpers = require('./helpers')
const path = require("path")

module.exports = {
  // specify vendor file where all vendors are imported
  "entry": {
    // optionally add your shims as well
    "polyfills": [helpers.root("src/app/shims.ts")],
    "vendor": [helpers.root("src/app/vendor.ts")],
  },

  // output vendor to dist
  "output": {
    "filename": "[name].js",
    "path": helpers.root("dist"),
    "publicPath": "/",
    "library": "[name]"
  },

  "resolve": {
    "extensions": ['.ts', '.js'],
  },

  "module": {
    "rules": [
      {
        "test": /\.ts$/,
        "loaders": [
          {
            "loader": 'awesome-typescript-loader',
            "options": {
              "configFileName": helpers.root("./tsconfig.json")
            }
          }
        ],
      }
    ],
  },

  "plugins": [

    // create DLL for entries
    new webpack.DllPlugin({

```

```

        "name": "[name]",
        "context": helpers.root("src/app"),
        "path": helpers.root("config/[name]-manifest.json")
    }),

    // minify generated js
    new webpack.optimize.UglifyJsPlugin(),
  ],
}

```

helpers.js

```

var path = require('path');

var _root = path.resolve(__dirname, '..');

function root(args) {
  args = Array.prototype.slice.call(arguments, 0);
  return path.join.apply(path, [_root].concat(args));
}

exports.root = root;

```

vendor.ts

```

import "@angular/platform-browser"
import "@angular/platform-browser-dynamic"
import "@angular/core"
import "@angular/common"
import "@angular/http"
import "@angular/router"
import "@angular/forms"
import "rxjs"

```

index.html

```

<!DOCTYPE html>
<html>
<head>
  <title>Angular 2 webpack</title>

  <script src="/dist/vendor.js" type="text/javascript"></script>
  <script src="/dist/app.js" type="text/javascript"></script>
</head>
<body>
  <app>loading...</app>
</body>
</html>

```

package.json

```

{
  "name": "webpack example",
  "version": "0.0.0",
  "description": "webpack",
  "scripts": {

```

```
"build:webpack": "webpack --config config/webpack.config.js",
"build:vendor": "webpack --config config/vendor.config.js",
"watch": "webpack --config config/webpack.config.js --watch"
},
"devDependencies": {
  "@angular/common": "2.4.7",
  "@angular/compiler": "2.4.7",
  "@angular/core": "2.4.7",
  "@angular/forms": "2.4.7",
  "@angular/http": "2.4.7",
  "@angular/platform-browser": "2.4.7",
  "@angular/platform-browser-dynamic": "2.4.7",
  "@angular/router": "3.4.7",
  "webpack": "^2.2.1",
  "awesome-typescript-loader": "^3.1.2",
},
"dependencies": {
}
}
```

Lire Angular2 en utilisant webpack en ligne: <https://riptutorial.com/fr/angular2/topic/9554/angular2-en-utilisant-webpack>

Chapitre 12: Angular2 Entrée () sortie ()

Exemples

Contribution()

Composant parent: Initialiser les listes d'utilisateurs.

```
@Component({
  selector: 'parent-component',
  template: '<div>
    <child-component [users]="users"></child-component>
  </div>'
})
export class ParentComponent implements OnInit{
  let users : List<User> = null;

  ngOnInit() {
    users.push(new User('A', 'A', 'A@gmail.com');
    users.push(new User('B', 'B', 'B@gmail.com');
    users.push(new User('C', 'C', 'C@gmail.com');
  }
}
```

Le composant enfant obtient l'utilisateur du composant parent avec Input ()

```
@Component({
  selector: 'child-component',
  template: '<div>
    <table *ngIf="users !== null">
      <thead>
        <th>Name</th>
        <th>FName</th>
        <th>Email</th>
      </thead>
      <tbody>
        <tr *ngFor="let user of users">
          <td>{{user.name}}</td>
          <td>{{user.fname}}</td>
          <td>{{user.email}}</td>
        </tr>
      </tbody>
    </table>

  </div>',
})
export class ChildComponent {
  @Input() users : List<User> = null;
}

export class User {
  name : string;
```

```
fname : string;
email : string;

constructor(_name : string, _fname : string, _email : string){
  this.name = _name;
  this.fname = _fname;
  this.email = _email;
}
}
```

Exemple simple de propriétés d'entrée

Élément parent html

```
<child-component [isSelected]="inputPropValue"></child-component>
```

Élément parent ts

```
export class AppComponent {
  inputPropValue: true
}
```

Composant enfant ts:

```
export class ChildComponent {
  @Input() inputPropValue = false;
}
```

Composant enfant html:

```
<div [class.simpleCssClass]="inputPropValue"></div>
```

Ce code enverra le `inputPropValue` du composant parent à l'enfant et il aura la valeur que nous avons définie dans le composant parent quand il y arrivera - `false` dans notre cas. Nous pouvons ensuite utiliser cette valeur dans le composant enfant pour, par exemple, ajouter une classe à un élément.

Lire Angular2 Entrée () sortie () en ligne: <https://riptutorial.com/fr/angular2/topic/8943/angular2-entree----sortie--->

Chapitre 13: Angular2 fournit des données externes à App avant le bootstrap

Introduction

Dans cet article, je montrerai comment transmettre des données externes à l'application Angular avant le démarrage de l'application. Ces données externes peuvent être des données de configuration, des données héritées, un serveur rendu, etc.

Exemples

Via l'injection de dépendance

Au lieu d'invoquer directement le code d'amorçage de l'angulaire, encapsulez le code d'amorçage dans une fonction et exportez la fonction. Cette fonction peut également accepter des paramètres.

```
import { platformBrowserDynamic } from "@angular/platform-browser-dynamic";
import { AppModule } from "../src/app";
export function runAngular2App(legacyModel: any) {
  platformBrowserDynamic([
    { provide: "legacyModel", useValue: model }
  ]).bootstrapModule(AppModule)
  .then(success => console.log("Ng2 Bootstrap success"))
  .catch(err => console.error(err));
}
```

Ensuite, dans tous les services ou composants, nous pouvons injecter le «modèle hérité» et y accéder.

```
import { Injectable } from "@angular/core";
@Injectable()
export class MyService {
  constructor(@Inject("legacyModel") private legacyModel) {
    console.log("Legacy data - ", legacyModel);
  }
}
```

Exiger l'application et ensuite l'exécuter.

```
require(["myAngular2App"], function(app) {
  app.runAngular2App(legacyModel); // Input to your APP
});
```

Lire Angular2 fournit des données externes à App avant le bootstrap en ligne:

<https://riptutorial.com/fr/angular2/topic/9203/angular2-fournit-des-donnees-externes-a-app-avant-le-bootstrap>

Chapitre 14: Angular2 Validations personnalisées

Paramètres

paramètre	la description
contrôle	C'est le contrôle en cours de validation. En général, vous voudrez voir si <code>control.value</code> répond à certains critères.

Exemples

Exemples de validateurs personnalisés:

Angular 2 dispose de deux types de validateurs personnalisés. Les validateurs synchrones, comme dans le premier exemple, qui s'exécuteront directement sur les validateurs client et asynchrone (le deuxième exemple), que vous pouvez utiliser pour appeler un service distant afin d'effectuer la validation pour vous. Dans cet exemple, le validateur doit appeler le serveur pour voir si une valeur est unique.

```
export class CustomValidators {  
  
  static cannotContainSpace(control: Control) {  
    if (control.value.indexOf(' ') >= 0)  
      return { cannotContainSpace: true };  
  
    return null;  
  }  
  
  static shouldBeUnique(control: Control) {  
    return new Promise((resolve, reject) => {  
      // Fake a remote validator.  
      setTimeout(function () {  
        if (control.value == "exisitingUser")  
          resolve({ shouldBeUnique: true });  
        else  
          resolve(null);  
      }, 1000);  
    });  
  }  
}
```

Si votre valeur de contrôle est valide, vous retournez simplement `null` à l'appelant. Sinon, vous pouvez retourner un objet qui décrit l'erreur.

Utiliser des validateurs dans le FormBuilder

```
constructor(fb: FormBuilder) {
```

```
this.form = fb.group({
  firstInput: ['', Validators.compose([Validators.required,
CustomValidators.cannotContainSpace]), CustomValidators.shouldBeUnique],
  secondInput: ['', Validators.required]
});
}
```

Ici, nous utilisons le FormBuilder pour créer une forme très simple avec deux zones de saisie. FormBuilder prend un tableau pour trois arguments pour chaque contrôle d'entrée.

1. La valeur par défaut du contrôle.
2. Les validateurs qui s'exécuteront sur le client. Vous pouvez utiliser `Validators.compose` ([arrayOfValidators]) pour appliquer plusieurs validateurs sur votre contrôle.
3. Un ou plusieurs validateurs asynchrones de la même manière que le second argument.

get / set Les paramètres de contrôles FormBuilder

Il existe deux manières de définir les paramètres des contrôles FormBuilder.

1. À l'initialisation:

```
exampleForm : FormGroup;
constructor(fb: FormBuilder){
  this.exampleForm = fb.group({
    name : new FormControl({value: 'default name'}, Validators.compose([Validators.required,
Validators.maxLength(15)]))
  });
}
```

2. Après initialisation:

```
this.exampleForm.controls['name'].setValue('default name');
```

Obtenir la valeur du contrôle FormBuilder:

```
let name = this.exampleForm.controls['name'].value();
```

Lire [Angular2 Validations personnalisées en ligne](https://riptutorial.com/fr/angular2/topic/6284/angular2-validations-personnalisées):

<https://riptutorial.com/fr/angular2/topic/6284/angular2-validations-personnalisées>

Chapitre 15: Animation

Exemples

Transition entre des états nuls

```
@Component ({
  ...
  animations: [
    trigger('appear', [
      transition(':enter', [
        style({
          //style applied at the start of animation
        }),
        animate('300ms ease-in', style({
          //style applied at the end of animation
        })))
      ])
    ]
  })
})
class AnimComponent {
}
]
```

Animation entre plusieurs états

Le `<div>` dans ce modèle augmente à 50px , puis 100px , puis rétrécit à 20px lorsque vous cliquez sur le bouton.

Chaque `state` a un style associé décrit dans les métadonnées `@Component` .

La logique de l' `state` actif peut être gérée dans la logique du composant. Dans ce cas, la `size` variable composant contient la valeur de chaîne "small", "medium" ou "large".

L'élément `<div>` répond à cette valeur par le biais du `trigger` spécifié dans les métadonnées

`@Component : [@size]="size" .`

```
@Component ({
  template: '<div [@size]="size">Some Text</div><button
(click)="toggleSize()">TOGGLE</button>',
  animations: [
    trigger('size', [
      state('small', style({
        height: '20px'
      })),
      state('medium', style({
        height: '50px'
      })),
      state('large', style({
        height: '100px'
      })))
    ]
  })
})
class AnimComponent {
}
]
```

```
    )),
    transition('small => medium', animate('100ms')),
    transition('medium => large', animate('200ms')),
    transition('large => small', animate('300ms'))
  ])
]
})
export class TestComponent {

  size: string;

  constructor(){
    this.size = 'small';
  }
  toggleSize(){
    switch(this.size) {
      case 'small':
        this.size = 'medium';
        break;
      case 'medium':
        this.size = 'large';
        break;
      case 'large':
        this.size = 'small';
    }
  }
}
```

Lire Animation en ligne: <https://riptutorial.com/fr/angular2/topic/8127/animation>

Chapitre 16: API Web Angular2 In Memory

Remarques

J'ai principalement demandé ce sujet car je n'ai trouvé aucune information sur la configuration de plusieurs routes API avec Angular2-In-Memory-Web-API. J'ai fini par le découvrir moi-même et j'ai pensé que cela pourrait être utile aux autres.

Exemples

Configuration de base

mock-data.ts

Créer les données api simulées

```
export class MockData {
  createDb() {
    let mock = [
      { id: '1', name: 'Object A' },
      { id: '2', name: 'Object B' },
      { id: '3', name: 'Object C' },
      { id: '4', name: 'Object D' }
    ];

    return {mock};
  }
}
```

main.ts

Demandez à l'injecteur de dépendance de fournir les demandes InMemoryBackendService pour XHRBackend. Aussi, fournissez une classe qui comprend un

```
createDb()
```

function (dans ce cas, MockData) spécifiant les routes d'API simulées pour les requêtes SEED_DATA.

```
import { XHRBackend, HTTP_PROVIDERS } from '@angular/http';
import { InMemoryBackendService, SEED_DATA } from 'angular2-in-memory-web-api';
import { MockData } from './mock-data';
import { bootstrap } from '@angular/platform-browser-dynamic';

import { AppComponent } from './app.component';

bootstrap(AppComponent, [
  HTTP_PROVIDERS,
  { provide: XHRBackend, useClass: InMemoryBackendService },
  { provide: SEED_DATA, useClass: MockData }
])
```

```
]);
```

mock.service.ts

Exemple d'appel d'une requête get pour la route API créée

```
import { Injectable } from '@angular/core';
import { Http, Response } from '@angular/http';
import { Mock } from './mock';

@Injectable()
export class MockService {
  // URL to web api
  private mockUrl = 'app/mock';

  constructor (private http: Http) {}

  getData(): Promise<Mock[]> {
    return this.http.get(this.mockUrl)
      .toPromise()
      .then(this.extractData)
      .catch(this.handleError);
  }

  private extractData(res: Response) {
    let body = res.json();
    return body.data || { };
  }

  private handleError (error: any) {
    let errMsg = (error.message) ? error.message :
      error.status ? `${error.status} - ${error.statusText}` : 'Server error';
    console.error(errMsg);
    return Promise.reject(errMsg);
  }
}
```

Configuration de plusieurs routes API de test

mock-data.ts

```
export class MockData {
  createDb() {
    let mock = [
      { id: '1', name: 'Object A' },
      { id: '2', name: 'Object B' },
      { id: '3', name: 'Object C' }
    ];

    let data = [
      { id: '1', name: 'Data A' },
      { id: '2', name: 'Data B' },
      { id: '3', name: 'Data C' }
    ];

    return { mock, data };
  }
}
```

```
}
```

Maintenant, vous pouvez interagir avec les deux

```
app/mock
```

et

```
app/data
```

pour extraire leurs données correspondantes.

Lire API Web Angular2 In Memory en ligne: <https://riptutorial.com/fr/angular2/topic/6576/api-web-angular2-in-memory>

Chapitre 17: Baril

Introduction

Un baril permet de regrouper les exportations de plusieurs modules ES2015 en un seul module ES2015. Le canon lui-même est un fichier de module ES2015 qui réexporte les exportations sélectionnées des autres modules ES2015.

Exemples

En utilisant baril

Par exemple, sans baril, un consommateur aurait besoin de trois déclarations d'importation:

```
import { HeroComponent } from '../heroes/hero.component.ts';
import { Hero }          from '../heroes/hero.model.ts';
import { HeroService }  from '../heroes/hero.service.ts';
```

Nous pouvons ajouter un baril en créant un fichier dans le même dossier de composants. Dans ce cas, le dossier s'appelle «heroes» et s'appelle index.ts (en utilisant les conventions) qui exporte tous ces éléments:

```
export * from './hero.model.ts'; // re-export all of its exports
export * from './hero.service.ts'; // re-export all of its exports
export { HeroComponent } from './hero.component.ts'; // re-export the named thing
```

Maintenant, un consommateur peut importer ce dont il a besoin du baril.

```
import { Hero, HeroService } from '../heroes/index';
```

Pourtant, cela peut devenir une très longue ligne; qui pourrait être réduit davantage.

```
import * as h from '../heroes/index';
```

C'est assez réduit! Le `* as h` importe tous les modules et alias comme `h`

Lire Baril en ligne: <https://riptutorial.com/fr/angular2/topic/10717/baril>

Chapitre 18: Bootstrap Module vide en angulaire 2

Exemples

Un module vide

```
import { NgModule } from '@angular/core';

@NgModule({
  declarations: [], // components your module owns.
  imports: [], // other modules your module needs.
  providers: [], // providers available to your module.
  bootstrap: [] // bootstrap this root component.
})
export class MyModule {}
```

Ceci est un module vide ne contenant aucune déclaration, importation, fournisseur ou composant à bootstrap. Utilisez ceci une référence.

Un module avec la mise en réseau sur le navigateur Web.

```
// app.module.ts

import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { HttpClientModule } from '@angular/http';
import { MyRootComponent } from './app.component';

@NgModule({
  declarations: [MyRootComponent],
  imports: [BrowserModule, HttpClientModule],
  bootstrap: [MyRootComponent]
})
export class MyModule {}
```

`MyRootComponent` est le composant racine `MyRootComponent` dans `MyModule`. C'est le point d'entrée de votre application Angular 2.

Amorçage de votre module

```
import { platformBrowserDynamic } from '@angular/platform-browser-dynamic';
import { MyModule } from './app.module';

platformBrowserDynamic().bootstrapModule( MyModule );
```

Dans cet exemple, `MyModule` est le module contenant votre composant racine. En démarrant `MyModule` votre application Angular 2 est prête à fonctionner.

Module racine d'application

```
import { NgModule }      from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';

import { AppComponent }  from './app.component';

@NgModule({
  imports: [ BrowserModule ],
  declarations: [ AppComponent ],
  bootstrap: [ AppComponent ]
})
export class AppModule { }
```

Amorçage statique avec les classes d'usine

Nous pouvons amorcer statiquement une application en prenant la sortie Javascript ES5 simple des classes de fabrication générées. Ensuite, nous pouvons utiliser cette sortie pour amorcer l'application:

```
import { platformBrowser } from '@angular/platform-browser';
import { AppModuleNgFactory } from './main.ngfactory';

// Launch with the app module factory.
platformBrowser().bootstrapModuleFactory(AppModuleNgFactory);
```

Cela entraînera une réduction de la taille du lot d'applications, car toute la compilation de modèles était déjà effectuée lors d'une étape de construction, en utilisant ngc ou en appelant directement ses composants internes.

Lire Bootstrap Module vide en angulaire 2 en ligne:

<https://riptutorial.com/fr/angular2/topic/5508/bootstrap-module-vide-en-angulaire-2>

Chapitre 19: Chargement paresseux d'un module

Exemples

Exemple de chargement paresseux

Les modules de **chargement paresseux** nous aident à réduire le temps de démarrage. Avec le chargement différé, notre application n'a pas besoin de tout charger en une fois, il suffit de charger ce que l'utilisateur s'attend à voir lorsque l'application est chargée pour la première fois. Les modules chargés paresseusement ne seront chargés que lorsque l'utilisateur navigue sur leurs itinéraires.

app / app.module.ts

```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import { EagerComponent } from './eager.component';
import { routing } from './app.routing';
@NgModule({
  imports: [
    BrowserModule,
    routing
  ],
  declarations: [
    AppComponent,
    EagerComponent
  ],
  bootstrap: [AppComponent]
})
export class AppModule {}
```

app / app.component.ts

```
import { Component } from '@angular/core';
@Component({
  selector: 'my-app',
  template: `<h1>My App</h1>    <nav>
    <a routerLink="eager">Eager</a>
    <a routerLink="lazy">Lazy</a>
  </nav>
  <router-outlet></router-outlet>
`
})
export class AppComponent {}
```

app / app.routing.ts

```
import { ModuleWithProviders } from '@angular/core';
```

```
import { Routes, RouterModule } from '@angular/router';
import { EagerComponent } from './eager.component';
const routes: Routes = [
  { path: '', redirectTo: 'eager', pathMatch: 'full' },
  { path: 'eager', component: EagerComponent },
  { path: 'lazy', loadChildren: './lazy.module' }
];
export const routing: ModuleWithProviders = RouterModule.forRoot(routes);
```

app / eager.component.ts

```
import { Component } from '@angular/core';
@Component({
  template: `

Eager Component</p>`
})
export class EagerComponent {}


```

Il n'y a rien de spécial à propos de LazyModule, à part qu'il possède son propre routage et un composant appelé LazyComponent (mais il n'est pas nécessaire de nommer votre module ou un simliar de manière similaire).

app / lazy.module.ts

```
import { NgModule } from '@angular/core';
import { LazyComponent } from './lazy.component';
import { routing } from './lazy.routing';
@NgModule({
  imports: [routing],
  declarations: [LazyComponent]
})
export class LazyModule {}
```

app / lazy.routing.ts

```
import { ModuleWithProviders } from '@angular/core';
import { Routes, RouterModule } from '@angular/router';
import { LazyComponent } from './lazy.component';
const routes: Routes = [
  { path: '', component: LazyComponent }
];
export const routing: ModuleWithProviders = RouterModule.forChild(routes);
```

app / lazy.component.ts

```
import { Component } from '@angular/core';
@Component({
  template: `

Lazy Component</p>`
})
export class LazyComponent {}


```

Lire Chargement paresseux d'un module en ligne:

<https://riptutorial.com/fr/angular2/topic/7751/chargement-paresseux-d-un-module>

Chapitre 20: CommandePar Pipe

Introduction

Comment écrire un tube de commande et l'utiliser.

Exemples

Le tuyau

La mise en œuvre de la pipe

```
import {Pipe, PipeTransform} from '@angular/core';

@Pipe({
  name: 'orderBy',
  pure: false
})
export class OrderBy implements PipeTransform {

  value:string[] =[];

  static _orderByComparator(a:any, b:any):number{

    if(a === null || typeof a === 'undefined') a = 0;
    if(b === null || typeof b === 'undefined') b = 0;

    if((isNaN(parseFloat(a)) || !isFinite(a)) || (isNaN(parseFloat(b)) || !isFinite(b))){
      //Isn't a number so lowercase the string to properly compare
      if(a.toLowerCase() < b.toLowerCase()) return -1;
      if(a.toLowerCase() > b.toLowerCase()) return 1;
    }else{
      //Parse strings as numbers to compare properly
      if(parseFloat(a) < parseFloat(b)) return -1;
      if(parseFloat(a) > parseFloat(b)) return 1;
    }

    return 0; //equal each other
  }

  transform(input:any, config:string = '+'): any{

    //make a copy of the input's reference
    this.value = [...input];
    let value = this.value;

    if(!Array.isArray(value)) return value;

    if(!Array.isArray(config) || (Array.isArray(config) && config.length === 1)){
      let propertyToCheck:string = !Array.isArray(config) ? config : config[0];
      let desc = propertyToCheck.substr(0, 1) === '-';

      //Basic array
      if(!propertyToCheck || propertyToCheck === '-' || propertyToCheck === '+'){
```

```

    return !desc ? value.sort() : value.sort().reverse();
  } else {
    let property:string = propertyToCheck.substr(0, 1) === '+' ||
propertyToCheck.substr(0, 1) === '-'
      ? propertyToCheck.substr(1)
      : propertyToCheck;

    return value.sort(function(a:any,b:any){
      return !desc
        ? OrderBy._orderByComparator(a[property], b[property])
        : -OrderBy._orderByComparator(a[property], b[property]);
    });
  }
} else {
  //Loop over property of the array in order and sort
  return value.sort(function(a:any,b:any){
    for(let i:number = 0; i < config.length; i++){
      let desc = config[i].substr(0, 1) === '-';
      let property = config[i].substr(0, 1) === '+' || config[i].substr(0, 1) === '-'
        ? config[i].substr(1)
        : config[i];

      let comparison = !desc
        ? OrderBy._orderByComparator(a[property], b[property])
        : -OrderBy._orderByComparator(a[property], b[property]);

      //Don't return 0 yet in case of needing to sort by next property
      if(comparison !== 0) return comparison;
    }

    return 0; //equal each other
  });
}
}
}
}

```

Comment utiliser le tube dans le HTML - ordre croissant par prénom

```

<table>
  <thead>
    <tr>
      <th>First Name</th>
      <th>Last Name</th>
      <th>Age</th>
    </tr>
  </thead>
  <tbody>
    <tr *ngFor="let user of users | orderBy : ['firstName']">
      <td>{{user.firstName}}</td>
      <td>{{user.lastName}}</td>
      <td>{{user.age}}</td>
    </tr>
  </tbody>
</table>

```

Comment utiliser le tube dans le code HTML - ordre décroissant par prénom

```
<table>
  <thead>
    <tr>
      <th>First Name</th>
      <th>Last Name</th>
      <th>Age</th>
    </tr>
  </thead>
  <tbody>
    <tr *ngFor="let user of users | orderBy : ['-firstName']">
      <td>{{user.firstName}}</td>
      <td>{{user.lastName}}</td>
      <td>{{user.age}}</td>
    </tr>
  </tbody>
</table>
```

Lire CommandePar Pipe en ligne: <https://riptutorial.com/fr/angular2/topic/8969/commandepar-pipe>

Chapitre 21: Comment utiliser ngfor

Introduction

La directive `ngFor` est utilisée par Angular2 pour instancier un modèle une fois pour chaque élément d'un objet pouvant être itéré. Cette directive lie l'itérable au DOM, donc si le contenu de l'itérable change, le contenu du DOM sera également modifié.

Exemples

Exemple de liste non ordonnée

```
<ul>
  <li *ngFor="let item of items">{{item.name}}</li>
</ul>
```

Exemple de modèle plus complexe

```
<div *ngFor="let item of items">
  <p>{{item.name}}</p>
  <p>{{item.price}}</p>
  <p>{{item.description}}</p>
</div>
```

Suivi de l'exemple d'interaction en cours

```
<div *ngFor="let item of items; let i = index">
  <p>Item number: {{i}}</p>
</div>
```

Dans ce cas, je prendrai la valeur de l'index, qui correspond à l'itération de la boucle en cours.

Angular2 aliasé valeurs exportées

Angular2 fournit plusieurs valeurs exportées pouvant être associées à des variables locales. Ceux-ci sont:

- indice
- premier
- dernier
- même
- impair

Sauf `index`, les autres prennent une valeur `Boolean`. Comme dans l'exemple précédent utilisant `index`, vous pouvez utiliser n'importe laquelle de ces valeurs exportées:


```
<div *ngFor="let item of items; let firstItem = first; let lastItem = last">
  <p *ngIf="firstItem">I am the first item and I am gonna be showed</p>
  <p *ngIf="firstItem">I am not the first item and I will not show up :(</p>
  <p *ngIf="lastItem">But I'm gonna be showed as I am the last item :)</p>
</div>
```

* ngPour la pipe

```
import { Pipe, PipeTransform } from '@angular/core';
@Pipe({
  name: 'even'
})

export class EvenPipe implements PipeTransform {
  transform(value: string): string {
    if(value && value %2 === 0){
      return value;
    }
  }
}

@Component({
  selector: 'example-component',
  template: '<div>
    <div *ngFor="let number of numbers | even">
      {{number}}
    </div>
  </div>'
})

export class exampleComponent {
  let numbers : List<number> = Array.apply(null, {length: 10}).map(Number.call, Number)
}
```

Lire Comment utiliser ngfor en ligne: <https://riptutorial.com/fr/angular2/topic/8051/comment-utiliser-ngfor>

Chapitre 22: Comment utiliser ngIf

Introduction

* **NgIf** : **Supprime** ou recrée une partie de l'arborescence DOM en fonction d'une évaluation d'expression. C'est une directive structurale et les directives structurales modifient la disposition du DOM en ajoutant, en remplaçant et en supprimant ses éléments.

Syntaxe

- `<div * ngIf = "false"> test </ div> <! - évalué à false ->`
- `<div * ngIf = "undefined"> test </ div> <! - évalué à false ->`
- `<div * ngIf = "null"> test </ div> <! - évalué à false ->`
- `<div * ngIf = "0"> test </ div> <! - évalué à false ->`
- `<div * ngIf = "NaN"> test </ div> <! - évalué à false ->`
- `<div * ngIf = ""> test </ div> <! - évalué à false ->`
- Toutes les autres valeurs ont la valeur true.

Exemples

Afficher un message de chargement

Si notre composant n'est pas prêt et attend des données du serveur, alors nous pouvons ajouter le chargeur en utilisant * ngIf. **Pas**:

Commencez par déclarer un booléen:

```
loading: boolean = false;
```

Ensuite, dans votre composant, ajoutez un crochet de cycle de vie appelé `ngOnInit`

```
ngOnInit() {  
    this.loading = true;  
}
```

et après avoir obtenu des données complètes du serveur, définissez la valeur booléenne sur false.

```
this.loading=false;
```

Dans votre modèle HTML, utilisez * ngIf avec la propriété de `loading` :

```
<div *ngIf="loading" class="progress">  
    <div class="progress-bar info" style="width: 125%;"></div>  
</div>
```

Afficher le message d'alerte à condition

```
<p class="alert alert-success" *ngIf="names.length > 2">Currently there are more than 2 names!</p>
```

Pour exécuter une fonction au début ou à la fin de * ngFor loop using * ngIf

NgFor fournit des valeurs pouvant être associées à des variables locales

- **index** - (variable) position de l'élément courant dans l'itérable à partir de 0
- **first** - (booléen) true si l'élément en cours est le premier élément de l'itérable
- **last** - (booléen) true si l'élément en cours est le dernier élément de l'itérable
- **even** - (booléen) true si l'index actuel est un nombre pair
- **odd** - (booléen) true si l'index actuel est un nombre impair

```
<div *ngFor="let note of csvdata; let i=index; let lastcall=last">
  <h3>{{i}}</h3> <!-- to show index position
  <h3>{{note}}</h3>
  <span *ngIf="lastcall">{{anyfunction()}} </span><!-- this lastcall boolean value will be
true only if this is last in loop
  // anyfunction() will run at the end of loop same way we can do at start
</div>
```

Utilisez * ngIf avec * ngFor

Bien que vous ne soyez pas autorisé à utiliser *ngIf et *ngFor dans le même div (cela donnera une erreur dans le runtime), vous pouvez imbriquer le *ngIf dans le *ngFor pour obtenir le comportement souhaité.

Exemple 1: syntaxe générale

```
<div *ngFor="let item of items; let i = index">
  <div *ngIf="<your condition here>">

    <!-- Execute code here if statement true -->

  </div>
</div>
```

Exemple 2: éléments d'affichage avec index pair

```
<div *ngFor="let item of items; let i = index">
  <div *ngIf="i % 2 == 0">
    {{ item }}
  </div>
</div>
```

L'inconvénient est qu'un élément `div` supplémentaire doit être ajouté.

Mais considérez ce cas d'utilisation où un élément `div` doit être itéré (en utilisant * ngFor) et

inclut également une vérification pour savoir si l'élément doit être supprimé ou non (en utilisant *ngIf), mais ajouter un `div` supplémentaire n'est pas préféré. Dans ce cas, vous pouvez utiliser la balise `template` pour *ngFor:

```
<template ngFor let-item [ngForOf]="items">
  <div *ngIf="item.price > 100">
    </div>
</template>
```

De cette façon, l'ajout d'un `div` externe supplémentaire n'est pas nécessaire et l'élément `<template>` ne sera pas ajouté au DOM. Les seuls éléments ajoutés dans le DOM à partir de l'exemple ci-dessus sont les éléments `div` itérés.

Remarque: Dans Angular v4, `<template>` est devenu obsolète en faveur de `<ng-template>` et sera supprimé dans la version 5. Dans Angular v2.x, les versions `<template>` sont toujours valides.

Lire Comment utiliser ngif en ligne: <https://riptutorial.com/fr/angular2/topic/8346/comment-utiliser-ngif>

Chapitre 23: Compilation AOT avec Angular 2

Exemples

1. Installez les dépendances Angular 2 avec le compilateur

REMARQUE: pour de meilleurs résultats, assurez-vous que votre projet a été créé à l'aide de l'angulaire CLI.

```
npm install angular/{core,common,compiler,platform-browser,platform-browser-dynamic,http,router,forms,compiler-cli,tsc-wrapped,platform-server}
```

Vous n'avez pas à faire cette étape si votre projet a déjà un angle 2 et toutes ces dépendances installées. Assurez-vous simplement que le `compiler` est bien là.

2. Ajoutez `angularCompilerOptions` à votre fichier `tsconfig.json`

```
...  
"angularCompilerOptions": {  
  "genDir": "./ngfactory"  
}  
...
```

C'est le dossier de sortie du compilateur.

3. Exécutez `ngc`, le compilateur angulaire

de la racine de votre projet `./node_modules/.bin/ngc -p src` où `src` est l'endroit où tous vos codes angulaires 2 vivent. Cela va générer un dossier appelé `ngfactory` où tout votre code compilé va vivre.

`"node_modules/.bin/ngc" -p src` pour windows

4. Modifiez le fichier `main.ts` pour utiliser `NgFactory` et le navigateur de plateforme statique

```
// this is the static platform browser, the usual counterpart is @angular/platform-browser-dynamic.  
import { platformBrowser } from '@angular/platform-browser';  
  
// this is generated by the angular compiler  
import { AppModuleNgFactory } from './ngfactory/app/app.module.ngfactory';  
  
// note the use of `bootstrapModuleFactory`, as opposed to `bootstrapModule`.  
platformBrowser().bootstrapModuleFactory(AppModuleNgFactory);
```

À ce stade, vous devriez pouvoir exécuter votre projet. Dans ce cas, mon projet a été créé à l'aide

de l'angulaire-CLI.

```
> ng serve
```

Pourquoi avons-nous besoin d'une compilation, d'une compilation de flux d'événements et d'un exemple?

Q. Pourquoi avons-nous besoin d'une compilation? Ans. Nous avons besoin d'une compilation pour atteindre un niveau d'efficacité plus élevé de nos applications angulaires.

Regardez l'exemple suivant,

```
// ...
compile: function (el, scope) {
  var dirs = this._getElDirectives(el);
  var dir;
  var scopeCreated;
  dirs.forEach(function (d) {
    dir = Provider.get(d.name + Provider.DIRECTIVES_SUFFIX);
    if (dir.scope && !scopeCreated) {
      scope = scope.$new();
      scopeCreated = true;
    }
    dir.link(el, scope, d.value);
  });
  Array.prototype.slice.call(el.children).forEach(function (c) {
    this.compile(c, scope);
  }, this);
},
// ...
```

En utilisant le code ci-dessus pour rendre le modèle,

```
<ul>
  <li *ngFor="let name of names"></li>
</ul>
```

Est beaucoup plus lent par rapport à:

```
// ...
this._text_9 = this.renderer.createText(this._el_3, '\n', null);
this._text_10 = this.renderer.createText(parentRenderNode, '\n\n', null);
this._el_11 = this.renderer.createElement(parentRenderNode, 'ul', null);
this._text_12 = this.renderer.createText(this._el_11, '\n ', null);
this._anchor_13 = this.renderer.createTemplateAnchor(this._el_11, null);
this._appEl_13 = new import2.AppElement(13, 11, this, this._anchor_13);
this._TemplateRef_13_5 = new import17.TemplateRef_(this._appEl_13,
viewFactory_HomeComponent1);
this._NgFor_13_6 = new import15.NgFor(this._appEl_13.vcRef, this._TemplateRef_13_5,
this.parentInjector.get(import18.IterableDiffers), this.ref);
// ...
```

Flux d'événements avec une compilation en avance

En revanche, avec AoT, nous passons par les étapes suivantes:

1. Développement de l'application Angular 2 avec TypeScript.
2. Compilation de l'application avec ngc.
3. Effectue la compilation des modèles avec le compilateur Angular en TypeScript.
4. Compilation du code TypeScript en JavaScript.
5. Bundling
6. Minification.
7. Déploiement.

Bien que le processus ci-dessus semble légèrement plus compliqué, l'utilisateur passe par les étapes suivantes:

1. Téléchargez tous les actifs.
2. Cordons angulaires.
3. L'application est rendue.

Comme vous pouvez le constater, la troisième étape est manquante, ce qui signifie que l'UX est plus rapide / meilleur et que des outils tels que angular2-seed et angular-cli automatiseront considérablement le processus de construction.

J'espère que cela pourrait vous aider! Je vous remercie!

Utilisation de la compilation AoT avec la CLI angulaire

L' [interface](#) de ligne de commande [Angular CLI](#) prend en charge la compilation AoT depuis la version bêta 17.

Pour créer votre application avec la compilation AoT, exécutez simplement:

```
ng build --prod --aot
```

[Lire Compilation AOT avec Angular 2 en ligne:](#)

<https://riptutorial.com/fr/angular2/topic/6634/compilation-aot-avec-angular-2>

Chapitre 24: Composants

Introduction

Les composants angulaires sont des éléments composés par un modèle qui rendra votre application.

Exemples

Un composant simple

Pour créer un composant, nous ajoutons `@Component` décorateur `@Component` dans une classe transmettant certains paramètres:

- `providers` : ressources qui seront injectées dans le constructeur du composant
- `selector` : le sélecteur de requête qui trouvera l'élément dans le HTML et le remplacera par le composant
- `styles` : `styles` ligne. NOTE: NE PAS utiliser ce paramètre avec `require`, cela fonctionne sur le développement mais quand vous construisez l'application en production tous vos styles sont perdus
- `styleUrls` : Tableau de chemin d'accès aux fichiers de style
- `template` : chaîne contenant votre code HTML
- `templateUrl` : Chemin d'accès à un fichier HTML

Il y a d'autres paramètres que vous pouvez configurer, mais ceux qui sont listés sont ceux que vous utiliserez le plus.

Un exemple simple:

```
import { Component } from '@angular/core';

@Component({
  selector: 'app-required',
  styleUrls: ['required.component.scss'],
  // template: `This field is required.`,
  templateUrl: 'required.component.html',
})
export class RequiredComponent { }
```

Modèles et styles

Les modèles sont des fichiers HTML pouvant contenir une logique.

Vous pouvez spécifier un modèle de deux manières:

Passer le modèle en tant que chemin de fichier


```
@Component ({
  templateUrl: 'hero.component.html',
})
```

Passer un modèle en tant que code en ligne

```
@Component ({
  template: `<div>My template here</div>`,
})
```

Les modèles peuvent contenir des styles. Les styles déclarés dans `@Component` sont différents de votre fichier de style d'application, tout ce qui est appliqué dans le composant sera limité à cette étendue. Par exemple, disons que vous ajoutez:

```
div { background: red; }
```

Toutes les `div` dans le composant seront rouges, mais si vous avez d'autres composants, d'autres `div` dans votre HTML, ils ne seront pas du tout modifiés.

Le code généré ressemblera à ceci:

```
<style>div[_ngcontent-c1] { background: red; }</style>
```

Vous pouvez ajouter des styles à un composant de deux manières:

Passer un tableau de chemins de fichiers

```
@Component ({
  styleUrls: ['hero.component.css'],
})
```

Passer un tableau de codes en ligne

```
styles: [ `div { background: lime; }` ]
```

Vous ne devez pas utiliser de `styles` avec `require` car cela ne fonctionnera pas lorsque vous construisez votre application en production.

Test d'un composant

hero.component.html

```
<form (ngSubmit)="submit($event)" [formGroup]="form" novalidate>
  <input type="text" formControlName="name" />
  <button type="submit">Show hero name</button>
</form>
```

hero.component.ts

```
import { FormControl, FormGroup, Validators } from '@angular/forms';

import { Component } from '@angular/core';

@Component({
  selector: 'app-hero',
  templateUrl: 'hero.component.html',
})
export class HeroComponent {
  public form = new FormGroup({
    name: new FormControl('', Validators.required),
  });

  submit(event) {
    console.log(event);
    console.log(this.form.controls.name.value);
  }
}
```

hero.component.spec.ts

```
import { ComponentFixture, TestBed, async } from '@angular/core/testing';

import { HeroComponent } from './hero.component';
import { ReactiveFormsModule } from '@angular/forms';

describe('HeroComponent', () => {
  let component: HeroComponent;
  let fixture: ComponentFixture<HeroComponent>;

  beforeEach(async(() => {
    TestBed.configureTestingModule({
      declarations: [HeroComponent],
      imports: [ReactiveFormsModule],
    }).compileComponents();

    fixture = TestBed.createComponent(HeroComponent);
    component = fixture.componentInstance;
    fixture.detectChanges();
  }));

  it('should be created', () => {
    expect(component).toBeTruthy();
  });

  it('should log hero name in the console when user submit form', async(() => {
    const heroName = 'Saitama';
    const element = <HTMLFormElement>fixture.debugElement.nativeElement.querySelector('form');

    spyOn(console, 'log').and.callThrough();

    component.form.controls['name'].setValue(heroName);

    element.querySelector('button').click();

    fixture.whenStable().then(() => {
      fixture.detectChanges();
    });
  }));
});
```

```

        expect(console.log).toHaveBeenCalledWith(heroName);
    });
});

it('should validate name field as required', () => {
    component.form.controls['name'].setValue('');
    expect(component.form.invalid).toBeTruthy();
});
});

```

Composants d'imbrication

Les composants s'affichent dans leur `selector` respectif, vous pouvez donc l'utiliser pour imbriquer des composants.

Si vous avez un composant qui affiche un message:

```

import { Component, Input } from '@angular/core';

@Component({
  selector: 'app-required',
  template: `{{name}} is required.`
})
export class RequiredComponent {
  @Input()
  public name: String = '';
}

```

Vous pouvez l'utiliser dans un autre composant en utilisant `app-required` (le sélecteur de ce composant):

```

import { Component, Input } from '@angular/core';

@Component({
  selector: 'app-sample',
  template: `
    <input type="text" name="heroName" />
    <app-required name="Hero Name"></app-required>
  `
})
export class RequiredComponent {
  @Input()
  public name: String = '';
}

```

Lire Composants en ligne: <https://riptutorial.com/fr/angular2/topic/10838/composants>

Chapitre 25: Conception de matériau angulaire

Exemples

Md2Select

Composant :

```
<md2-select [(ngModel)]="item" (change)="change($event)" [disabled]="disabled">
<md2-option *ngFor="let i of items" [value]="i.value" [disabled]="i.disabled">
  {{i.name}}</md2-option>
</md2-select>
```

Sélectionnez autoriser l'utilisateur à sélectionner une option parmi les options.

```
<md2-select></md2-select>
<md2-option></md2-option>
<md2-select-header></md2-select-header>
```

Md2Tooltip

L'info-bulle est une directive, elle permet à l'utilisateur d'afficher un texte de conseil pendant que l'utilisateur passe la souris sur un élément.

Une info-bulle aurait le balisage suivant.

```
<span tooltip-direction="left" tooltip="On the Left! ">Left</span>
<button tooltip="some message"
  tooltip-position="below"
  tooltip-delay="1000">Hover Me
</button>
```

Md2Toast

Toast est un service qui affiche des notifications dans la vue.

Crée et affiche une simple notification de toast.

```
import {Md2Toast} from 'md2/toast/toast';

@Component({
  selector: "...",
})

export class ... {

  ...
```

```

constructor(private toast: Md2Toast) { }
toastMe() {
this.toast.show('Toast message...');

--- or ---

this.toast.show('Toast message...', 1000);
}

...
}

```

Md2Datepicker

Le datepicker permet à l'utilisateur de sélectionner la date et l'heure.

```
<md2-datepicker [(ngModel)]="date"></md2-datepicker>
```

voir pour plus de détails [ici](#)

Md2Accordion et Md2Collapse

Md2Collapse : Réduire est une directive, il permet à l'utilisateur de basculer la visibilité de la section.

Exemples

Un effondrement aurait le balisage suivant.

```

<div [collapse]="isCollapsed">
  Lorem Ipsum Content
</div>

```

Md2Accordion : Accordéon permet à l'utilisateur de basculer la visibilité des différentes sections.

Exemples

Un accordéon aurait le balisage suivant.

```

<md2-accordion [multiple]="multiple">
  <md2-accordion-tab *ngFor="let tab of accordions"
    [header]="tab.title"
    [active]="tab.active"
    [disabled]="tab.disabled">
    {{tab.content}}
  </md2-accordion-tab>
  <md2-accordion-tab>
    <md2-accordion-header>Custom Header</md2-accordion-header>
    test content
  </md2-accordion-tab>
</md2-accordion>

```

Lire Conception de matériau angulaire en ligne:

<https://riptutorial.com/fr/angular2/topic/10005/conception-de-materiau-angulaire>

Chapitre 26: Configuration de l'application ASP.net Core pour qu'elle fonctionne avec Angular 2 et TypeScript

Introduction

SCENARIO: Fond Core ASP.NET Angular 2 Composants Angular 2 Front-End utilisant les contrôleurs Asp.net Core

Cela permet d'implémenter Angular 2 sur l'application Asp.Net Core. Il nous permet également d'appeler les contrôleurs MVC à partir de composants Angular 2 avec le résultat MVC View Support Angular 2.

Exemples

Asp.Net Core + Angular2 + Gulp

Startup.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Hosting;
using Microsoft.AspNetCore.Identity.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Logging;
using CoreAngular000.Data;
using CoreAngular000.Models;
using CoreAngular000.Services;
using Microsoft.Extensions.FileProviders;
using System.IO;

namespace CoreAngular000
{
    public class Startup
    {
        public Startup(IHostingEnvironment env)
        {
            var builder = new ConfigurationBuilder()
                .SetBasePath(env.ContentRootPath)
                .AddJsonFile("appsettings.json", optional: false, reloadOnChange:
true)
                .AddJsonFile($"appsettings.{env.EnvironmentName}.json", optional:
true);

            if (env.IsDevelopment())
```

```

    {

        builder.AddUserSecrets<Startup>();
    }

    builder.AddEnvironmentVariables();
    Configuration = builder.Build();
}

public IConfigurationRoot Configuration { get; }

public void ConfigureServices(IServiceCollection services)
{
    // Add framework services.
    services.AddDbContext<ApplicationDbContext>(options =>
        options.UseSqlServer(Configuration.GetConnectionString("DefaultConnection")));

    services.AddIdentity<ApplicationUser, IdentityRole>()
        .AddEntityFrameworkStores<ApplicationDbContext>()
        .AddDefaultTokenProviders();

    services.AddMvc();

    // Add application services.
    services.AddTransient<IEmailSender, AuthMessageSender>();
    services.AddTransient<ISmsSender, AuthMessageSender>();
}

public void Configure(IApplicationBuilder app, IHostingEnvironment env,
ILoggerFactory loggerFactory)
{
    loggerFactory.AddConsole(Configuration.GetSection("Logging"));
    loggerFactory.AddDebug();

    if (env.IsDevelopment())
    {
        app.UseDeveloperExceptionPage();
        app.UseDatabaseErrorPage();
        app.UseBrowserLink();
    }
    else
    {
        app.UseExceptionHandler("/Home/Error");
    }

    app.UseDefaultFiles();
    app.UseStaticFiles();
    app.UseStaticFiles(new StaticFileOptions
    {
        FileProvider = new
PhysicalFileProvider(Path.Combine(env.ContentRootPath, "node_modules"),
        RequestPath = "/node_modules"
    });

    app.UseMvc(routes =>
    {
        routes.MapRoute(
            name: "default",
            template: "{controller=Home}/{action=Index}/{id?}");
    });
}

```



```
    }  
  }  
}
```

tsConfig.json

```
{  
  "compilerOptions": {  
    "diagnostics": true,  
    "emitDecoratorMetadata": true,  
    "experimentalDecorators": true,  
    "lib": [ "es2015", "dom" ],  
    "listFiles": true,  
    "module": "commonjs",  
    "moduleResolution": "node",  
    "noImplicitAny": true,  
    "outDir": "wwwroot",  
    "removeComments": false,  
    "rootDir": "wwwroot",  
    "sourceMap": true,  
    "suppressImplicitAnyIndexErrors": true,  
    "target": "es5"  
  },  
  "exclude": [  
    "node_modules",  
    "wwwroot/lib/"  
  ]  
}
```

Package.json

```
{  
  "name": "angular dependencies and web dev package",  
  "version": "1.0.0",  
  "description": "Angular 2 MVC. Samuel Maicas Template",  
  "scripts": {},  
  "dependencies": {  
    "@angular/common": "~2.4.0",  
    "@angular/compiler": "~2.4.0",  
    "@angular/core": "~2.4.0",  
    "@angular/forms": "~2.4.0",  
    "@angular/http": "~2.4.0",  
    "@angular/platform-browser": "~2.4.0",  
    "@angular/platform-browser-dynamic": "~2.4.0",  
    "@angular/router": "~3.4.0",  
    "angular-in-memory-web-api": "~0.2.4",  
    "systemjs": "0.19.40",  
    "core-js": "^2.4.1",  
    "rxjs": "5.0.1",  
    "zone.js": "^0.7.4"  
  },  
  "devDependencies": {  
    "del": "^2.2.2",  
    "gulp": "^3.9.1",  
    "gulp-concat": "^2.6.1",  
    "gulp-cssmin": "^0.1.7",  
    "gulp-htmlmin": "^3.0.0",  
    "gulp-uglify": "^2.1.2",  
    "merge-stream": "^1.0.1",  
  }  
}
```

```
"tslint": "^3.15.1",
"typescript": "~2.0.10"
},
"repository": {}
}
```

bundleconfig.json

```
[
  {
    "outputFileName": "wwwroot/css/site.min.css",
    "inputFiles": [
      "wwwroot/css/site.css"
    ]
  },
  {
    "outputFileName": "wwwroot/js/site.min.js",
    "inputFiles": [
      "wwwroot/js/site.js"
    ],
    "minify": {
      "enabled": true,
      "renameLocals": true
    },
    "sourceMap": false
  }
]
```

Convertissez bundleconfig.json en gulpfile (RightClick bundleconfig.json sur l'explorateur de solutions, Bundler & Minifier> Convert to Gulp

Vues / Accueil / Index.cshtml

```
@{
    ViewData["Title"] = "Home Page";
}
<div>{{ nombre }}</div>
```

Pour le dossier wwwroot, utilisez <https://github.com/angular/quickstart> seed. Vous avez besoin de: **index.html** **main.ts**, **systemjs-angular-loader.js**, **systemjs.config.js**, **tsconfig.json** et le **dossier app**

wwwroot / Index.html

```
<html>
<head>
  <title>SMTemplate Angular2 & ASP.NET Core</title>
  <base href="/">
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1">

  <script src="node_modules/core-js/client/shim.min.js"></script>

  <script src="node_modules/zone.js/dist/zone.js"></script>
  <script src="node_modules/systemjs/dist/system.src.js"></script>
```

```

<script src="systemjs.config.js"></script>
<script>
  System.import('main.js').catch(function(err){ console.error(err); });
</script>
</head>

<body>
  <my-app>Loading AppComponent here ...</my-app>
</body>
</html>

```

Vous pouvez appeler comme cela pour les contrôleurs à partir de templateUrl. `wwwroot / app / app.component.ts`

```

import { Component } from '@angular/core';

@Component({
  selector: 'my-app',
  templateUrl: '/home/index',
})
export class AppComponent { nombre = 'Samuel Maïcas'; }

```

[Graine] Asp.Net Core + Angular2 + Gulp sur Visual Studio 2017

1. Télécharger la graine
2. Exécuter la restauration dotnet
3. Exécutez npm install

Toujours. Prendre plaisir.

<https://github.com/SamML/CoreAngular000>

MVC <-> Angular 2

Comment: APPELER ANGULAIRE 2 COMPOSANTES HTML / JS À PARTIR DU CONTRÔLEUR ASP.NET Core:

Nous appelons le HTML à la place de retour Voir ()

```
return File("~/html/About.html", "text/html");
```

Et charger la composante angulaire dans le HTML. Ici, nous pouvons décider si nous voulons travailler avec un module identique ou différent. Dépend de la situation

`wwwroot / html / About.html`

```

<!DOCTYPE html>
<html>
  <head>
    <title>About Page</title>
    <base href="/">

```

```

<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1">
  <link href="../../css/site.min.css" rel="stylesheet" type="text/css"/>

<script src="../../node_modules/core-js/client/shim.min.js"></script>

<script src="../../node_modules/zone.js/dist/zone.js"></script>
<script src="../../node_modules/systemjs/dist/system.src.js"></script>

<script src="../../systemjs.config.js"></script>
<script>
  System.import('../main.js').catch(function(err){ console.error(err); });
</script>
</head>

<body>
  <aboutpage>Loading AppComponent here ...</aboutpage>
</body>
</html>

```

(*) Cette graine doit déjà charger la liste complète des ressources

Comment: APPELER ASP.NET Core Controller pour afficher une vue MVC avec prise en charge Angular2:

```

import { Component } from '@angular/core';

@Component({
  selector: 'aboutpage',
  templateUrl: '/home/about',
})
export class AboutComponent {

}

```

Lire Configuration de l'application ASP.net Core pour qu'elle fonctionne avec Angular 2 et TypeScript en ligne: <https://riptutorial.com/fr/angular2/topic/9543/configuration-de-l-application-asp-net-core-pour-qu-elle-fonctionne-avec-angular-2-et-typescript>

Chapitre 27: couverture de test angulaire-cli

Introduction

La couverture de test est définie comme une technique qui détermine si nos scénarios de test couvrent réellement le code de l'application et combien de code est exercé lorsque nous exécutons ces cas de test.

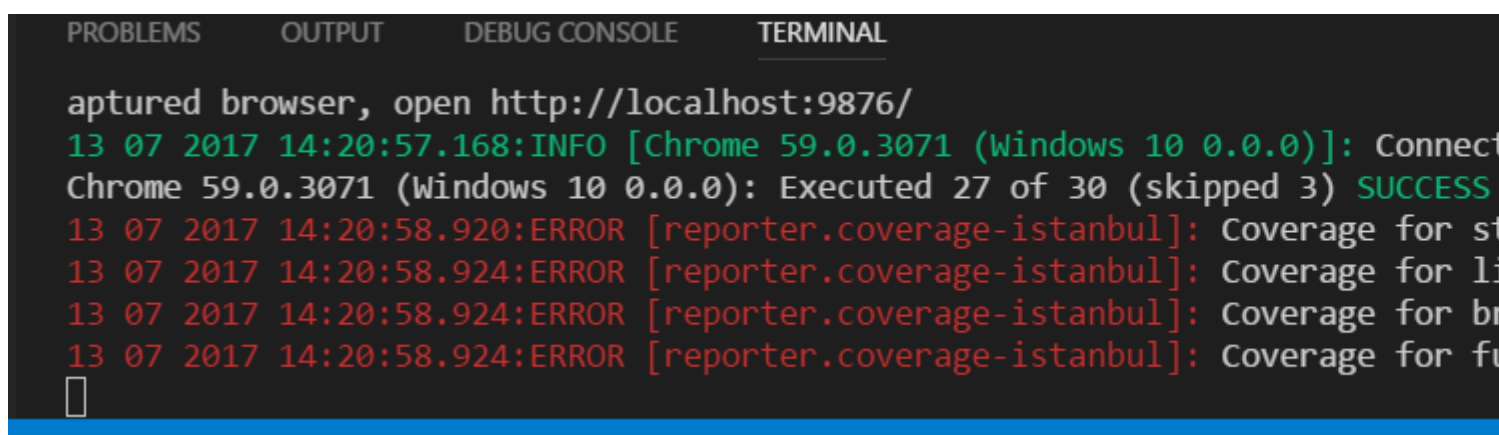
CLI angulaire a intégré une fonctionnalité de couverture de code avec une simple commande `ng test --cc`

Exemples

Une simple couverture de test de base de commande angulaire-cli

Si vous voulez voir des statistiques de couverture de test globales bien sûr dans Angular CLI, vous pouvez simplement saisir la commande ci-dessous et voir les résultats en bas de votre fenêtre d'invite de commande.

```
ng test --cc // or --code-coverage
```



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
aptured browser, open http://localhost:9876/
13 07 2017 14:20:57.168:INFO [Chrome 59.0.3071 (Windows 10 0.0.0)]: Connect
Chrome 59.0.3071 (Windows 10 0.0.0): Executed 27 of 30 (skipped 3) SUCCESS
13 07 2017 14:20:58.920:ERROR [reporter.coverage-istanbul]: Coverage for st
13 07 2017 14:20:58.924:ERROR [reporter.coverage-istanbul]: Coverage for li
13 07 2017 14:20:58.924:ERROR [reporter.coverage-istanbul]: Coverage for br
13 07 2017 14:20:58.924:ERROR [reporter.coverage-istanbul]: Coverage for fu
```

Rapport détaillé de couverture de test graphique des composants individuels

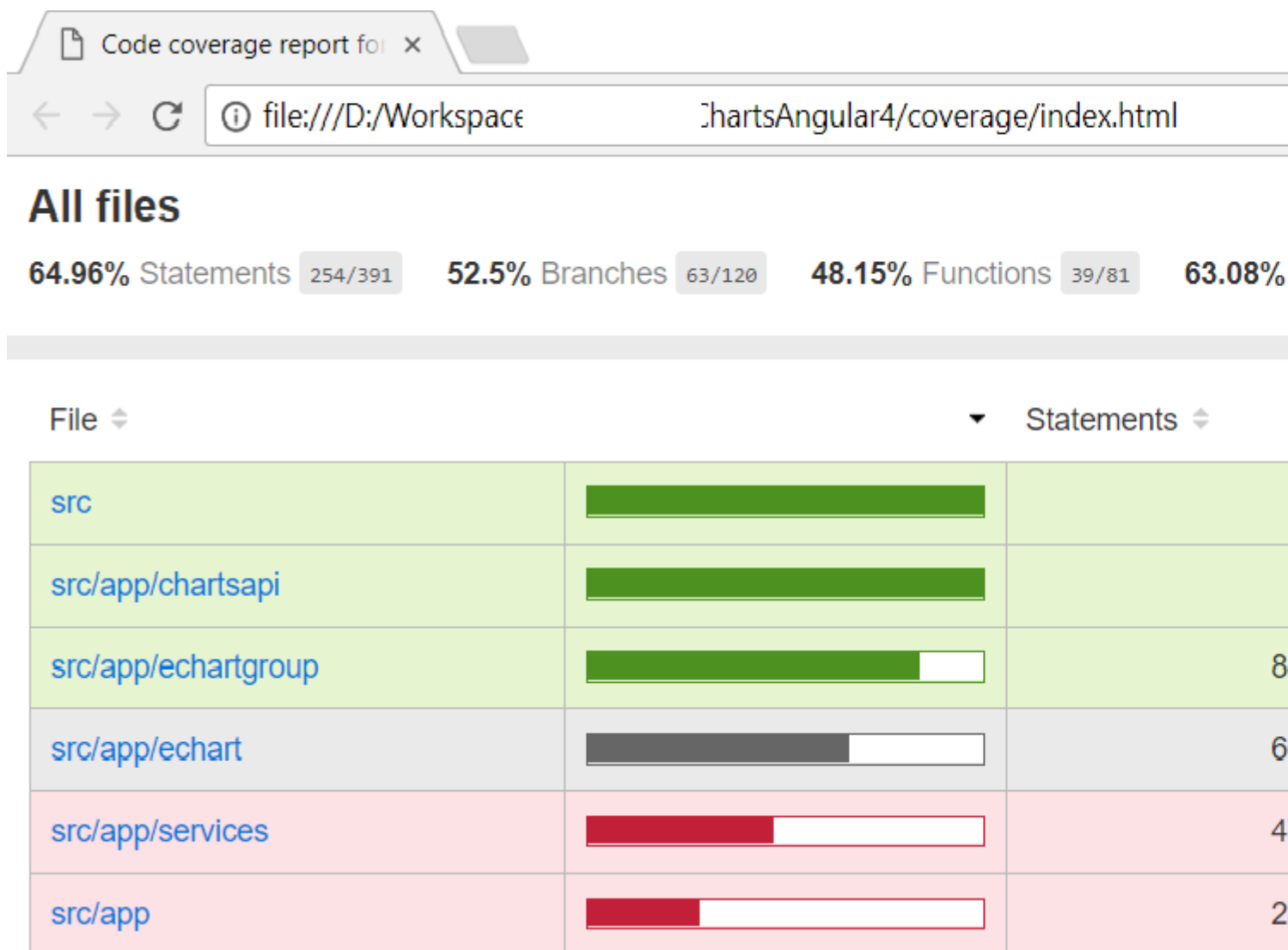
Si vous voulez voir la couverture individuelle des tests des composants, suivez ces étapes.

1. `npm install --save-dev karma-teamcity-reporter`
2. Add ``require('karma-teamcity-reporter')`` to list of plugins in `karma.conf.js`
3. `ng test --code-coverage --reporters=teamcity,coverage-istanbul`

Notez que la liste des journalistes est séparée par des virgules, car nous avons ajouté un nouveau journaliste, `teamcity`.

Après avoir exécuté cette commande, vous pouvez voir la `coverage` du dossier dans votre

répertoire et ouvrir `index.html` pour obtenir une vue graphique de la couverture du test.



Vous pouvez également définir le seuil de couverture que vous souhaitez atteindre, dans `karma.conf.js`, comme ceci.

```
coverageIstanbulReporter: {
  reports: ['html', 'lcovonly'],
  fixWebpackSourcePaths: true,
  thresholds: {
    statements: 90,
    lines: 90,
    branches: 90,
    functions: 90
  }
},
```

Lire couverture de test angulaire-cli en ligne:

<https://riptutorial.com/fr/angular2/topic/10764/couverture-de-test-angulaire-cli>

Chapitre 28: Créer un paquet Angular 2+ NPM

Introduction

Parfois, nous devons partager certains composants entre certaines applications et le publier en npm est l'une des meilleures façons de le faire.

Il y a quelques astuces que nous devons savoir pour pouvoir utiliser un composant normal en tant que paquet npm sans changer la structure en insérant des styles externes.

Vous pouvez voir un exemple minimal [ici](#)

Exemples

Forfait le plus simple

Nous partageons ici un flux de travail minimal pour créer et publier un paquet Angular 2+ npm.

Fichiers de configuration

Nous avons besoin de fichiers de configuration pour indiquer à `git`, `npm`, `gulp` et `typescript` comment agir.

`.gitignore`

Tout d'abord, nous créons un fichier `.gitignore` pour éviter de `.gitignore` les fichiers et dossiers indésirables. Le contenu est:

```
npm-debug.log
node_modules
jspm_packages
.idea
build
```

`.npmignore`

Deuxièmement, nous créons un fichier `.npmignore` pour éviter de publier des fichiers et des dossiers indésirables. Le contenu est:

```
examples
node_modules
src
```

`gulpfile.js`

Nous devons créer un `gulpfile.js` pour indiquer à Gulp comment compiler notre application. Cette partie est nécessaire car nous devons minimiser et intégrer tous les modèles et styles externes avant de publier notre package. Le contenu est:

```
var gulp = require('gulp');
var embedTemplates = require('gulp-angular-embed-templates');
var inlineNg2Styles = require('gulp-inline-ng2-styles');

gulp.task('js:build', function () {
  gulp.src('src/*.ts') // also can use *.js files
    .pipe(embedTemplates({sourceType:'ts'}))
    .pipe(inlineNg2Styles({ base: '/src' }))
    .pipe(gulp.dest('./dist'));
});
```

index.d.ts

Le fichier `index.d.ts` est utilisé par typescript lors de l'importation d'un module externe. Il aide l'éditeur avec l'auto-complétion et les conseils de fonction.

```
export * from './lib';
```

index.js

C'est le point d'entrée du paquet. Lorsque vous installez ce package à l'aide de NPM et que vous l'importez dans votre application, il vous suffit de passer le nom du package et votre application apprend où trouver tout composant EXPORTED de votre package.

```
exports.AngularXMinimalNpmPackageModule = require('./lib').AngularXMinimalNpmPackageModule;
```

Nous avons utilisé le dossier `lib` car lorsque nous compilons notre code, la sortie est placée dans le dossier `/lib`.

package.json

Ce fichier est utilisé pour configurer votre publication npm et définit les paquets nécessaires à son fonctionnement.

```
{
  "name": "angular-x-minimal-npm-package",
  "version": "0.0.18",
  "description": "An Angular 2+ Data Table that uses HTTP to create, read, update and delete data from an external API such REST.",
  "main": "index.js",
  "scripts": {
    "watch": "tsc -p src -w",
    "build": "gulp js:build && rm -rf lib && tsc -p dist"
  },
  "repository": {
    "type": "git",
    "url": "git+https://github.com/vinagreti/angular-x-minimal-npm-package.git"
  }
}
```



```

},
"keywords": [
  "Angular",
  "Angular2",
  "Datatable",
  "Rest"
],
"author": "bruno@tzadi.com",
"license": "MIT",
"bugs": {
  "url": "https://github.com/vinagreti/angular-x-minimal-npm-package/issues"
},
"homepage": "https://github.com/vinagreti/angular-x-minimal-npm-package#readme",
"devDependencies": {
  "gulp": "3.9.1",
  "gulp-angular-embed-templates": "2.3.0",
  "gulp-inline-ng2-styles": "0.0.1",
  "typescript": "2.0.0"
},
"dependencies": {
  "@angular/common": "2.4.1",
  "@angular/compiler": "2.4.1",
  "@angular/core": "2.4.1",
  "@angular/http": "2.4.1",
  "@angular/platform-browser": "2.4.1",
  "@angular/platform-browser-dynamic": "2.4.1",
  "rxjs": "5.0.2",
  "zone.js": "0.7.4"
}
}

```

dist / tsconfig.json

Créez un dossier dist et placez ce fichier à l'intérieur. Ce fichier est utilisé pour indiquer à Typescript comment compiler votre application. Où trouver le dossier dactylographié et où placer les fichiers compilés.

```

{
  "compilerOptions": {
    "emitDecoratorMetadata": true,
    "experimentalDecorators": true,
    "mapRoot": "",
    "rootDir": ".",
    "target": "es5",
    "lib": ["es6", "es2015", "dom"],
    "inlineSources": true,
    "stripInternal": true,
    "module": "commonjs",
    "moduleResolution": "node",
    "removeComments": true,
    "sourceMap": true,
    "outDir": "../lib",
    "declaration": true
  }
}

```

Après avoir créé les fichiers de configuration, nous devons créer notre composant et notre

module. Ce composant reçoit un clic et affiche un message. Il est utilisé comme une balise html `<angular-x-minimal-npm-package></angular-x-minimal-npm-package>` . Installez simplement ce paquet npm et chargez son module dans le modèle que vous souhaitez utiliser.

src / angular-x-minimal-npm-package.component.ts

```
import {Component} from '@angular/core';
@Component({
  selector: 'angular-x-minimal-npm-package',
  styleUrls: ['./angular-x-minimal-npm-package.component.scss'],
  templateUrl: './angular-x-minimal-npm-package.component.html'
})
export class AngularXMinimalNpmPackageComponent {
  message = "Click Me ...";
  onClick() {
    this.message = "Angular 2+ Minimal NPM Package. With external scss and html!";
  }
}
```

src / angular-x-minimal-npm-package.component.html

```
<div>
  <h1 (click)="onClick()">{{message}}</h1>
</div>
```

src / angular-x-data-table.component.css

```
h1{
  color: red;
}
```

src / angular-x-minimal-npm-package.module.ts

```
import { NgModule } from '@angular/core';
import { CommonModule } from '@angular/common';

import { AngularXMinimalNpmPackageComponent } from './angular-x-minimal-npm-
package.component';

@NgModule({
  imports: [ CommonModule ],
  declarations: [ AngularXMinimalNpmPackageComponent ],
  exports: [ AngularXMinimalNpmPackageComponent ],
  entryComponents: [ AngularXMinimalNpmPackageComponent ],
})
export class AngularXMinimalNpmPackageModule {}
```

Après cela, vous devez compiler, compiler et publier votre package.

Construire et compiler

Pour la construction, nous utilisons `gulp` et pour la compilation, nous utilisons `tsc`. La commande est définie dans le fichier `package.json`, à l'option `scripts.build`. Nous avons cet ensemble `gulp js:build && rm -rf lib && tsc -p dist`. Ce sont nos tâches de chaîne qui feront le travail pour nous.

Pour générer et compiler, exécutez la commande suivante à la racine de votre package:

```
npm run build
```

Cela déclenchera la chaîne et vous vous retrouverez avec votre dossier `build` `/dist` et le paquet compilé dans votre dossier `/lib`. C'est pourquoi, dans `index.js` nous avons exporté le code du dossier `/lib` et non de `/src`.

Publier

Maintenant, nous avons juste besoin de publier notre paquet pour que nous puissions l'installer jusqu'à npm. Pour cela, lancez simplement la commande:

```
npm publish
```

C'est tout!!!

Lire [Créer un paquet Angular 2+ NPM en ligne](https://riptutorial.com/fr/angular2/topic/8790/creer-un-paquet-angular-2plus-npm): <https://riptutorial.com/fr/angular2/topic/8790/creer-un-paquet-angular-2plus-npm>

Chapitre 29: Créer une bibliothèque Angular npm

Introduction

Comment publier votre NgModule, écrit en TypeScript dans le registre npm. Mise en place du projet npm, du compilateur de typescript, du rollup et de la construction d'une intégration continue.

Exemples

Module minimal avec classe de service

Structure de fichier

```
/
  -src/
    awesome.service.ts
    another-awesome.service.ts
    awesome.module.ts
  -index.ts
  -tsconfig.json
  -package.json
  -rollup.config.js
  -.npmignore
```

Service et module

Placez votre travail impressionnant ici.

src / awesome.service.ts:

```
export class AwesomeService {
  public doSomethingAwesome(): void {
    console.log("I am so awesome!");
  }
}
```

src / awesome.module.ts:

```
import { NgModule } from '@angular/core'
import { AwesomeService } from './awesome.service';
import { AnotherAwesomeService } from './another-awesome.service';

@NgModule({
  providers: [AwesomeService, AnotherAwesomeService]
```

```
)  
export class AwesomeModule {}
```

Rendez votre module et votre service accessibles à l'extérieur.

/index.ts:

```
export { AwesomeService } from './src/awesome.service';  
export { AnotherAwesomeService } from './src/another-awesome.service';  
export { AwesomeModule } from './src/awesome.module';
```

Compilation

Dans `compilerOptions.paths`, vous devez spécifier tous les modules externes que vous avez utilisés dans votre package.

/tsconfig.json

```
{  
  "compilerOptions": {  
    "baseUrl": ".",  
    "declaration": true,  
    "stripInternal": true,  
    "experimentalDecorators": true,  
    "strictNullChecks": false,  
    "noImplicitAny": true,  
    "module": "es2015",  
    "moduleResolution": "node",  
    "paths": {  
      "@angular/core": ["node_modules/@angular/core"],  
      "rxjs/*": ["node_modules/rxjs/*"]  
    },  
    "rootDir": ".",  
    "outDir": "dist",  
    "sourceMap": true,  
    "inlineSources": true,  
    "target": "es5",  
    "skipLibCheck": true,  
    "lib": [  
      "es2015",  
      "dom"  
    ]  
  },  
  "files": [  
    "index.ts"  
  ],  
  "angularCompilerOptions": {  
    "strictMetadataEmit": true  
  }  
}
```

Spécifiez à nouveau vos externes

/rollup.config.js

```

export default {
  entry: 'dist/index.js',
  dest: 'dist/bundles/awesome.module.umd.js',
  sourceMap: false,
  format: 'umd',
  moduleName: 'ng.awesome.module',
  globals: {
    '@angular/core': 'ng.core',
    'rxjs': 'Rx',
    'rxjs/Observable': 'Rx',
    'rxjs/ReplaySubject': 'Rx',
    'rxjs/add/operator/map': 'Rx.Observable.prototype',
    'rxjs/add/operator/mergeMap': 'Rx.Observable.prototype',
    'rxjs/add/observable/fromEvent': 'Rx.Observable',
    'rxjs/add/observable/of': 'Rx.Observable'
  },
  external: ['@angular/core', 'rxjs']
}

```

Paramètres NPM

Maintenant, laisse quelques instructions pour npm

/package.json

```

{
  "name": "awesome-angular-module",
  "version": "1.0.4",
  "description": "Awesome angular module",
  "main": "dist/bundles/awesome.module.umd.min.js",
  "module": "dist/index.js",
  "typings": "dist/index.d.ts",
  "scripts": {
    "test": "",
    "transpile": "ngc",
    "package": "rollup -c",
    "minify": "uglifyjs dist/bundles/awesome.module.umd.js --screw-ie8 --compress --mangle --
comments --output dist/bundles/awesome.module.umd.min.js",
    "build": "rimraf dist && npm run transpile && npm run package && npm run minify",
    "prepublishOnly": "npm run build"
  },
  "repository": {
    "type": "git",
    "url": "git+https://github.com/maciejtreder/awesome-angular-module.git"
  },
  "keywords": [
    "awesome",
    "angular",
    "module",
    "minimal"
  ],
  "author": "Maciej Treder <contact@maciejtreder.com>",
  "license": "MIT",
  "bugs": {
    "url": "https://github.com/maciejtreder/awesome-angular-module/issues"
  },
  "homepage": "https://github.com/maciejtreder/awesome-angular-module#readme",

```

```
"devDependencies": {
  "@angular/compiler": "^4.0.0",
  "@angular/compiler-cli": "^4.0.0",
  "rimraf": "^2.6.1",
  "rollup": "^0.43.0",
  "typescript": "^2.3.4",
  "uglify-js": "^3.0.21"
},
"dependencies": {
  "@angular/core": "^4.0.0",
  "rxjs": "^5.3.0"
}
}
```

Nous pouvons également spécifier quels fichiers, npm devraient ignorer

.npmignore

```
node_modules
npm-debug.log
Thumbs.db
.DS_Store
src
!dist/src
plugin
!dist/plugin
*.ngsummary.json
*.iml
rollup.config.js
tsconfig.json
*.ts
!*d.ts
.idea
```

Intégration continue

Enfin, vous pouvez configurer une intégration continue

.travis.yml

```
language: node_js
node_js:
- node

deploy:
  provider: npm
  email: contact@maciejtreder.com
  api_key:
    secure: <your api key>
  on:
    tags: true
    repo: maciejtreder/awesome-angular-module
```

La démo peut être trouvée ici: <https://github.com/maciejtreder/awesome-angular-module>

Lire Créer une bibliothèque Angular npm en ligne:

<https://riptutorial.com/fr/angular2/topic/10704/creer-une-bibliotheque-angular-npm>

Chapitre 30: Crochets de cycle de vie

Remarques

Disponibilité des événements

`AfterViewInit` et `AfterViewChecked` ne sont disponibles que **dans les composants** et **non dans les directives** .

Ordre des événements

- `OnChanges` (plusieurs fois)
- `OnInit` (une fois)
- `DoCheck` (plusieurs fois)
- `AfterContentInit` (une fois)
- `AfterContentChecked` (plusieurs fois)
- `AfterViewInit` (une fois) (composant uniquement)
- `AfterViewChecked` (plusieurs fois) (Composant uniquement)
- `OnDestroy` (une fois)

Lectures complémentaires

- [Documentation angulaire - Crochets de cycle de vie](#)

Exemples

OnInit

Fired lorsque les propriétés du composant ou de la directive ont été initialisées.

(Avant ceux des directives enfants)

```
import { Component, OnInit } from '@angular/core';

@Component({
  selector: 'so-oninit-component',
  templateUrl: 'oninit-component.html',
  styleUrls: ['oninit-component.']
})
class OnInitComponent implements OnInit {

  ngOnInit(): void {
    console.log('Component is ready !');
  }
}
```

```
}
```

OnDestroy

Déclenché lorsque l'instance du composant ou de la directive est détruite.

```
import { Component, OnDestroy } from '@angular/core';

@Component({
  selector: 'so-ondestroy-component',
  templateUrl: 'ondestroy-component.html',
  styleUrls: ['ondestroy-component.Styles.css']
})
class OnDestroyComponent implements OnDestroy {

  ngOnDestroy(): void {
    console.log('Component was destroyed !');
  }
}
```

Modifications

Déclenché quand une ou plusieurs des propriétés du composant ou de la directive ont été modifiées.

```
import { Component, OnChanges, Input } from '@angular/core';

@Component({
  selector: 'so-onchanges-component',
  templateUrl: 'onchanges-component.html',
  styleUrls: ['onchanges-component.Styles.css']
})
class OnChangesComponent implements OnChanges {
  @Input() name: string;
  message: string;

  ngOnChanges(changes: SimpleChanges): void {
    console.log(changes);
  }
}
```

L'événement de modification se connectera

```
name: {
  currentValue: 'new name value',
  previousValue: 'old name value'
},
message: {
  currentValue: 'new message value',
  previousValue: 'old message value'
}
```

AfterContentInit

Le feu après l'initialisation du contenu du composant ou de la directive est terminé.

(Juste après OnInit)

```
import { Component, AfterContentInit } from '@angular/core';

@Component({
  selector: 'so-aftercontentinit-component',
  templateUrl: 'aftercontentinit-component.html',
  styleUrls: ['aftercontentinit-component.'],
})
class AfterContentInitComponent implements AfterContentInit {

  ngAfterContentInit(): void {
    console.log('Component content have been loaded!');
  }
}
```

AfterContentChecked

Feu après que la vue a été complètement initialisée.

(Disponible uniquement pour les composants)

```
import { Component, AfterContentChecked } from '@angular/core';

@Component({
  selector: 'so-aftercontentchecked-component',
  templateUrl: 'aftercontentchecked-component.html',
  styleUrls: ['aftercontentchecked-component.'],
})
class AfterContentCheckedComponent implements AfterContentChecked {

  ngAfterContentChecked(): void {
    console.log('Component content have been checked!');
  }
}
```

AfterViewInit

Se déclenche après l'initialisation de la vue du composant et de ses vues enfants. Ceci est un crochet de cycle de vie utile pour les plugins en dehors de l'écosystème Angular 2. Par exemple, vous pouvez utiliser cette méthode pour initialiser un sélecteur de date jQuery basé sur le balisage rendu par Angular 2.

```
import { Component, AfterViewInit } from '@angular/core';

@Component({
  selector: 'so-afterviewinit-component',
  templateUrl: 'afterviewinit-component.html',
  styleUrls: ['afterviewinit-component.'],
})
class AfterViewInitComponent implements AfterViewInit {

  ngAfterViewInit(): void {
```

```
        console.log('This event fire after the content init have been loaded!');
    }
}
```

AfterViewChecked

Le feu après la vérification de la vue, du composant, est terminé.

(Disponible uniquement pour les composants)

```
import { Component, AfterViewChecked } from '@angular/core';

@Component({
  selector: 'so-afterviewchecked-component',
  templateUrl: 'afterviewchecked-component.html',
  styleUrls: ['afterviewchecked-component.Styles.css']
})
class AfterViewCheckedComponent implements AfterViewChecked {

  ngAfterViewChecked(): void {
    console.log('This event fire after the content have been checked!');
  }
}
```

DoCheck

Permet d'écouter les modifications uniquement sur les propriétés spécifiées

```
import { Component, DoCheck, Input } from '@angular/core';

@Component({
  selector: 'so-docheck-component',
  templateUrl: 'docheck-component.html',
  styleUrls: ['docheck-component.Styles.css']
})
class DoCheckComponent implements DoCheck {
  @Input() elements: string[];
  differ: any;
  ngDoCheck(): void {
    // get value for elements property
    const changes = this.differ.diff(this.elements);

    if (changes) {
      changes.forEachAddedItem(res => console.log('Added', res.item));
      changes.forEachRemovedItem(r => console.log('Removed', r.item));
    }
  }
}
```

Lire Crochets de cycle de vie en ligne: <https://riptutorial.com/fr/angular2/topic/2935/crochets-de-cycle-de-vie>

Chapitre 31: CRUD dans Angular2 avec API Restful

Syntaxe

- `@Injectable ()` // Indique à l'injecteur de dépendance d'injecter des dépendances lors de la création de l'instance de ce service.
- `request.subscribe ()` // C'est là que vous faites en *fait* la demande. Sans cela, votre demande ne sera pas envoyée. Vous souhaitez également lire la réponse dans la fonction de rappel.
- `constructeur (wikiService privé: WikipediaService) {}` // Étant donné que notre service et ses dépendances peuvent être injectés par l'injecteur de dépendance, il est conseillé d'injecter le service dans le composant pour que l'unité teste l'application.

Exemples

Lecture d'une API Restful dans Angular2

Pour séparer la logique API du composant, nous créons le client API en tant que classe distincte. Cet exemple de classe demande à l'API Wikipedia d'obtenir des articles wiki aléatoires.

```
import { Http, Response } from '@angular/http';
import { Injectable } from '@angular/core';
import { Observable } from 'rxjs/Observable';
import 'rxjs/Rx';

@Injectable()
export class WikipediaService{
  constructor(private http: Http) {}

  getRandomArticles(numberOfArticles: number)
  {
    var request =
this.http.get("https://en.wikipedia.org/w/api.php?action=query&list=random&format=json&rnlimit="
+ numberOfArticles);
    return request.map((response: Response) => {
      return response.json();
    }, (error) => {
      console.log(error);
      //your want to implement your own error handling here.
    });
  }
}
```

Et avoir un composant pour consommer notre nouveau client API.

```
import { Component, OnInit } from '@angular/core';
import { WikipediaService } from './wikipedia.Service';
```

```
@Component({
  selector: 'wikipedia',
  templateUrl: 'wikipedia.component.html'
})
export class WikipediaComponent implements OnInit {
  constructor(private wikiService: WikipediaService) { }

  private articles: any[] = null;
  ngOnInit() {
    var request = this.wikiService.getRandomArticles(5);
    request.subscribe((res) => {
      this.articles = res.query.random;
    });
  }
}
```

Lire CRUD dans Angular2 avec API Restful en ligne:

<https://riptutorial.com/fr/angular2/topic/7343/crud-dans-angular2-avec-api-restful>

Chapitre 32: Débogage de l'application typographique Angular2 à l'aide du code Visual Studio

Exemples

Configuration de Launch.json pour votre espace de travail

1. Activer le débogage à partir du menu - afficher> déboguer
2. Il retourne une erreur lors du démarrage du débogage, affiche la notification pop-out et ouvre launch.json à partir de cette notification contextuelle. C'est simplement à cause de launch.json qui n'est pas défini pour votre espace de travail. Copiez et collez le code ci-dessous dans launch.json // new launch.json
voire ancien lancement.json

```
{
  "version": "0.2.0",
  "configurations": [
    {
      "name": "Launch Extension",
      "type": "extensionHost",
      "request": "launch",
      "runtimeExecutable": "${execPath}",
      "args": [
        "--extensionDevelopmentPath=${workspaceRoot}"
      ],
      "stopOnEntry": false,
      "sourceMaps": true,
      "outDir": "${workspaceRoot}/out",
      "preLaunchTask": "npm"
    }
  ]
}
```

Maintenant, mettez à jour votre launch.json comme ci-dessous
nouveau launch.json

**** // rappelez-vous s'il vous plaît mentionner votre chemin main.js dedans ****

```
{
  "version": "0.2.0",
  "configurations": [
    {
      "name": "Launch",
      "type": "node",
      "request": "launch",
      "program": "${workspaceRoot}/app/main.js", // put your main.js path
      "stopOnEntry": false,
```

```

    "args": [],
    "cwd": "${workspaceRoot}",
    "preLaunchTask": null,
    "runtimeExecutable": null,
    "runtimeArgs": [
        "--nolazy"
    ],
    "env": {
        "NODE_ENV": "development"
    },
    "console": "internalConsole",
    "sourceMaps": false,
    "outDir": null
},
{
    "name": "Attach",
    "type": "node",
    "request": "attach",
    "port": 5858,
    "address": "localhost",
    "restart": false,
    "sourceMaps": false,
    "outDir": null,
    "localRoot": "${workspaceRoot}",
    "remoteRoot": null
},
{
    "name": "Attach to Process",
    "type": "node",
    "request": "attach",
    "processId": "${command.PickProcess}",
    "port": 5858,
    "sourceMaps": false,
    "outDir": null
}
]
}

```

3. Maintenant que le débogage fonctionne, afficher la notification contextuelle pour le débogage pas à pas

Lire [Débogage de l'application typographique Angular2 à l'aide du code Visual Studio en ligne](https://riptutorial.com/fr/angular2/topic/7139/debogage-de-l-application-typographique-angular2-a-l-aide-du-code-visual-studio):
<https://riptutorial.com/fr/angular2/topic/7139/debogage-de-l-application-typographique-angular2-a-l-aide-du-code-visual-studio>

Chapitre 33: Détection des événements de redimensionnement

Exemples

Un composant écoutant la fenêtre redimensionne l'événement.

Supposons que nous ayons un composant qui se cachera à une certaine largeur de fenêtre.

```
import { Component } from '@angular/core';

@Component({
  ...
  template: `
    <div>
      <p [hidden]="!visible" (window:resize)="onResize($event)" >Now you see me...</p>
      <p>now you dont!</p>
    </div>
  `
  ...
})
export class MyComponent {
  visible: boolean = false;
  breakpoint: number = 768;

  constructor() {
  }

  onResize(event) {
    const w = event.target.innerWidth;
    if (w >= this.breakpoint) {
      this.visible = true;
    } else {
      // whenever the window is less than 768, hide this component.
      this.visible = false;
    }
  }
}
```

Une balise `p` dans notre modèle se cachera lorsque `visible` est fausse. `visible` changera la valeur chaque fois que le gestionnaire d'événement `onResize` est appelé. Son appel se produit à chaque `window:resize` lorsque `window:resize` déclenche un événement.

Lire [Détection des événements de redimensionnement en ligne](https://riptutorial.com/fr/angular2/topic/5276/detection-des-evenements-de-redimensionnement):

<https://riptutorial.com/fr/angular2/topic/5276/detection-des-evenements-de-redimensionnement>

Chapitre 34: Directives

Syntaxe

- `<input [value]="value">` - Lie le `name` membre de la classe d'attribut.
- `<div [attr.data-note]="note">` - Lie l'attribut `data-note` à la `note` variable.
- `<p green></p>` - Directive personnalisée

Remarques

La principale source d'informations sur les directives Angular 2 est la documentation officielle <https://angular.io/docs/ts/latest/guide/attribute-directives.html>

Exemples

Directive attribut

```
<div [class.active]="isActive"></div>

<span [style.color]='red'></span>

<p [attr.data-note]='This is value for data-note attribute'>A lot of text here</p>
```

Le composant est une directive avec un modèle

```
import { Component } from '@angular/core';
@Component({
  selector: 'my-app',
  template: `
    <h1>Angular 2 App</h1>
    <p>Component is directive with template</p>
  `
})
export class AppComponent {
}
```

Directives structurelles

```
<div *ngFor="let item of items">{{ item.description }}</div>

<span *ngIf="isVisible"></span>
```

Directive personnalisée

```
import {Directive, ElementRef, Renderer} from '@angular/core';

@Directive({
  selector: '[green]',
})

class GreenDirective {
  constructor(private _elementRef: ElementRef,
              private _renderer: Renderer) {
    _renderer.setStyle(_elementRef.nativeElement, 'color', 'green');
  }
}
```

Usage:

```
<p green>A lot of green text here</p>
```

* ngPour

form1.component.ts:

```
import { Component } from '@angular/core';

// Defines example component and associated template
@Component({
  selector: 'example',
  template: `
    <div *ngFor="let f of fruit"> {{f}} </div>
    <select required>
      <option *ngFor="let f of fruit" [value]="f"> {{f}} </option>
    </select>
  `
})

// Create a class for all functions, objects, and variables
export class ExampleComponent {
  // Array of fruit to be iterated by *ngFor
  fruit = ['Apples', 'Oranges', 'Bananas', 'Limes', 'Lemons'];
}
```

Sortie:

```
<div>Apples</div>
<div>Oranges</div>
<div>Bananas</div>
<div>Limes</div>
<div>Lemons</div>
<select required>
  <option value="Apples">Apples</option>
  <option value="Oranges">Oranges</option>
  <option value="Bananas">Bananas</option>
  <option value="Limes">Limes</option>
  <option value="Lemons">Lemons</option>
</select>
```

Dans sa forme la plus simple, `*ngFor` a deux parties: `let variableName of object/array`

Dans le cas des `fruit = ['Apples', 'Oranges', 'Bananas', 'Limes', 'Lemons'];`,

Pommes, oranges, etc. sont les valeurs à l'intérieur du tableau de `fruit`.

`[value]="f"` sera égal à chaque `fruit` courant (`f`) sur lequel `*ngFor` a été `*ngFor`.

Contrairement à AngularJS, Angular2 n'a pas continué à utiliser `ng-options` pour `<select>` et `ng-repeat` pour toutes les autres répétitions générales.

`*ngFor` est très similaire à `ng-repeat` avec une syntaxe légèrement variée.

Les références:

Angular2 | [Affichage des données](#)

Angular2 | [ngpour](#)

Angular2 | [Formes](#)

Directive Copier dans le Presse-papiers

Dans cet exemple, nous allons créer une directive pour copier un texte dans le presse-papier en cliquant sur un élément.

copy-text.directive.ts

```
import {
  Directive,
  Input,
  HostListener
} from "@angular/core";

@Directive({
  selector: '[text-copy]'
})
export class TextCopyDirective {

  // Parse attribute value into a 'text' variable
  @Input('text-copy') text:string;

  constructor() {
  }

  // The HostListener will listen to click events and run the below function, the
  // HostListener supports other standard events such as mouseenter, mouseleave etc.
  @HostListener('click') copyText() {

    // We need to create a dummy textarea with the text to be copied in the DOM
    var textArea = document.createElement("textarea");

    // Hide the textarea from actually showing
    textArea.style.position = 'fixed';
    textArea.style.top = '-999px';
    textArea.style.left = '-999px';
```

```

    textArea.style.width = '2em';
    textArea.style.height = '2em';
    textArea.style.padding = '0';
    textArea.style.border = 'none';
    textArea.style.outline = 'none';
    textArea.style.boxShadow = 'none';
    textArea.style.background = 'transparent';

    // Set the texarea's content to our value defined in our [text-copy] attribute
    textArea.value = this.text;
    document.body.appendChild(textArea);

    // This will select the textarea
    textArea.select();

    try {
        // Most modern browsers support execCommand('copy'|'cut'|'paste'), if it doesn't
it should throw an error
        var successful = document.execCommand('copy');
        var msg = successful ? 'successful' : 'unsuccessful';
        // Let the user know the text has been copied, e.g toast, alert etc.
        console.log(msg);
    } catch (err) {
        // Tell the user copying is not supported and give alternative, e.g alert window
with the text to copy
        console.log('unable to copy');
    }

    // Finally we remove the textarea from the DOM
    document.body.removeChild(textArea);
}
}

export const TEXT_COPY_DIRECTIVES = [TextCopyDirective];

```

some-page.component.html

N'oubliez pas d'injecter TEXT_COPY_DIRECTIVES dans le tableau de directives de votre composant

```

...
<!-- Insert variable as the attribute's value, let textToBeCopied = 'http://facebook.com/'
-->
<button [text-copy]="textToBeCopied">Copy URL</button>
<button [text-copy]=" 'https://www.google.com/' ">Copy URL</button>
...

```

Tester une directive personnalisée

Étant donné une directive qui met en évidence le texte sur les événements de la souris

```

import { Directive, ElementRef, HostListener, Input } from '@angular/core';

@Directive({ selector: '[appHighlight]' })
export class HighlightDirective {
    @Input('appHighlight') // tslint:disable-line no-input-rename
    highlightColor: string;
}

```

```

constructor(private el: ElementRef) { }

@HostListener('mouseenter')
onMouseEnter() {
  this.highlight(this.highlightColor || 'red');
}

@HostListener('mouseleave')
onMouseLeave() {
  this.highlight(null);
}

private highlight(color: string) {
  this.el.nativeElement.style.backgroundColor = color;
}
}

```

Il peut être testé comme ça

```

import { ComponentFixture, ComponentFixtureAutoDetect, TestBed } from '@angular/core/testing';

import { Component } from '@angular/core';
import { HighlightDirective } from './highlight.directive';

@Component({
  selector: 'app-test-container',
  template: `
    <div>
      <span id="red" appHighlight>red text</span>
      <span id="green" [appHighlight]="'green'">green text</span>
      <span id="no">no color</span>
    </div>
  `
})
class ContainerComponent { }

const mouseEvents = {
  get enter() {
    const mouseenter = document.createEvent('MouseEvent');
    mouseenter.initEvent('mouseenter', true, true);
    return mouseenter;
  },
  get leave() {
    const mouseleave = document.createEvent('MouseEvent');
    mouseleave.initEvent('mouseleave', true, true);
    return mouseleave;
  },
};

describe('HighlightDirective', () => {
  let fixture: ComponentFixture<ContainerComponent>;
  let container: ContainerComponent;
  let element: HTMLElement;

  beforeEach(() => {
    TestBed.configureTestingModule({
      declarations: [ContainerComponent, HighlightDirective],
      providers: [
        { provide: ComponentFixtureAutoDetect, useValue: true },

```

```

    ],
  });

  fixture = TestBed.createComponent(ContainerComponent);
  // fixture.detectChanges(); // without the provider
  container = fixture.componentInstance;
  element = fixture.nativeElement;
});

it('should set background-color to empty when mouse leaves with directive without arguments', () => {
  const targetElement = <HTMLSpanElement>element.querySelector('#red');

  targetElement.dispatchEvent(mouseEvents.leave);
  expect(targetElement.style.backgroundColor).toEqual('');
});

it('should set background-color to empty when mouse leaves with directive with arguments', () => {
  const targetElement = <HTMLSpanElement>element.querySelector('#green');

  targetElement.dispatchEvent(mouseEvents.leave);
  expect(targetElement.style.backgroundColor).toEqual('');
});

it('should set background-color red with no args passed', () => {
  const targetElement = <HTMLSpanElement>element.querySelector('#red');

  targetElement.dispatchEvent(mouseEvents.enter);
  expect(targetElement.style.backgroundColor).toEqual('red');
});

it('should set background-color green when passing green parameter', () => {
  const targetElement = <HTMLSpanElement>element.querySelector('#green');

  targetElement.dispatchEvent(mouseEvents.enter);
  expect(targetElement.style.backgroundColor).toEqual('green');
});
});

```

Lire Directives en ligne: <https://riptutorial.com/fr/angular2/topic/2202/directives>

Chapitre 35: Directives d'attribut affectant la valeur des propriétés sur le noeud hôte à l'aide du décorateur @HostBinding.

Exemples

@HostBinding

Le décorateur @HostBinding nous permet de définir par programmation une valeur de propriété sur l'élément hôte de la directive. Il fonctionne de manière similaire à une liaison de propriété définie dans un modèle, sauf qu'il cible spécifiquement l'élément hôte. La liaison est vérifiée pour chaque cycle de détection des modifications afin de pouvoir changer dynamiquement si vous le souhaitez. Par exemple, disons que nous voulons créer une directive pour les boutons qui ajoutent dynamiquement une classe lorsque nous l'appuyons. Cela pourrait ressembler à quelque chose comme:

```
import { Directive, HostBinding, HostListener } from '@angular/core';

@Directive({
  selector: '[appButtonPress]'
})
export class ButtonPressDirective {
  @HostBinding('attr.role') role = 'button';
  @HostBinding('class.pressed') isPressed: boolean;

  @HostListener('mousedown') hasPressed() {
    this.isPressed = true;
  }
  @HostListener('mouseup') hasReleased() {
    this.isPressed = false;
  }
}
```

Notez que pour les deux cas d'utilisation de @HostBinding, nous transmettons une valeur de chaîne pour la propriété que nous voulons affecter. Si nous ne fournissons pas de chaîne au décorateur, le nom du membre de la classe sera utilisé à la place. Dans le premier @HostBinding, nous définissons statiquement l'attribut role sur button. Pour le deuxième exemple, la classe pressée sera appliquée lorsque isPressed est vrai

Lire Directives d'attribut affectant la valeur des propriétés sur le noeud hôte à l'aide du décorateur @HostBinding. en ligne: <https://riptutorial.com/fr/angular2/topic/9455/directives-d-attribut-affectant-la-valeur-des-proprietes-sur-le-noeud-hote-a-l-aide-du-decorateur--hostbinding->

Chapitre 36: Directives et composants: @Input @Output

Syntaxe

1. Liaison à sens unique du composant parent au composant imbriqué: [propertyName]
2. Liaison à sens unique d'un composant imbriqué au composant parent: (propertyName)
3. Liaison bidirectionnelle (notation de la boîte banane): [(propertyName)]

Exemples

Exemple de saisie

@input est utile pour lier des données entre des composants

Tout d'abord, importez-le dans votre composant.

```
import { Input } from '@angular/core';
```

Ensuite, ajoutez l'entrée en tant que propriété de votre classe de composant

```
@Input() car: any;
```

Disons que le sélecteur de votre composant est 'car-component', lorsque vous appelez le composant, ajoutez l'attribut 'car'

```
<car-component [car]="car"></car-component>
```

Maintenant, votre voiture est accessible en tant qu'attribut dans votre objet (this.car)

Exemple complet:

1. car.entity.ts

```
export class CarEntity {  
  constructor(public brand : string, public color : string) {  
  }  
}
```

2. car.component.ts

```
import { Component, Input } from '@angular/core';  
import { CarEntity } from "../car.entity";  
  
@Component({
```

```

    selector: 'car-component',
    template: require('./templates/car.html'),
  })

export class CarComponent {
  @Input() car: CarEntity;

  constructor() {
    console.log('gros');
  }
}

```

3. garage.component.ts

```

import { Component } from '@angular/core';
import { CarEntity } from "../car.entity";
import { CarComponent } from "../car.component";

@Component({
  selector: 'garage',
  template: require('./templates/garage.html'),
  directives: [CarComponent]
})

export class GarageComponent {
  public cars : Array<CarEntity>;

  constructor() {
    var carOne : CarEntity = new CarEntity('renault', 'blue');
    var carTwo : CarEntity = new CarEntity('fiat', 'green');
    var carThree : CarEntity = new CarEntity('citroen', 'yellow');
    this.cars = [carOne, carTwo, carThree];
  }
}

```

4. garage.html

```

<div *ngFor="let car of cars">
  <car-component [car]="car"></car-component>
</div>

```

5. car.html

```

<div>
  <span>{{ car.brand }}</span> |
  <span>{{ car.color }}</span>
</div>

```

Angular2 @Input et @Output dans un composant imbriqué

Une directive Button qui accepte un @Input() pour spécifier une limite de clic jusqu'à ce que le bouton soit désactivé. Le composant parent peut écouter un événement qui sera émis lorsque la limite de clic est atteinte via @Output :

```

import { Component, Input, Output, EventEmitter } from '@angular/core';

@Component({
  selector: 'limited-button',
  template: `<button (click)="onClick()"
              [disabled]="disabled">
              <ng-content></ng-content>
            </button>`,
  directives: []
})

export class LimitedButton {
  @Input() clickLimit: number;
  @Output() limitReached: EventEmitter<number> = new EventEmitter();

  disabled: boolean = false;

  private clickCount: number = 0;

  onClick() {
    this.clickCount++;
    if (this.clickCount === this.clickLimit) {
      this.disabled = true;
      this.limitReached.emit(this.clickCount);
    }
  }
}

```

Composant parent qui utilise la directive Button et alerte un message lorsque la limite de clics est atteinte:

```

import { Component } from '@angular/core';
import { LimitedButton } from './limited-button.component';

@Component({
  selector: 'my-parent-component',
  template: `<limited-button [clickLimit]="2"
                          (limitReached)="onLimitReached($event)">
              You can only click me twice
            </limited-button>`,
  directives: [LimitedButton]
})

export class MyParentComponent {
  onLimitReached(clickCount: number) {
    alert('Button disabled after ' + clickCount + ' clicks.');
```

Angular2 @Input avec des données asynchrones

Parfois, vous devez extraire des données de manière asynchrone avant de les transmettre à un composant enfant à utiliser. Si le composant enfant tente d'utiliser les données avant leur réception, une erreur est générée. Vous pouvez utiliser `ngOnChanges` pour détecter les modifications apportées à la fonction `@Input` s d'un composant et attendre qu'ils soient définis avant d'agir sur eux.

Composant parent avec appel asynchrone à un noeud final

```
import { Component, OnChanges, OnInit } from '@angular/core';
import { Http, Response } from '@angular/http';
import { ChildComponent } from './child.component';

@Component ({
  selector : 'parent-component',
  template : `
    <child-component [data]="asyncData"></child-component>
  `
})
export class ParentComponent {

  asyncData : any;

  constructor(
    private _http : Http
  ){}

  ngOnInit () {
    this._http.get('some.url')
      .map(this.extractData)
      .subscribe(this.handleData)
      .catch(this.handleError);
  }

  extractData (res:Response) {
    let body = res.json();
    return body.data || { };
  }

  handleData (data:any) {
    this.asyncData = data;
  }

  handleError (error:any) {
    console.error(error);
  }
}
```

Composant enfant ayant des données asynchrones en entrée

Ce composant enfant prend les données asynchrones en entrée. Par conséquent, il doit attendre que les données existent avant de les utiliser. Nous utilisons `ngOnChanges` qui se déclenche à chaque fois que l'entrée d'un composant change, vérifie si les données existent et les utilise si c'est le cas. Notez que le modèle pour l'enfant ne s'affiche pas si une propriété qui repose sur les données transmises n'est pas vraie.

```
import { Component, OnChanges, Input } from '@angular/core';

@Component ({
  selector : 'child-component',
  template : `
    <p *ngIf="doesDataExist">Hello child</p>
  `
})
export class ChildComponent {

  doesDataExist: boolean = false;

  @Input('data') data : any;

  // Runs whenever component @Inputs change
  ngOnChanges () {
    // Check if the data exists before using it
    if (this.data) {
      this.useData(data);
    }
  }

  // contrived example to assign data to reliesOnData
  useData (data) {
    this.doesDataExist = true;
  }
}
```

Lire Directives et composants: @Input @Output en ligne:

<https://riptutorial.com/fr/angular2/topic/3046/directives-et-composants---input--output>

Chapitre 37: Directives et services généralement intégrés

Introduction

@ angular / commun - directives et services communément requis @ angular / noyau - cadre de base angulaire

Exemples

Classe d'emplacement

L'emplacement est un service que les applications peuvent utiliser pour interagir avec l'URL d'un navigateur. Selon l'emplacement utilisé, Location restera dans le chemin de l'URL ou dans le segment de hachage de l'URL.

Location est responsable de la normalisation de l'URL par rapport à la base href de l'application.

```
import {Component} from '@angular/core';
import {Location} from '@angular/common';

@Component({
  selector: 'app-component'
})
class AppCmp {

  constructor(_location: Location) {

    //Changes the browsers URL to the normalized version of the given URL,
    //and pushes a new item onto the platform's history.
    _location.go('/foo');

  }

  backClicked() {
    //Navigates back in the platform's history.
    this._location.back();
  }

  forwardClicked() {
    //Navigates forward in the platform's history.
    this._location.back();
  }
}
```

AsyncPipe

Le canal asynchrone s'abonne à un objet Observable ou Promise et renvoie la dernière valeur émise. Lorsqu'une nouvelle valeur est émise, le tuyau asynchrone marque le composant à vérifier

pour les modifications. Lorsque le composant est détruit, le canal asynchrone se désabonne automatiquement pour éviter les fuites de mémoire potentielles.

```
@Component({
  selector: 'async-observable-pipe',
  template: '<div><code>observable|async</code>: Time: {{ time | async }}</div>'
})
export class AsyncObservablePipeComponent {
  time = new Observable<string>((observer: Subscriber<string>) => {
    setInterval(() => observer.next(new Date().toString()), 1000);
  });
}
```

Affichage de la version angulaire2 actuelle utilisée dans votre projet

Pour afficher la version actuelle, nous pouvons utiliser la **version** de @ package angulaire / core.

```
import { Component, VERSION } from '@angular/core';

@Component({
  selector: 'my-app',
  template: `<h1>Hello {{name}}</h1>
<h2>Current Version: {{ver}}</h2>
`,
})
export class AppComponent {
  name = 'Angular2';
  ver = VERSION.full;
}
```

Tuyau de monnaie

Le canal de devise vous permet de travailler avec vos données sous forme de nombres normaux, mais de les afficher avec le format de devise standard (symbole monétaire, décimales, etc.) dans la vue.

```
@Component({
  selector: 'currency-pipe',
  template: `<div>
    <p>A: {{myMoney | currency:'USD':false}}</p>
    <p>B: {{yourMoney | currency:'USD':true:'4.2-2'}}</p>
  </div>`
})
export class CurrencyPipeComponent {
  myMoney: number = 100000.653;
  yourMoney: number = 5.3495;
}
```

Le tuyau prend trois paramètres facultatifs:

- **currencyCode** : Vous permet de spécifier le code de devise ISO 4217.
- **symbolDisplay** : Booléen indiquant s'il faut utiliser le symbole monétaire
- **digitInfo** : vous permet de spécifier comment les décimales doivent être affichées.

Plus de documentation sur le canal des devises:

<https://angular.io/docs/ts/latest/api/common/index/CurrencyPipe-pipe.html>

Lire Directives et services généralement intégrés en ligne:

<https://riptutorial.com/fr/angular2/topic/8252/directives-et-services-generalement-integres>

Chapitre 38: Dropzone dans Angular2

Exemples

Zone de largage

Bibliothèque d'encapsulation angulaire 2 pour Dropzone.

```
npm installer angular2-dropzone-wrapper --save-dev
```

Chargez le module pour votre module d'application

```
import { DropzoneModule } from 'angular2-dropzone-wrapper';
import { DropzoneConfigInterface } from 'angular2-dropzone-wrapper';

const DROPZONE_CONFIG: DropzoneConfigInterface = {
  // Change this to your upload POST address:
  server: 'https://example.com/post',
  maxFileSize: 10,
  acceptedFiles: 'image/*'
};

@NgModule({
  ...
  imports: [
    ...
    DropzoneModule.forRoot(DROPZONE_CONFIG)
  ]
})
```

UTILISATION DES COMPOSANTS

Remplacez simplement l'élément qui serait oralement transmis à Dropzone avec le composant dropzone.

```
<dropzone [config]="config" [message]='Click or drag images here to upload'
(error)="onUploadError($event)" (success)="onUploadSuccess($event)"></dropzone>
```

Créer un composant dropzone

```
import {Component} from '@angular/core';
@Component({
  selector: 'app-new-media',
  templateUrl: './dropzone.component.html',
  styleUrls: ['./dropzone.component.scss']
})
export class DropZoneComponent {

  onUploadError(args: any) {
    console.log('onUploadError:', args);
  }
}
```

```
onUploadSuccess(args: any) {  
    console.log('onUploadSuccess:', args);  
}  
}
```

Lire Dropzone dans Angular2 en ligne: <https://riptutorial.com/fr/angular2/topic/10010/dropzone-dans-angular2>

Chapitre 39: Exemple pour des routes telles que / route / subroute pour les URL statiques

Exemples

Exemple de route de base avec arbre de sous-routes

app.module.ts

```
import {routes} from "./app.routes";

@NgModule({
  declarations: [AppComponent],
  imports: [BrowserModule, mainModule.forRoot(), RouterModule.forRoot(routes)],
  providers: [],
  bootstrap: [AppComponent]
})

export class AppModule { }
```

app.routes.ts

```
import { Routes } from '@angular/router';
import {SubTreeRoutes} from "./subTree/subTreeRoutes.routes";

export const routes: Routes = [
  ...SubTreeRoutes,
  { path: '', redirectTo: 'home', pathMatch: 'full' }
];
```

subTreeRoutes.ts

```
import {Route} from '@angular/router';
import {exampleComponent} from "./example.component";

export const SubTreeRoutes: Route[] = [
  {
    path: 'subTree',
    children: [
      {path: '', component: exampleComponent}
    ]
  }
];
```

Lire Exemple pour des routes telles que / route / subroute pour les URL statiques en ligne:
<https://riptutorial.com/fr/angular2/topic/8910/exemple-pour-des-routes-telles-que---route---subroute-pour-les-url-statiques>

Chapitre 40: Exemples de composants avancés

Remarques

Rappelez-vous que Angular 2 est tout au sujet de la responsabilité singulière. Quelle que soit la taille de votre composant, dédiez une logique distincte à chaque composant. Qu'il s'agisse d'un bouton, d'un lien d'ancrage sophistiqué, d'un en-tête de dialogue ou même d'un sous-élément de sidenav.

Exemples

Sélecteur d'image avec aperçu

Dans cet exemple, nous allons créer un sélecteur d'image qui prévisualise votre image avant de la télécharger. L'aperçu prend également en charge le glisser-déposer des fichiers dans l'entrée. Dans cet exemple, je ne couvrirai que le téléchargement de fichiers uniques, mais vous pourrez bricoler un peu pour que le téléchargement de fichiers multiples fonctionne.

image-preview.html

Ceci est la mise en page HTML de notre aperçu de l'image

```
<!-- Icon as placeholder when no file picked -->
<i class="material-icons">cloud_upload</i>

<!-- file input, accepts images only. Detect when file has been picked/changed with Angular's
native (change) event listener -->
<input type="file" accept="image/*" (change)="updateSource($event)">

<!-- img placeholder when a file has been picked. shows only when 'source' is not empty -->
<img *ngIf="source" [src]="source" src="">
```

image-preview.ts

Ceci est le fichier principal de notre composant `<image-preview>`

```
import {
  Component,
  Output,
  EventEmitter,
} from '@angular/core';

@Component({
  selector: 'image-preview',
  styleUrls: [ './image-preview.css' ],
  templateUrl: './image-preview.html'
})
```

```

export class MtImagePreviewComponent {

  // Emit an event when a file has been picked. Here we return the file itself
  @Output() onChange: EventEmitter<File> = new EventEmitter<File>();

  constructor() {}

  // If the input has changed(file picked) we project the file into the img previewer
  updateSource($event: Event) {
    // We access he file with $event.target['files'][0]
    this.projectImage($event.target['files'][0]);
  }

  // Uses FileReader to read the file from the input
  source:string = '';
  projectImage(file: File) {
    let reader = new FileReader;
    // TODO: Define type of 'e'
    reader.onload = (e: any) => {
      // Simply set e.target.result as our <img> src in the layout
      this.source = e.target.result;
      this.onChange.emit(file);
    };
    // This will process our file and get it's attributes/data
    reader.readAsDataURL(file);
  }
}

```

another.component.html

```

<form (ngSubmit)="submitPhoto()">
  <image-preview (onChange)="getFile($event)"></image-preview>
  <button type="submit">Upload</button>
</form>

```

Et c'est tout. Beaucoup plus facile que c'était dans AngularJS 1.x. J'ai fait ce composant basé sur une ancienne version de AngularJS 1.5.5.

Filterer les valeurs de table par l'entrée

Importer `ReactiveFormsModule` , puis

```

import { Component, OnInit, OnDestroy } from '@angular/core';
import { FormControl } from '@angular/forms';
import { Subscription } from 'rxjs';

@Component({
  selector: 'component',
  template: `
    <input [formControl]="control" />
    <div *ngFor="let item of content">
      {{item.id}} - {{item.name}}
    </div>
  `
})
export class MyComponent implements OnInit, OnDestroy {

```

```
public control = new FormControl('');

public content: { id: number; name: string; }[];

private originalContent = [
  { id: 1, name: 'abc' },
  { id: 2, name: 'abce' },
  { id: 3, name: 'ced' }
];

private subscription: Subscription;

public ngOnInit() {
  this.subscription = this.control.valueChanges.subscribe(value => {
    this.content = this.originalContent.filter(item => item.name.startsWith(value));
  });
}

public ngOnDestroy() {
  this.subscription.unsubscribe();
}
}
```

Lire Exemples de composants avancés en ligne:

<https://riptutorial.com/fr/angular2/topic/5597/exemples-de-composants-avances>

Chapitre 41: Http Interceptor

Remarques

Ce que nous faisons avec la classe `HttpServiceLayer` est d'étendre la classe `Http` à partir de la forme angulaire et d'y ajouter notre propre logique.

Nous injectons ensuite cette classe dans la classe bootstrap de l'application et indiquons angulairement si nous importons la classe `Http`, à l'arrière pour insérer le `HttpServiceLayer`.

Partout dans le code, nous pouvons simplement importer

```
import { Http } from '@angular/http';
```

Mais notre classe sera utilisée pour chaque appel.

Exemples

Classe simple Extension de la classe Http angulaire

```
import { Http, Request, RequestOptionsArgs, Response, RequestOptions, ConnectionBackend, Headers } from '@angular/http';
import { Router } from '@angular/router';
import { Observable } from 'rxjs/Observable';
import 'rxjs/add/observable/empty';
import 'rxjs/add/observable/throw';
import 'rxjs/add/operator/catch';
import { ApplicationConfiguration } from '../application-configuration/application-configuration';

/**
 * This class extends the Http class from angular and adds automatically the server URL(if in
 * development mode) and 2 headers by default:
 * Headers added: 'Content-Type' and 'X-AUTH-TOKEN'.
 * 'Content-Type' can be set in any othe service, and if set, it will NOT be overwritten in
 * this class any more.
 */
export class HttpServiceLayer extends Http {

    constructor(backend: ConnectionBackend, defaultOptions: RequestOptions, private _router: Router, private appConfig: ApplicationConfiguration) {
        super(backend, defaultOptions);
    }

    request(url: string | Request, options?: RequestOptionsArgs): Observable<Response> {
        this.getRequestOptionArgs(options);
        return this.intercept(super.request(this.appConfig.getServerAdress() + url, options));
    }

    /**
     * This method checks if there are any headers added and if not created the headers map and
     * ads 'Content-Type' and 'X-AUTH-TOKEN'
     */
}
```

```

* 'Content-Type' is not overwritten if it is already available in the headers map
*/
getRequestOptionArgs(options?: RequestOptionsArgs): RequestOptionsArgs {
  if (options == null) {
    options = new RequestOptions();
  }
  if (options.headers == null) {
    options.headers = new Headers();
  }

  if (!options.headers.get('Content-Type')) {
    options.headers.append('Content-Type', 'application/json');
  }

  if (this.appConfig.getAuthToken() != null) {
    options.headers.append('X-AUTH-TOKEN', this.appConfig.getAuthToken());
  }

  return options;
}

/**
 * This method as the name suggests intercepts the request and checks if there are any errors.
 * If an error is present it will be checked what error there is and if it is a general one
then it will be handled here, otherwise, will be
 * thrown up in the service layers
 */
intercept(observable: Observable<Response>): Observable<Response> {

  // return observable;
  return observable.catch((err, source) => {
    if (err.status == 401) {
      this._router.navigate(['/login']);
      //return observable;
      return Observable.empty();
    } else {
      //return observable;
      return Observable.throw(err);
    }
  });
}
}

```

Utiliser notre classe à la place de Http d'Angular

Après avoir étendu la classe Http, nous devons indiquer angular pour utiliser cette classe au lieu de la classe Http.

Pour ce faire, dans notre module principal (ou selon les besoins, juste un module particulier), nous devons écrire dans la section fournisseurs:

```

export function httpServiceFactory(xhrBackend: XHRBackend, requestOptions: RequestOptions,
router: Router, appConfig: ApplicationConfiguration) {
  return new HttpServiceLayer(xhrBackend, requestOptions, router, appConfig);
}

import { HttpModule, Http, Request, RequestOptionsArgs, Response, XHRBackend, RequestOptions,
ConnectionBackend, Headers } from '@angular/http';

```



```

import { Router } from '@angular/router';

@NgModule({
  declarations: [ ... ],
  imports: [ ... ],
  exports: [ ... ],
  providers: [
    ApplicationConfiguration,
    {
      provide: Http,
      useFactory: httpServiceFactory,
      deps: [XHRBackend, RequestOptions, Router, ApplicationConfiguration]
    }
  ],
  bootstrap: [AppComponent]
})
export class AppModule { }

```

Remarque: ApplicationConfiguration est juste un service que j'utilise pour conserver certaines valeurs pendant la durée de l'application

Simple HttpClient AuthToken Interceptor (Angular 4.3+)

```

import { Injectable } from '@angular/core';
import { HttpEvent, HttpRequest, HttpInterceptor, HttpHandler } from '@angular/common/http';
import { UserService } from '../services/user.service';
import { Observable } from 'rxjs/Observable';

@Injectable()
export class AuthHeaderInterceptor implements HttpInterceptor {

  constructor(private userService: UserService) {
  }

  intercept(req: HttpRequest<any>, next: HttpHandler): Observable<HttpEvent<any>> {
    if (this.userService.isAuthenticated()) {
      req = req.clone({
        setHeaders: {
          Authorization: `Bearer ${this.userService.token}`
        }
      });
    }
    return next.handle(req);
  }
}

```

Fourniture d'intercepteur (some-module.module.ts)

```
{provide: HTTP_INTERCEPTORS, useClass: AuthHeaderInterceptor, multi: true},
```

Lire Http Interceptor en ligne: <https://riptutorial.com/fr/angular2/topic/1413/http-interceptor>

Chapitre 42: Ignorer la désinfection pour les valeurs de confiance

Paramètres

Params	Détails
sélecteur	nom de tag vous référencez votre composant par dans le html
template (templateUrl)	une chaîne représentant html qui sera insérée partout où se trouve la <code><selector></code> . templateUrl est un chemin vers un fichier html ayant le même comportement
des tuyaux	un tableau de canaux utilisés par ce composant.

Remarques

SUPER IMPORTANT!

DÉSACTIVER LA DÉSINFECTION VOUS PERMET DE RISQUE DE XSS (Cross-Site Scripting) ET D'AUTRES VECTEURS D'ATTAQUE. S'IL VOUS PLAÎT VOUS ASSURER LA CONFIANCE CE QUE VOUS OBTENEZ 100%

L'utilisation de Pipes vous permet de ne modifier que les valeurs d'attributs comme suit:

```
<tag [attribute]="expression or variable reference | pipeName">
```

vous ne pouvez pas utiliser les tuyaux de cette façon:

```
<tag attribute="expression or variable reference | pipeName">
```

ou de cette façon

```
<tag attribute={{expression or variable reference | pipeName}}>
```

Exemples

Bypassing Sanitizing with pipes (pour réutiliser le code)

Le projet suit la structure du guide Angular2 Quickstart [ici](#) .

```
RootOfProject
|
+-- app
|   |-- app.component.ts
|   |-- main.ts
|   |-- pipeUser.component.ts
|   \-- sanitize.pipe.ts
|
|-- index.html
|-- main.html
|-- pipe.html
```

main.ts

```
import { bootstrap } from '@angular/platform-browser-dynamic';
import { AppComponent } from './app.component';

bootstrap(AppComponent);
```

Cela trouve le fichier index.html à la racine du projet et le génère.

app.component.ts

```
import { Component } from '@angular/core';
import { PipeUserComponent } from './pipeUser.component';

@Component({
  selector: 'main-app',
  templateUrl: 'main.html',
  directives: [PipeUserComponent]
})

export class AppComponent { }
```

C'est le composant de niveau supérieur qui regroupe les autres composants utilisés.

pipeUser.component.ts

```
import { Component } from '@angular/core';
import { IgnoreSanitize } from "./sanitize.pipe";

@Component({
  selector: 'pipe-example',
  templateUrl: "pipe.html",
  pipes: [IgnoreSanitize]
})

export class PipeUserComponent{
  constructor () { }
  unsafeValue: string = "unsafe/picUrl?id=";
  docNum: string;

  getUrl(input: string): any {
    if(input !== undefined) {
```

```

        return this.unsafeValue.concat(input);
        // returns : "unsafe/picUrl?id=input"
    } else {
        return "fallback/to/something";
    }
}
}
}

```

Ce composant fournit la vue avec laquelle le Pipe doit fonctionner.

sanitize.pipe.ts

```

import { Pipe, PipeTransform } from '@angular/core';
import { DomSanitizationService } from '@angular/platform-browser';

@Pipe({
  name: 'sanitaryPipe'
})
export class IgnoreSanitize implements PipeTransform {

  constructor(private sanitizer: DomSanitizationService){}

  transform(input: string) : any {
    return this.sanitizer.bypassSecurityTrustUrl(input);
  }
}

```

C'est la logique qui décrit ce que sont les formats de tuyau.

index.html

```

<head>
  Stuff goes here...
</head>
<body>
  <main-app>
    main.html will load inside here.
  </main-app>
</body>

```

main.html

```

<othertags>
</othertags>

<pipe-example>
  pipe.html will load inside here.
</pipe-example>

<moretags>
</moretags>

```

pipe.html

```
<img [src]="getUrl('1234') | sanitaryPipe">
<embed [src]="getUrl() | sanitaryPipe">
```

Si vous deviez inspecter le HTML pendant que l'application est en cours d'exécution, vous verriez qu'il ressemble à ceci:

```
<head>
  Stuff goes here...
</head>

<body>

  <othertags>
  </othertags>

  <img [src]="getUrl('1234') | sanitaryPipe">
  <embed [src]="getUrl() | sanitaryPipe">

  <moretags>
  </moretags>

</body>
```

Lire Ignorer la désinfection pour les valeurs de confiance en ligne:

<https://riptutorial.com/fr/angular2/topic/5942/ignorer-la-desinfection-pour-les-valeurs-de-confiance>

Chapitre 43: Installer des plugins tiers avec angular-cli@1.0.0-beta.10

Remarques

Il est possible d'installer d'autres bibliothèques en suivant cette approche. Cependant, il peut être nécessaire de spécifier le type de module, le fichier principal et l'extension par défaut.

```
'lodash': {  
  format: 'cjs',  
  defaultExtension: 'js',  
  main: 'index.js'  
}
```

```
'moment': {  
  main: 'moment.js'  
}
```

Exemples

Ajout de la bibliothèque jquery dans un projet angular-cli

1. Installez jquery via npm:

```
npm install jquery --save
```

Installez les typages pour la bibliothèque:

Pour ajouter des saisies pour une bibliothèque, procédez comme suit:

```
typings install jquery --global --save
```

2. Ajoutez jquery au fichier angular-cli-build.js au tableau vendorNpmFiles:

Ceci est nécessaire pour que le système de compilation récupère le fichier. Après l'installation, angular-cli-build.js devrait ressembler à ceci:

Parcourez les `node_modules` et recherchez les fichiers et les dossiers que vous souhaitez ajouter au dossier du fournisseur.

```
var Angular2App = require('angular-cli/lib/broccoli/angular2-app');  
  
module.exports = function(defaults) {  
  return new Angular2App(defaults, {
```

```
vendorNpmFiles: [  
  // ...  
  'jquery/dist/*.js'  
  
  ]  
});  
};
```

3. Configurez les mappages SystemJS pour savoir où chercher jquery:

La configuration de SystemJS se trouve dans `system-config.ts` et après la configuration personnalisée, la section associée devrait ressembler à:

```
/** Map relative paths to URLs. */  
const map: any = {  
  'jquery': 'vendor/jquery'  
};  
  
/** User packages configuration. */  
const packages: any = {  
  
  // no need to add anything here for jquery  
  
};
```

4. Dans votre `src / index.html` ajoutez cette ligne

```
<script src="vendor/jquery/dist/jquery.min.js" type="text/javascript"></script>
```

Vos autres options sont les suivantes:

```
<script src="vendor/jquery/dist/jquery.js" type="text/javascript"></script>
```

ou

```
<script src="/vendor/jquery/dist/jquery.slim.js" type="text/javascript"></script>
```

et

```
<script src="/vendor/jquery/dist/jquery.slim.min.js" type="text/javascript"></script>
```

5. Importer et utiliser la bibliothèque jquery dans les fichiers sources de votre projet:

Importez la bibliothèque jquery dans vos fichiers source `.ts` comme ceci:

```
declare var $:any;  
  
@Component({
```

```
)  
export class YourComponent {  
  ngOnInit() {  
    $(".button").click(function(){  
      // now you can DO, what ever you want  
    });  
    console.log();  
  }  
}
```

Si vous avez suivi les étapes correctement, vous devriez maintenant avoir la bibliothèque jquery dans votre projet. Prendre plaisir!

Ajouter une bibliothèque tierce qui n'a pas de typage

Notez que ce n'est que pour les versions angulaires à 1.0.0-beta.10!

Certaines bibliothèques ou plugins peuvent ne pas avoir de typage. Sans ceux-ci, TypeScript ne peut pas les vérifier et provoque donc des erreurs de compilation. Ces bibliothèques peuvent toujours être utilisées mais différemment des modules importés.

1. Inclure une référence de script à la bibliothèque sur votre page (`index.html`)

```
<script src="//cdn.somewhe.re/lib.min.js" type="text/javascript"></script>  
<script src="/local/path/to/lib.min.js" type="text/javascript"></script>
```

- Ces scripts devraient ajouter un global (par exemple, `THREE`, `mapbox`, `$`, etc.) ou être `mapbox global`.

2. Dans le composant qui les requiert, utilisez `declare` pour initialiser une variable correspondant au nom global utilisé par la lib. Cela permet à TypeScript de savoir qu'il a déjà été initialisé. ¹

```
declare var <globalname>: any;
```

Certaines librairies sont attachées à une `window`, qui doit être étendue pour être accessible dans l'application.

```
interface WindowIntercom extends Window { Intercom: any; }  
declare var window: WindowIntercom;
```

3. Utilisez la lib dans vos composants si nécessaire.

```
@Component { ... }  
export class AppComponent implements AfterViewInit {  
  ...  
  ngAfterViewInit() {  
    var geometry = new THREE.BoxGeometry( 1, 1, 1 );  
    window.Intercom('boot', { ... }  
  }  
}
```


- REMARQUE: Certaines bibliothèques peuvent interagir avec le DOM et doivent être utilisées dans la méthode de [cycle de vie des composants](#) appropriée.

Lire [Installer des plugins tiers avec angular-cli@1.0.0-beta.10 en ligne](#):

<https://riptutorial.com/fr/angular2/topic/2328/installer-des-plugins-tiers-avec-angular-cli-1-0-0-beta-10>

Chapitre 44: Interactions entre composants

Syntaxe

- `<element [variableName]="value"></element> //Declaring input to child when using @Input() method.`
- `<element (childOutput)="parentFunction($event)"></element> //Declaring output from child when using @Output() method.`
- `@Output() pageNumberClicked = new EventEmitter(); //Used for sending output data from child component when using @Output() method.`
- `this.pageNumberClicked.emit(pageNum); //Used to trigger data output from child component. when using @Output() method.`
- `@ViewChild(ComponentClass) //Property decorator is required when using ViewChild.`

Paramètres

prénom	Valeur
nombre de pages	Utilisé pour indiquer le nombre de pages à créer pour le composant enfant.
pageNumberClicked	Nom de la variable de sortie dans le composant enfant.
pageChanged	Fonction du composant parent qui écoute les sorties des composants enfants.

Exemples

Parent - Interaction enfant utilisant les propriétés @Input & @Output

Nous avons un `DataListComponent` qui affiche des données que nous tirons d'un service. `DataListComponent` a également un composant `PagerComponent` en tant qu'enfant.

`PagerComponent` crée une liste de numéros de page en fonction du nombre total de pages qu'il reçoit du `DataListComponent`. `PagerComponent` permet également à `DataListComponent` de savoir quand l'utilisateur clique sur un numéro de page via la propriété `Output`.

```
import { Component, NgModule } from '@angular/core';
import { CommonModule } from '@angular/common';
import { DataListService } from '../dataList.service';
import { PagerComponent } from '../pager.component';

@Component({
  selector: 'datalist',
  template: `
    <table>
    <tr *ngFor="let person of personsData">
      <td>{{person.name}}</td>
      <td>{{person.surname}}</td>
    </tr>
  `
})
```

```

        </tr>
    </table>

    <pager [pageCount]="pageCount" (pageNumberClicked)="pageChanged($event)"></pager>
    `
})
export class DataListComponent {
    private personsData = null;
    private pageCount: number;

    constructor(private dataListService: DataListService) {
        var response = this.dataListService.getData(1); //Request first page from the service
        this.personsData = response.persons;
        this.pageCount = response.totalCount / 10; //We will show 10 records per page.
    }

    pageChanged(pageNumber: number){
        var response = this.dataListService.getData(pageNumber); //Request data from the
service with new page number
        this.personsData = response.persons;
    }
}

@NgModule({
    imports: [CommonModule],
    exports: [],
    declarations: [DataListComponent, PagerComponent],
    providers: [DataListService],
})
export class DataListModule { }

```

PagerComponent répertorie tous les numéros de page. Nous définissons l'événement click sur chacun d'eux afin que nous puissions informer le parent du numéro de page cliqué.

```

import { Component, Input, Output, EventEmitter } from '@angular/core';

@Component({
    selector: 'pager',
    template: `
    <div id="pager-wrapper">
        <span *ngFor="#page of pageCount" (click)="pageClicked(page)">{{page}}</span>
    </div>
    `
})
export class PagerComponent {
    @Input() pageCount: number;
    @Output() pageNumberClicked = new EventEmitter();
    constructor() { }

    pageClicked(pageNum) {
        this.pageNumberClicked.emit(pageNum); //Send clicked page number as output
    }
}

```

Parent - Interaction enfant avec ViewChild

ViewChild offre une interaction à sens unique entre parent et enfant. Il n'y a pas de retour ou de sortie de l'enfant lorsque ViewChild est utilisé.

Nous avons un `DataListComponent` qui affiche des informations. `DataListComponent` a `PagerComponent` comme étant son enfant. Lorsque l'utilisateur effectue une recherche sur `DataListComponent`, il obtient des données d'un service et demande à `PagerComponent` d'actualiser la disposition de la pagination en fonction du nouveau nombre de pages.

```
import { Component, NgModule, ViewChild } from '@angular/core';
import { CommonModule } from '@angular/common';
import { DataListService } from '../dataList.service';
import { PagerComponent } from '../pager.component';

@Component({
  selector: 'datalist',
  template: `<input type='text' [(ngModel)]="searchText" />
    <button (click)="getData()">Search</button>
    <table>
    <tr *ngFor="let person of personsData">
      <td>{{person.name}}</td>
      <td>{{person.surname}}</td>
    </tr>
    </table>
    <pager></pager>
  `
})
export class DataListComponent {
  private personsData = null;
  private searchText: string;

  @ViewChild(PagerComponent)
  private pagerComponent: PagerComponent;

  constructor(private dataListService: DataListService) {}

  getData() {
    var response = this.dataListService.getData(this.searchText);
    this.personsData = response.data;
    this.pagerComponent.setPaging(this.personsData / 10); //Show 10 records per page
  }
}

@NgModule({
  imports: [CommonModule],
  exports: [],
  declarations: [DataListComponent, PagerComponent],
  providers: [DataListService],
})
export class DataListModule { }
```

De cette manière, vous pouvez appeler des fonctions définies sur des composants enfants.

Le composant enfant n'est pas disponible tant que le composant parent n'est pas rendu. Tenter d'accéder à l'enfant avant les parents Le `AfterViewInit` vie du crochet `AfterViewInit` provoquera des exceptions.

Interaction bidirectionnelle parent-enfant via un service

Service utilisé pour la communication:

```

import { Injectable } from '@angular/core';
import { Subject } from 'rxjs/Subject';

@Injectable()
export class ComponentCommunicationService {

    private componentChangeSource = new Subject();
    private newDateCreationSource = new Subject<Date>();

    componentChanged$ = this.componentChangeSource.asObservable();
    dateCreated$ = this.newDateCreationSource.asObservable();

    refresh() {
        this.componentChangeSource.next();
    }

    broadcastDate(date: Date) {
        this.newDateCreationSource.next(date);
    }
}

```

Composant parent:

```

import { Component, Inject } from '@angular/core';
import { ComponentCommunicationService } from './component-refresh.service';

@Component({
    selector: 'parent',
    template: `
<button (click)="refreshSubscribed()">Refresh</button>
<h1>Last date from child received: {{lastDate}}</h1>
<child-component></child-component>
`
})
export class ParentComponent implements OnInit {

    lastDate: Date;
    constructor(private communicationService: ComponentCommunicationService) { }

    ngOnInit() {
        this.communicationService.dateCreated$.subscribe(newDate => {
            this.lastDate = newDate;
        });
    }

    refreshSubscribed() {
        this.communicationService.refresh();
    }
}

```

Composant enfant:

```

import { Component, OnInit, Inject } from '@angular/core';
import { ComponentCommunicationService } from './component-refresh.service';

@Component({
    selector: 'child-component',
    template: `
<h1>Last refresh from parent: {{lastRefreshed}}</h1>
`
})

```

```

    <button (click)="sendNewDate()">Send new date</button>
    `
  })
  export class ChildComponent implements OnInit {

    lastRefreshed: Date;
    constructor(private communicationService: ComponentCommunicationService) { }

    ngOnInit() {
      this.communicationService.componentChanged$.subscribe(event => {
        this.onRefresh();
      });
    }

    sendNewDate() {
      this.communicationService.broadcastDate(new Date());
    }

    onRefresh() {
      this.lastRefreshed = new Date();
    }
  }
}

```

AppModule:

```

@NgModule({
  declarations: [
    ParentComponent,
    ChildComponent
  ],
  providers: [ComponentCommunicationService],
  bootstrap: [AppComponent] // not included in the example
})
export class AppModule {}

```

Lire Interactions entre composants en ligne:

<https://riptutorial.com/fr/angular2/topic/7400/interactions-entre-composants>

Chapitre 45: Interactions entre composants

Introduction

Partager des informations entre différentes directives et composants.

Exemples

Passer des données de parent à enfant avec une liaison d'entrée

HeroChildComponent a deux propriétés d'entrée, généralement ornées de décorations @Input.

```
import { Component, Input } from '@angular/core';
import { Hero } from './hero';
@Component({
  selector: 'hero-child',
  template: `
    <h3>{{hero.name}} says:</h3>
    <p>I, {{hero.name}}, am at your service, {{masterName}}.</p>
  `
})
export class HeroChildComponent {
  @Input() hero: Hero;
  @Input('master') masterName: string;
}
```

La propriété d'interception des entrées d'entrée avec un setter

Utilisez un paramètre de propriété en entrée pour intercepter et agir sur une valeur du parent.

Le setter de la propriété input de nom dans l'enfant NameChildComponent ajuste les espaces à partir d'un nom et remplace une valeur vide par du texte par défaut.

```
import { Component, Input } from '@angular/core';
@Component({
  selector: 'name-child',
  template: '<h3>{{name}}</h3>'
})
export class NameChildComponent {
  private _name = '';
  @Input()
  set name(name: string) {
    this._name = (name && name.trim()) || '<no name set>';
  }
  get name(): string { return this._name; }
}
```

Voici le NameParentComponent démontrant des variantes de nom, y compris un nom avec tous les espaces:

```
import { Component } from '@angular/core';
@Component({
  selector: 'name-parent',
  template: `
    <h2>Master controls {{names.length}} names</h2>
    <name-child *ngFor="let name of names" [name]="name"></name-child>
  `
})
export class NameParentComponent {
  // Displays 'Mr. IQ', '<no name set>', 'Bombasto'
  names = ['Mr. IQ', ' ', ' Bombasto '];
}
```

Parent écoute un événement enfant

Le composant enfant expose une propriété `EventEmitter` avec laquelle il émet des événements lorsque quelque chose se produit. Le parent se lie à cette propriété d'événement et réagit à ces événements.

La propriété `EventEmitter` de l'enfant est une propriété de sortie, généralement ornée d'une décoration `@Output`, telle qu'elle apparaît dans ce composant `VoterComponent`:

```
import { Component, EventEmitter, Input, Output } from '@angular/core';
@Component({
  selector: 'my-voter',
  template: `
    <h4>{{name}}</h4>
    <button (click)="vote(true)" [disabled]="voted">Agree</button>
    <button (click)="vote(false)" [disabled]="voted">Disagree</button>
  `
})
export class VoterComponent {
  @Input() name: string;
  @Output() onVoted = new EventEmitter<boolean>();
  voted = false;
  vote(agreed: boolean) {
    this.onVoted.emit(agreed);
    this.voted = true;
  }
}
```

En cliquant sur un bouton, vous déclenchez l'émission d'une valeur vraie ou fausse (la charge utile booléenne).

Le parent `VoteTakerComponent` lie un gestionnaire d'événement (`onVoted`) qui répond à la charge utile de l'événement enfant (`$event`) et met à jour un compteur.

```
import { Component } from '@angular/core';
@Component({
  selector: 'vote-taker',
  template: `
    <h2>Should mankind colonize the Universe?</h2>
    <h3>Agree: {{agreed}}, Disagree: {{disagreed}}</h3>
    <my-voter *ngFor="let voter of voters"
      [name]="voter"
      (onVoted)="onVoted($event)">
  `
})
export class VoteTakerComponent {
  // ...
}
```



```

    </my-voter>
    `
  })
  export class VoteTakerComponent {
    agreed = 0;
    disagreed = 0;
    voters = ['Mr. IQ', 'Ms. Universe', 'Bombasto'];
    onVoted(agreed: boolean) {
      agreed ? this.agreed++ : this.disagreed++;
    }
  }
}

```

Le parent interagit avec l'enfant via une variable locale

Un composant parent ne peut pas utiliser la liaison de données pour lire les propriétés enfant ou appeler des méthodes enfants. Nous pouvons faire les deux en créant une variable de référence de modèle pour l'élément enfant, puis en référençant cette variable dans le modèle parent, comme indiqué dans l'exemple suivant.

Nous avons un enfant `CountdownTimerComponent` qui compte à rebours jusqu'à zéro et lance une fusée. Il a des méthodes de démarrage et d'arrêt qui contrôlent l'horloge et il affiche un message d'état de compte à rebours dans son propre modèle.

```

import { Component, OnDestroy, OnInit } from '@angular/core';
@Component({
  selector: 'countdown-timer',
  template: '<p>{{message}}</p>'
})
export class CountdownTimerComponent implements OnInit, OnDestroy {
  intervalId = 0;
  message = '';
  seconds = 11;
  clearTimer() { clearInterval(this.intervalId); }
  ngOnInit() { this.start(); }
  ngOnDestroy() { this.clearTimer(); }
  start() { this.countDown(); }
  stop() {
    this.clearTimer();
    this.message = `Holding at T-${this.seconds} seconds`;
  }
  private countDown() {
    this.clearTimer();
    this.intervalId = window.setInterval(() => {
      this.seconds -= 1;
      if (this.seconds === 0) {
        this.message = 'Blast off!';
      } else {
        if (this.seconds < 0) { this.seconds = 10; } // reset
        this.message = `T-${this.seconds} seconds and counting`;
      }
    }, 1000);
  }
}

```

Voyons le `CountdownLocalVarParentComponent` qui héberge le composant timer.

```

import { Component }           from '@angular/core';
import { CountdownTimerComponent } from './countdown-timer.component';
@Component({
  selector: 'countdown-parent-lv',
  template: `
    <h3>Countdown to Liftoff (via local variable)</h3>
    <button (click)="timer.start()">Start</button>
    <button (click)="timer.stop()">Stop</button>
    <div class="seconds">{{timer.seconds}}</div>
    <countdown-timer #timer></countdown-timer>
  `,
  styleUrls: ['demo.css']
})
export class CountdownLocalVarParentComponent { }

```

Le composant parent ne peut pas lier de données aux méthodes de démarrage et d'arrêt de l'enfant ni à sa propriété seconds.

Nous pouvons placer une variable locale (#timer) sur le tag () représentant le composant enfant. Cela nous donne une référence au composant enfant lui-même et à la possibilité d'accéder à l'une de ses propriétés ou méthodes depuis le modèle parent.

Dans cet exemple, nous connectons les boutons parents au démarrage et à l'arrêt de l'enfant et utilisons l'interpolation pour afficher la propriété secondes de l'enfant.

Ici, nous voyons le parent et l'enfant travailler ensemble.

Le parent appelle un ViewChild

L'approche des variables locales est simple et facile. Mais il est limité car le câblage parent-enfant doit être effectué entièrement dans le modèle parent. Le composant parent lui-même n'a pas accès à l'enfant.

Nous ne pouvons pas utiliser la technique de la variable locale si une instance de la classe du composant parent doit lire ou écrire des valeurs de composant enfant ou doit appeler des méthodes de composant enfant.

Lorsque la classe du composant parent requiert ce type d'accès, nous injectons le composant enfant dans le parent en tant que ViewChild.

Nous allons illustrer cette technique avec le même exemple de compte à rebours. Nous ne changerons pas d'apparence ou de comportement. L'enfant CountdownTimerComponent est également identique.

Nous passons de la variable locale à la technique ViewChild uniquement à des fins de démonstration. Voici le parent, CountdownViewChildParentComponent:

```

import { AfterViewInit, ViewChild } from '@angular/core';
import { Component }           from '@angular/core';
import { CountdownTimerComponent } from './countdown-timer.component';
@Component({
  selector: 'countdown-parent-vc',
  template: `

```

```

<h3>Countdown to Liftoff (via ViewChild)</h3>
<button (click)="start()">Start</button>
<button (click)="stop()">Stop</button>
<div class="seconds">{{ seconds() }}</div>
<countdown-timer></countdown-timer>
`,
styleUrls: ['demo.css']
})
export class CountdownViewChildParentComponent implements AfterViewInit {
  @ViewChild(CountdownTimerComponent)
  private timerComponent: CountdownTimerComponent;
  seconds() { return 0; }
  ngAfterViewInit() {
    // Redefine `seconds()` to get from the `CountdownTimerComponent.seconds` ...
    // but wait a tick first to avoid one-time devMode
    // unidirectional-data-flow-violation error
    setTimeout(() => this.seconds = () => this.timerComponent.seconds, 0);
  }
  start() { this.timerComponent.start(); }
  stop() { this.timerComponent.stop(); }
}

```

Il faut un peu plus de travail pour placer la vue enfant dans la classe du composant parent.

Nous importons des références au décorateur ViewChild et au hook du cycle de vie AfterViewInit.

Nous injectons l'enfant CountdownTimerComponent dans la propriété private timerComponent via la décoration de la propriété @ViewChild.

La variable locale #timer a disparu des métadonnées du composant. Au lieu de cela, nous liions les boutons aux propres méthodes de démarrage et d'arrêt du composant parent et présentons les secondes de coche dans une interpolation autour de la méthode des secondes du composant parent.

Ces méthodes accèdent directement au composant temporisateur injecté.

Le crochet du cycle de vie ngAfterViewInit est une ride importante. Le composant du minuteur n'est disponible qu'après l'affichage de la vue parent par Angular. Nous affichons donc 0 seconde au départ.

Angular appelle ensuite le hook du cycle de vie ngAfterViewInit. Il est alors trop tard pour mettre à jour l'affichage des secondes du compte à rebours. La règle de flux de données unidirectionnelle d'Angular nous empêche de mettre à jour la vue parente dans le même cycle. Nous devons attendre un tour avant de pouvoir afficher les secondes.

Nous utilisons setTimeout pour attendre une coche, puis révisons la méthode des secondes afin qu'elle prenne les valeurs futures du composant timer.

Parent et enfants communiquent via un service

Un composant parent et ses enfants partagent un service dont l'interface permet une communication bidirectionnelle au sein de la famille.

La portée de l'instance de service est le composant parent et ses enfants. Les composants en

dehors de ce sous-arbre de composant n'ont pas accès au service ou à leurs communications.

Ce `MissionService` connecte `MissionControlComponent` à plusieurs enfants `AstronautComponent`.

```
import { Injectable } from '@angular/core';
import { Subject } from 'rxjs/Subject';
@Injectable()
export class MissionService {
  // Observable string sources
  private missionAnnouncedSource = new Subject<string>();
  private missionConfirmedSource = new Subject<string>();
  // Observable string streams
  missionAnnounced$ = this.missionAnnouncedSource.asObservable();
  missionConfirmed$ = this.missionConfirmedSource.asObservable();
  // Service message commands
  announceMission(mission: string) {
    this.missionAnnouncedSource.next(mission);
  }
  confirmMission(astronaut: string) {
    this.missionConfirmedSource.next(astronaut);
  }
}
```

`MissionControlComponent` fournit à la fois l'instance du service qu'il partage avec ses enfants (via le tableau de métadonnées des fournisseurs) et injecte cette instance dans son constructeur via son constructeur:

```
import { Component } from '@angular/core';
import { MissionService } from './mission.service';
@Component({
  selector: 'mission-control',
  template: `
    <h2>Mission Control</h2>
    <button (click)="announce()">Announce mission</button>
    <my-astronaut *ngFor="let astronaut of astronauts"
      [astronaut]="astronaut">
    </my-astronaut>
    <h3>History</h3>
    <ul>
      <li *ngFor="let event of history">{{event}}</li>
    </ul>
  `,
  providers: [MissionService]
})
export class MissionControlComponent {
  astronauts = ['Lovell', 'Swigert', 'Haise'];
  history: string[] = [];
  missions = ['Fly to the moon!',
    'Fly to mars!',
    'Fly to Vegas!'];
  nextMission = 0;
  constructor(private missionService: MissionService) {
    missionService.missionConfirmed$.subscribe(
      astronaut => {
        this.history.push(`${astronaut} confirmed the mission`);
      }
    );
  }
  announce() {
```

```

    let mission = this.missions[this.nextMission++];
    this.missionService.announceMission(mission);
    this.history.push(`Mission "${mission}" announced`);
    if (this.nextMission >= this.missions.length) { this.nextMission = 0; }
  }
}

```

L'AstronautComponent injecte également le service dans son constructeur. Chaque AstronautComponent est un enfant de MissionControlComponent et reçoit donc l'instance de service de son parent:

```

import { Component, Input, OnDestroy } from '@angular/core';
import { MissionService } from './mission.service';
import { Subscription } from 'rxjs/Subscription';
@Component({
  selector: 'my-astronaut',
  template: `
    <p>
      {{astronaut}}: <strong>{{mission}}</strong>
      <button
        (click)="confirm()"
        [disabled]="!announced || confirmed">
        Confirm
      </button>
    </p>
  `
})
export class AstronautComponent implements OnDestroy {
  @Input() astronaut: string;
  mission = '<no mission announced>';
  confirmed = false;
  announced = false;
  subscription: Subscription;
  constructor(private missionService: MissionService) {
    this.subscription = missionService.missionAnnounced$.subscribe(
      mission => {
        this.mission = mission;
        this.announced = true;
        this.confirmed = false;
      }
    );
  }
  confirm() {
    this.confirmed = true;
    this.missionService.confirmMission(this.astronaut);
  }
  ngOnDestroy() {
    // prevent memory leak when component destroyed
    this.subscription.unsubscribe();
  }
}

```

Notez que nous capturons l'abonnement et que nous nous désabonnons lorsque l'AstronautComponent est détruit. Ceci est une étape de protection contre la fuite de mémoire. Il n'y a pas de risque réel dans cette application car la durée de vie d'un AstronautComponent est la même que celle de l'application elle-même. Cela ne serait pas toujours vrai dans une application plus complexe.

Nous n'ajoutons pas cette protection à MissionControlComponent car, en tant que parent, il contrôle la durée de vie de MissionService. Le journal de l'historique montre que les messages circulent dans les deux sens entre le parent MissionControlComponent et les enfants AstronautComponent, facilité par le service:

Lire [Interactions entre composants en ligne](https://riptutorial.com/fr/angular2/topic/9454/interactions-entre-composants):

<https://riptutorial.com/fr/angular2/topic/9454/interactions-entre-composants>

Chapitre 46: Le routage

Exemples

Routage de base

Le routeur permet la navigation d'une vue à l'autre en fonction des interactions de l'utilisateur avec l'application.

Voici les étapes de la mise en œuvre du routage de base dans Angular 2 -

Précaution élémentaire : Assurez-vous d'avoir l'étiquette

```
<base href='/>
```

comme premier enfant sous votre balise head dans votre fichier index.html. Cette balise indique que votre dossier d'application est la racine de l'application. Angular 2 sait alors organiser vos liens.

La première étape consiste à vérifier si vous pointez sur les dépendances de routage correctes / les plus récentes dans package.json -

```
"dependencies": {  
  .....  
  "@angular/router": "3.0.0-beta.1",  
  .....  
}
```

La deuxième étape consiste à définir l'itinéraire selon sa définition de classe -

```
class Route {  
  path : string  
  pathMatch : 'full'|'prefix'  
  component : Type|string  
  .....  
}
```

Dans un fichier de routes (`route/routes.ts`), importez tous les composants que vous devez configurer pour différents chemins de routage. Chemin vide signifie que la vue est chargée par défaut. ":" dans le chemin indique un paramètre dynamique transmis au composant chargé.

Les routes sont mises à la disposition des applications via l'injection de dépendances. La méthode `ProviderRouter` est appelée avec le paramètre `RouterConfig` pour pouvoir être injectée dans les composants pour appeler des tâches spécifiques au routage.

```
import { provideRouter, RouterConfig } from '@angular/router';  
import { BarDetailComponent } from '../components/bar-detail.component';  
import { DashboardComponent } from '../components/dashboard.component';
```

```
import { LoginComponent } from '../components/login.component';
import { SignupComponent } from '../components/signup.component';

export const appRoutes: RouterConfig = [
  { path: '', pathMatch: 'full', redirectTo: 'login' },
  { path: 'dashboard', component: DashboardComponent },
  { path: 'bars/:id', component: BarDetailComponent },
  { path: 'login', component: LoginComponent },
  { path: 'signup', component: SignupComponent }
];

export const APP_ROUTER_PROVIDER = [provideRouter(appRoutes)];
```

La troisième étape consiste à amorcer le fournisseur de route.

Dans votre `main.ts` (il peut s'agir de n'importe quel nom. En gros, votre fichier principal devrait être défini dans `systemjs.config`)

```
import { bootstrap } from '@angular/platform-browser-dynamic';
import { AppComponent } from './components/app.component';
import { APP_ROUTER_PROVIDER } from './routes/routes';

bootstrap(AppComponent, [ APP_ROUTER_PROVIDER ]).catch(err => console.error(err));
```

La quatrième étape consiste à charger / afficher les composants du routeur en fonction du chemin d'accès accédé. directive est utilisée pour indiquer angulaire où charger le composant. Pour utiliser import le `ROUTER_DIRECTIVES`.

```
import { ROUTER_DIRECTIVES } from '@angular/router';

@Component({
  selector: 'demo-app',
  template: `
    .....
    <div>
      <router-outlet></router-outlet>
    </div>
    .....
  `,
  // Add our router directives we will be using
  directives: [ROUTER_DIRECTIVES]
})
```

La cinquième étape consiste à relier les autres itinéraires. Par défaut, RouterOutlet charge le composant pour lequel le chemin vide est spécifié dans RouterConfig. La directive RouterLink est utilisée avec la balise d'ancrage HTML pour charger les composants associés aux itinéraires. RouterLink génère l'attribut href qui est utilisé pour générer des liens. Pour ex:

```
import { Component } from '@angular/core';
import { ROUTER_DIRECTIVES } from '@angular/router';

@Component({
  selector: 'demo-app',
  template: `
    <a [routerLink]="['/login']">Login</a>
  `
})
```



```

    <a [routerLink]="['/signup']">Signup</a>
    <a [routerLink]="['/dashboard']">Dashboard</a>
  <div>
    <router-outlet></router-outlet>
  </div>
  `,
  // Add our router directives we will be using
  directives: [ROUTER_DIRECTIVES]
})
export class AppComponent { }

```

Maintenant, nous sommes bons avec le routage vers le chemin statique. RouterLink prend également en charge le chemin dynamique en transmettant des paramètres supplémentaires avec le chemin.

```
import {Component} from '@angular/core'; import {ROUTER_DIRECTIVES} from '@angular/router';
```

```

@Component({
  selector: 'demo-app',
  template: `
    <ul>
      <li *ngFor="let bar of bars | async">
        <a [routerLink]="['/bars', bar.id]">
          {{bar.name}}
        </a>
      </li>
    </ul>
  <div>
    <router-outlet></router-outlet>
  </div>
  `,
  // Add our router directives we will be using
  directives: [ROUTER_DIRECTIVES]
})
export class AppComponent { }

```

RouterLink prend un tableau où le premier élément est le chemin de routage et les éléments suivants sont destinés aux paramètres de routage dynamique.

Itinéraires enfant

Parfois, il est logique d'imbriquer les vues ou les itinéraires les uns dans les autres. Par exemple, sur le tableau de bord, vous voulez plusieurs sous-vues, similaires aux onglets mais implémentées via le système de routage, pour afficher les projets, les contacts, les messages des utilisateurs. Pour prendre en charge de tels scénarios, le routeur nous permet de définir des itinéraires enfants.

Nous ajustons d'abord notre `RouterConfig` ci-dessus et ajoutons les routes enfants:

```

import { ProjectsComponent } from '../components/projects.component';
import { MessagesComponent } from '../components/messages.component';

export const appRoutes: RouterConfig = [

```

```

{ path: '', pathMatch: 'full', redirectTo: 'login' },
{ path: 'dashboard', component: DashboardComponent,
  children: [
    { path: '', redirectTo: 'projects', pathMatch: 'full' },
    { path: 'projects', component: 'ProjectsComponent' },
    { path: 'messages', component: 'MessagesComponent' }
  ] },
{ path: 'bars/:id', component: BarDetailComponent },
{ path: 'login', component: LoginComponent },
{ path: 'signup', component: SignupComponent }
];

```

Maintenant que nos routes enfants sont définies, nous devons nous assurer que ces routes enfants peuvent être affichées dans notre `DashboardComponent`, car nous avons ajouté les enfants. Auparavant, nous avons appris que les composants sont affichés dans une `<router-outlet></router-outlet>`. De même, nous déclarons un autre `RouterOutlet` dans `DashboardComponent` :

```

import { Component } from '@angular/core';

@Component({
  selector: 'dashboard',
  template: `
    <a [routerLink]="['projects']">Projects</a>
    <a [routerLink]="['messages']">Messages</a>
    <div>
      <router-outlet></router-outlet>
    </div>
  `
})
export class DashboardComponent { }

```

Comme vous pouvez le voir, nous avons ajouté un autre `RouterOutlet` dans lequel les routes enfants seront affichées. Habituellement, la route avec un chemin vide sera affichée, cependant, nous configurons une redirection vers la route du `projects`, car nous voulons que celle-ci soit affichée immédiatement lorsque la route du `dashboard` est chargée. Cela étant dit, nous avons besoin d'un itinéraire vide, sinon vous obtiendrez une erreur comme celle-ci:

```
Cannot match any routes: 'dashboard'
```

Donc, en ajoutant la route *vide*, c'est-à-dire une route avec un chemin vide, nous avons défini un point d'entrée pour le routeur.

ResolveData

Cet exemple vous montre comment résoudre les données extraites d'un service avant d'afficher la vue de votre application.

Utilise angular / router 3.0.0-beta.2 au moment de l'écriture

users.service.ts

```

...
import { Http, Response } from '@angular/http';
import { Observable } from 'rxjs/Rx';
import { User } from './user.ts';

@Injectable()
export class UsersService {

  constructor(public http:Http) {}

  /**
   * Returns all users
   * @returns {Observable<User[]>}
   */
  index():Observable<User[]> {

    return this.http.get('http://mywebsite.com/api/v1/users')
      .map((res:Response) => res.json());
  }

  /**
   * Returns a user by ID
   * @param id
   * @returns {Observable<User>}
   */
  get(id:number|string):Observable<User> {

    return this.http.get('http://mywebsite.com/api/v1/users/' + id)
      .map((res:Response) => res.json());
  }
}

```

users.resolver.ts

```

...
import { UsersService } from './users.service.ts';
import { Observable } from 'rxjs/Rx';
import {
  Resolve,
  ActivatedRouteSnapshot,
  RouterStateSnapshot
} from "@angular/router";

@Injectable()
export class UsersResolver implements Resolve<User[] | User> {

  // Inject UsersService into the resolver
  constructor(private service:UsersService) {}

  resolve(route:ActivatedRouteSnapshot, state:RouterStateSnapshot):Observable<User[] | User>
  {
    // If userId param exists in current URL, return a single user, else return all users
    // Uses brackets notation to access `id` to suppress editor warning, may use dot
    notation if you create an interface extending ActivatedRoute with an optional id? attribute
    if (route.params['id']) return this.service.get(route.params['id']);
  }
}

```

```

        return this.service.index();
    }
}

```

users.component.ts

Ceci est un composant de page avec une liste de tous les utilisateurs. Il fonctionnera de la même manière pour le composant de la page de détail de l'utilisateur, remplacez `data.users` par `data.user` ou la clé définie dans *app.routes.ts* (voir ci-dessous).

```

...
import { ActivatedRoute } from "@angular/router";

@Component(...)
export class UsersComponent {

    users:User[];

    constructor(route: ActivatedRoute) {
        route.data.subscribe(data => {
            // data['Match key defined in RouterConfig, see below']
            this.users = data.users;
        });
    }

    /**
     * It is not required to unsubscribe from the resolver as Angular's HTTP
     * automatically completes the subscription when data is received from the server
     */
}

```

app.routes.ts

```

...
import { UsersResolver } from '../resolvers/users.resolver';

export const routes:RouterConfig = <RouterConfig>[
    ...
    {
        path: 'user/:id',
        component: UserComponent,
        resolve: {
            // hence data.user in UserComponent
            user: UsersResolver
        }
    },
    {
        path: 'users',
        component: UsersComponent,
        resolve: {
            // hence data.users in UsersComponent, note the pluralisation
            users: UsersResolver
        }
    }
]

```

```
    },  
    ...  
  ]  
  ...
```

app.resolver.ts

Combinez éventuellement plusieurs résolveurs ensemble.

IMPORTANT: Les services utilisés dans le résolveur doivent être importés en premier ou vous obtiendrez un "Aucun fournisseur pour l'erreur ..Resolver". N'oubliez pas que ces services seront disponibles dans le monde entier et que vous n'aurez plus à les déclarer dans les *providers* de composants. Veillez à vous désabonner de tout abonnement pour éviter toute fuite de mémoire

```
...  
import { UsersService } from './users.service';  
import { UsersResolver } from './users.resolver';  
  
export const ROUTE_RESOLVERS = [  
  ...,  
  UsersService,  
  UsersResolver  
]
```

main.browser.ts

Les résolveurs doivent être injectés lors du bootstrap.

```
...  
import {bootstrap} from '@angular/platform-browser-dynamic';  
import { ROUTE_RESOLVERS } from './app.resolver';  
  
bootstrap(<Type>App, [  
  ...  
  ...ROUTE_RESOLVERS  
)  
.catch(err => console.error(err));
```

Routage avec des enfants

Contrairement à la documentation d'origine, j'ai trouvé que c'était le moyen d'imbriquer correctement les routes enfants dans le fichier *app.routing.ts* ou *app.module.ts* (selon vos préférences). Cette approche fonctionne lorsque vous utilisez WebPack ou SystemJS.

L'exemple ci-dessous montre les routes pour les données *home*, *home / counter* et *home / counter / fetch*. Les premier et dernier itinéraires sont des exemples de redirections. Enfin, à la fin de l'exemple, vous pouvez exporter la route à importer dans un fichier distinct. Pour ex. *app.module.ts*

Pour mieux expliquer, Angular exige que vous ayez un itinéraire sans chemin dans le tableau d'enfants qui inclut le composant parent, pour représenter la route parente. C'est un peu déroutant, mais si vous pensez à une URL vide pour une route enfant, elle correspondrait

essentiellement à la même URL que la route parente.

```
import { NgModule } from "@angular/core";
import { RouterModule, Routes } from "@angular/router";

import { HomeComponent } from "../components/home/home.component";
import { FetchDataComponent } from "../components/fetchdata/fetchdata.component";
import { CounterComponent } from "../components/counter/counter.component";

const appRoutes: Routes = [
  {
    path: "",
    redirectTo: "home",
    pathMatch: "full"
  },
  {
    path: "home",
    children: [
      {
        path: "",
        component: HomeComponent
      },
      {
        path: "counter",
        children: [
          {
            path: "",
            component: CounterComponent
          },
          {
            path: "fetch-data",
            component: FetchDataComponent
          }
        ]
      }
    ]
  },
  {
    path: "**",
    redirectTo: "home"
  }
];

@NgModule({
  imports: [
    RouterModule.forRoot(appRoutes)
  ],
  exports: [
    RouterModule
  ]
})
export class AppRoutingModule { }
```

[Excellent exemple et description via Siraj](#)

[Lire Le routage en ligne: https://riptutorial.com/fr/angular2/topic/2334/le-routage](https://riptutorial.com/fr/angular2/topic/2334/le-routage)

Chapitre 47: Mise à jour angulaire de 2 formulaires

Remarques

Angular 2 permet d'accéder à l'instance ngForm en créant une variable de modèle locale. Angular 2 expose des instances de directive telles que ngForm en spécifiant la propriété exportAs de la métadonnée de la directive. Maintenant, l'avantage ici est de ne pas trop coder, vous pouvez accéder à l'instance ngForm et l'utiliser pour accéder aux valeurs soumises ou pour vérifier si tous les champs sont valides en utilisant les propriétés (valide, soumis, valeur, etc.).

```
#f = ngForm (creates local template instance "f")
```

ngForm émet l'événement "ngSubmit" lorsqu'il est soumis (consultez la documentation de @Output pour plus de détails sur l'émetteur d'événement)

```
(ngSubmit)= "login(f.value, f.submitted) "
```

"ngModel" crée un contrôle de formulaire en combinaison avec l'attribut "name".

```
<input type="text" [(ngModel)]="username" placeholder="enter username" required>
```

Lorsque le formulaire est soumis, f.value a l'objet JSON représentant les valeurs soumises.

```
{nom d'utilisateur: 'Sachin', mot de passe: 'Welcome1'}
```

Exemples

Formulaire de modification de mot de passe simple avec validation multi-contrôle

Les exemples ci-dessous utilisent la nouvelle API de formulaire introduite dans RC3.

pw-change.template.html

```
<form class="container" [formGroup]="pwChangeForm">
  <label for="current">Current Password</label>
  <input id="current" formControlName="current" type="password" required><br />

  <label for="newPW">New Password</label>
  <input id="newPW" formControlName="newPW" type="password" required><br />
  <div *ngIf="newPW.touched && newPW.newIsNotOld">
    New password can't be the same as current password.
  </div>
```

```

<label for="confirm">Confirm new password</label>
<input id="confirm" formControlName="confirm" type="password" required><br />
<div *ngIf="confirm.touched && confirm.errors.newMatchesConfirm">
  The confirmation does not match.
</div>

<button type="submit">Submit</button>
</form>

```

pw-change.component.ts

```

import {Component} from '@angular/core'
import {REACTIVE_FORM_DIRECTIVES, FormBuilder, AbstractControl, FormGroup,
  Validators} from '@angular/forms'
import {PWChangeValidators} from './pw-validators'

@Component({
  moduleId: module.id
  selector: 'pw-change-form',
  templateUrl: './pw-change.template.html`,
  directives: [REACTIVE_FORM_DIRECTIVES]
})

export class PWChangeFormComponent {
  pwChangeForm: FormGroup;

  // Properties that store paths to FormControls makes our template less verbose
  current: AbstractControl;
  newPW: AbstractControl;
  confirm: AbstractControl;

  constructor(private fb: FormBuilder) { }
  ngOnInit() {
    this.pwChangeForm = this.fb.group({
      current: ['', Validators.required],
      newPW: ['', Validators.required],
      confirm: ['', Validators.required]
    }, {
      // Here we create validators to be used for the group as a whole
      validator: Validators.compose([
        PWChangeValidators.newIsNotOld,
        PWChangeValidators.newMatchesConfirm
      ])
    });
    this.current = this.pwChangeForm.controls['current'];
    this.newPW = this.pwChangeForm.controls['newPW'];
    this.confirm = this.pwChangeForm.controls['confirm'];
  }
}

```

pw-validators.ts

```

import {FormControl, FormGroup} from '@angular/forms'
export class PWChangeValidators {

  static OldPasswordMustBeCorrect(control: FormControl) {

```



```

    var invalid = false;
    if (control.value != PWChangeValidators.oldPW)
        return { oldPasswordMustBeCorrect: true }
    return null;
}

// Our cross control validators are below
// NOTE: They take in type FormGroup rather than FormControl
static newIsNotOld(group: FormGroup){
    var newPW = group.controls['newPW'];
    if(group.controls['current'].value == newPW.value)
        newPW.setErrors({ newIsNotOld: true });
    return null;
}

static newMatchesConfirm(group: FormGroup){
    var confirm = group.controls['confirm'];
    if(group.controls['newPW'].value != confirm.value)
        confirm.setErrors({ newMatchesConfirm: true });
    return null;
}
}
}

```

Une liste comprenant des classes de bootstrap peut être trouvée [ici](#) .

Angular 2: Formulaires pilotés par un modèle

```

import { Component } from '@angular/core';
import { Router , ROUTER_DIRECTIVES} from '@angular/router';
import { NgForm } from '@angular/forms';

@Component({
    selector: 'login',
    template: `
<h2>Login</h2>
<form #f="ngForm" (ngSubmit)="login(f.value,f.valid)" novalidate>
  <div>
    <label>Username</label>
    <input type="text" [(ngModel)]="username" placeholder="enter username" required>
  </div>
  <div>
    <label>Password</label>
    <input type="password" name="password" [(ngModel)]="password" placeholder="enter password" required>
  </div>
  <input class="btn-primary" type="submit" value="Login">
</form>`
    //For long form we can use **templateUrl** instead of template
})

export class LoginComponent{

    constructor(private router : Router){ }

    login (formValue: any, valid: boolean){
        console.log(formValue);

        if(valid){
            console.log(valid);
        }
    }
}

```

```
    }  
  }  
}
```

Angular 2 Form - Validation personnalisée du courrier électronique / mot de passe

Pour un [clic de démonstration en direct](#) ..

Index des applications ts

```
import {bootstrap} from '@angular/platform-browser-dynamic';  
import {MyForm} from './my-form.component.ts';  
  
bootstrap(MyForm);
```

Valdateur personnalisé

```
import {Control} from '@angular/common';  
  
export class CustomValidators {  
  static emailFormat(control: Control): [[key: string]: boolean] {  
    let pattern:RegExp = /\S+@\S+\.\S+;/;  
    return pattern.test(control.value) ? null : {"emailFormat": true};  
  }  
}
```

Composants du formulaire

```
import {Component} from '@angular/core';  
import {FORM_DIRECTIVES, NgForm, FormBuilder, Control, ControlGroup, Validators} from  
'@angular/common';  
import {CustomValidators} from './custom-validators';  
  
@Component({  
  selector: 'my-form',  
  templateUrl: 'app/my-form.component.html',  
  directives: [FORM_DIRECTIVES],  
  styleUrls: ['styles.css']  
})  
export class MyForm {  
  email: Control;  
  password: Control;  
  group: ControlGroup;  
  
  constructor(builder: FormBuilder) {  
    this.email = new Control('',  
      Validators.compose([Validators.required, CustomValidators.emailFormat])  
    );  
  
    this.password = new Control('',  
      Validators.compose([Validators.required, Validators.minLength(4)])  
    );  
  
    this.group = builder.group({
```

```

        email: this.email,
        password: this.password
    });
}

onSubmit() {
    console.log(this.group.value);
}
}

```

Composants du formulaire HTML

```

<form [ngFormModel]="group" (ngSubmit)="onSubmit()" novalidate>

  <div>
    <label for="email">Email:</label>
    <input type="email" id="email" [ngFormControl]="email">

    <ul *ngIf="email.dirty && !email.valid">
      <li *ngIf="email.hasError('required')">An email is required</li>
    </ul>
  </div>

  <div>
    <label for="password">Password:</label>
    <input type="password" id="password" [ngFormControl]="password">

    <ul *ngIf="password.dirty && !password.valid">
      <li *ngIf="password.hasError('required')">A password is required</li>
      <li *ngIf="password.hasError('minlength')">A password needs to have at least 4
characters</li>
    </ul>
  </div>

  <button type="submit">Register</button>

</form>

```

Angular 2: Reactive Forms (alias les formes réactives)

Cet exemple utilise Angular 2.0.0 Final Release

formulaire d'inscription.component.ts

```

import { FormGroup,
  FormControl,
  FormBuilder,
  Validators } from '@angular/forms';

@Component({
  templateUrl: './registration-form.html'
})
export class ExampleComponent {
  constructor(private _fb: FormBuilder) { }

  exampleForm = this._fb.group({

```

```
name: ['DefaultValue', [<any>Validators.required, <any>Validators.minLength(2)],
email: ['default@defa.ult', [<any>Validators.required, <any>Validators.minLength(2)]]
})
```

formulaire d'inscription.html

```
<form [formGroup]="exampleForm" novalidate (ngSubmit)="submit(exampleForm)">
  <label>Name: </label>
  <input type="text" formControlName="name"/>
  <label>Email: </label>
  <input type="email" formControlName="email"/>
  <button type="submit">Submit</button>
</form>
```

Angular 2 Forms (Reactive Forms) avec formulaire d'inscription et confirmation de la validation du mot de passe

app.module.ts

Ajoutez-les dans votre fichier app.module.ts pour utiliser des formulaires réactifs

```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { FormsModule, ReactiveFormsModule } from '@angular/forms';
import { AppComponent } from './app.component';
@NgModule({
  imports: [
    BrowserModule,
    FormsModule,
    ReactiveFormsModule,
  ],
  declarations: [
    AppComponent
  ]
  providers: [],
  bootstrap: [
    AppComponent
  ]
})
export class AppModule {}
```

app.component.ts

```
import { Component, OnInit } from '@angular/core';
import template from './add.component.html';
import { FormGroup, FormBuilder, Validators } from '@angular/forms';
import { matchingPasswords } from './validators';
@Component({
  selector: 'app',
  template
})
export class AppComponent implements OnInit {
```

```

addForm: FormGroup;
constructor(private FormBuilder: FormBuilder) {
}
ngOnInit() {

this.addForm = this.formBuilder.group({
  username: ['', Validators.required],
  email: ['', Validators.required],
  role: ['', Validators.required],
  password: ['', Validators.required],
  password2: ['', Validators.required] },
  { validator: matchingPasswords('password', 'password2')
  })
});

addUser() {
  if (this.addForm.valid) {
    var adduser = {
      username: this.addForm.controls['username'].value,
      email: this.addForm.controls['email'].value,
      password: this.addForm.controls['password'].value,
      profile: {
        role: this.addForm.controls['role'].value,
        name: this.addForm.controls['username'].value,
        email: this.addForm.controls['email'].value
      }
    };
    console.log(adduser); // adduser var contains all our form values. store it where you
want
    this.addForm.reset(); // this will reset our form values to null
  }
}
}
}

```

app.component.html

```

<div>
  <form [formGroup]="addForm">
    <input type="text" placeholder="Enter username" formControlName="username" />
    <input type="text" placeholder="Enter Email Address" formControlName="email"/>
    <input type="password" placeholder="Enter Password" formControlName="password" />
    <input type="password" placeholder="Confirm Password" name="password2"
formControlName="password2"/>
    <div class='error' *ngIf="addForm.controls.password2.touched">
      <div class="alert-danger errorMessageadduser"
*ngIf="addForm.hasError('mismatchedPasswords') "> Passwords do
not match
      </div>
    </div>
  </div>
  <select name="Role" formControlName="role">
    <option value="admin" >Admin</option>
    <option value="Accounts">Accounts</option>
    <option value="guest">Guest</option>
  </select>
  <br/>
  <br/>
  <button type="submit" (click)="addUser()"><span><i class="fa fa-user-plus" aria-
hidden="true"></i></span> Add User </button>
</form>

```

```
</div>
```

validateurs.ts

```
export function matchingPasswords(passwordKey: string, confirmPasswordKey: string) {
  return (group: ControlGroup): {
    [key: string]: any
  } => {
    let password = group.controls[passwordKey];
    let confirmPassword = group.controls[confirmPasswordKey];

    if (password.value !== confirmPassword.value) {
      return {
        mismatchedPasswords: true
      };
    }
  }
}
```

Angular2 - Créateur de formulaire

FormComponent.ts

```
import {Component} from "@angular/core";
import {FormBuilder} from "@angular/forms";

@Component({
  selector: 'app-form',
  templateUrl: './form.component.html',
  styleUrls: ['./form.component.scss'],
  providers : [FormBuilder]
})

export class FormComponent{
  form : FormGroup;
  emailRegex = /^^\w+([\.-]?\w+)*@\w+([\.-]?\w+)*(\.\w{2,3})+$/;

  constructor(fb: FormBuilder) {

    this.form = fb.group({
      FirstName : new FormControl({value: null}, Validators.compose([Validators.required,
Validators.maxLength(15)])),
      LastName : new FormControl({value: null}, Validators.compose([Validators.required,
Validators.maxLength(15)])),
      Email : new FormControl({value: null}, Validators.compose([
Validators.required,
Validators.maxLength(15),
Validators.pattern(this.emailRegex)]))
    });
  }
}
```

form.component.html

```
<form class="form-details" role="form" [formGroup]="form">
```

```

<div class="row input-label">
  <label class="form-label" for="FirstName">First name</label>
  <input
    [formControl]="form.controls['FirstName']"
    type="text"
    class="form-control"
    id="FirstName"
    name="FirstName">
</div>
<div class="row input-label">
  <label class="form-label" for="LastName">Last name</label>
  <input
    [formControl]="form.controls['LastName']"
    type="text"
    class="form-control"
    id="LastName"
    name="LastName">
</div>
<div class="row">
  <label class="form-label" for="Email">Email</label>
  <input
    [formControl]="form.controls['Email']"
    type="email"
    class="form-control"
    id="Email"
    name="Email">
</div>
<div class="row">
  <button
    (click)="submit()"
    role="button"
    class="btn btn-primary submit-btn"
    type="button"
    [disabled]="!form.valid">Submit</button>
</div>
</div>
</form>

```

Lire Mise à jour angulaire de 2 formulaires en ligne:

<https://riptutorial.com/fr/angular2/topic/4607/mise-a-jour-angulaire-de-2-formulaires>

Chapitre 48: Mise à jour des typings

Exemples

Mettre à jour les typages quand: typings WARN est obsolète

Message d'alerte:

```
typings WARN deprecated 10/25/2016: "registry:dt/jasmine#2.5.0+20161003201800" is deprecated  
(updated, replaced or removed)
```

Mettre à jour la référence avec:

```
npm run typings -- install dt~jasmine --save --global
```

Remplacez [jasmine] pour toute bibliothèque qui lance un avertissement

Lire [Mise à jour des typings en ligne](https://riptutorial.com/fr/angular2/topic/7814/mise-a-jour-des-typings): <https://riptutorial.com/fr/angular2/topic/7814/mise-a-jour-des-typings>

Chapitre 49: Mise à niveau de la force brutale

Introduction

Si vous souhaitez mettre à niveau la version de votre projet Angular CLI, vous risquez de rencontrer des erreurs et des bogues difficiles à résoudre en modifiant simplement le numéro de version de la CLI angulaire dans votre projet. De plus, comme la CLI angulaire cache une grande partie de ce qui se passe dans le processus de construction et de regroupement, vous ne pouvez pas vraiment faire grand-chose lorsque les choses tournent mal.

Parfois, le moyen le plus simple de mettre à jour la version du projet Angular CLI consiste à simplement étoffer un nouveau projet avec la version Angular CLI que vous souhaitez utiliser.

Remarques

Comme Angular 2 est *tellement* modulaire et encapsulé, vous pouvez à peu près simplement copier tous vos composants, services, canaux, directives, puis remplir le NgModule tel qu'il était dans l'ancien projet.

Exemples

Echafaudage d'un nouveau projet CLI angulaire

```
ng new NewProject
```

ou

```
ng init NewProject
```

Lire [Mise à niveau de la force brutale en ligne](https://riptutorial.com/fr/angular2/topic/9152/mise-a-niveau-de-la-force-brutale): <https://riptutorial.com/fr/angular2/topic/9152/mise-a-niveau-de-la-force-brutale>

Chapitre 50: Mocking @ngrx / Store

Introduction

@ngrx / Store est de plus en plus utilisé dans les projets Angular 2. En tant que tel, le magasin doit être injecté dans le constructeur de tout composant ou service qui souhaite l'utiliser. Les tests unitaires Store n'est cependant pas aussi simple que de tester un service simple. Comme pour beaucoup de problèmes, il existe une multitude de façons de mettre en œuvre des solutions. Cependant, la recette de base consiste à écrire une classe fictive pour l'interface Observer et à écrire une classe fictive pour Store. Ensuite, vous pouvez injecter Store en tant que fournisseur dans votre TestBed.

Paramètres

prénom	la description
valeur	valeur suivante à observer
Erreur	la description
se tromper	erreur à jeter
super	la description
action \$	mock Observer qui ne fait rien à moins d'être défini pour le faire dans la classe simulée
actionReducer \$	mock Observer qui ne fait rien à moins d'être défini pour le faire dans la classe simulée
obs \$	faux observable

Remarques

Observer est un générique, mais doit être de type `any` pour éviter la complexité des tests unitaires. La raison de cette complexité est que le constructeur de Store attend des arguments Observer avec différents types génériques. En utilisant `any` évite cette complication.

Il est possible de transmettre des valeurs nulles dans le super constructeur de StoreMock, mais cela limite le nombre d'assertions pouvant être utilisées pour tester la classe plus tard.

Le composant utilisé dans cet exemple est simplement utilisé comme contexte pour savoir comment injecter Store en tant que fourniture dans la configuration de test.

Examples

Observer Mock

```
class ObserverMock implements Observer<any> {
  closed?: boolean = false; // inherited from Observer
  nextVal: any = ''; // variable I made up

  constructor() {}

  next = (value: any): void => { this.nextVal = value; };
  error = (err: any): void => { console.error(err); };
  complete = (): void => { this.closed = true; }
}

let actionReducer$: ObserverMock = new ObserverMock();
let action$: ObserverMock = new ObserverMock();
let obs$: Observable<any> = new Observable<any>();

class StoreMock extends Store<any> {
  constructor() {
    super(action$, actionReducer$, obs$);
  }
}

describe('Component:Typeahead', () => {
  beforeEach(() => {
    TestBed.configureTestingModule({
      imports: [...],
      declarations: [Typeahead],
      providers: [
        {provide: Store, useClass: StoreMock} // NOTICE useClass instead of useValue
      ]
    }).compileComponents();
  });
});
```

Test unitaire pour composant avec Mock Store

Il s'agit d'un test unitaire d'un composant qui contient *Store* en tant que dépendance. Ici, nous créons une nouvelle classe appelée *MockStore* qui est injectée dans notre composant au lieu du Store habituel.

```
import { Injectable } from '@angular/core';
import { TestBed, async } from '@angular/core/testing';
import { AppComponent } from './app.component';
import { DumbComponentComponent } from './dumb-component/dumb-component.component';
import { SmartComponentComponent } from './smart-component/smart-component.component';
import { mainReducer } from './state-management/reducers/main-reducer';
import { StoreModule } from '@ngrx/store';
import { Store } from '@ngrx/store';
import { Observable } from 'rxjs';

class MockStore {
  public dispatch(obj) {
```

```

    console.log('dispatching from the mock store!')
  }

  public select(obj) {
    console.log('selecting from the mock store!');

    return Observable.of({})
  }
}

describe('AppComponent', () => {
  beforeEach(() => {
    TestBed.configureTestingModule({
      declarations: [
        AppComponent,
        SmartComponentComponent,
        DumbComponentComponent,
      ],
      imports: [
        StoreModule.provideStore({mainReducer})
      ],
      providers: [
        {provide: Store, useClass: MockStore}
      ]
    });
  });

  it('should create the app', async(() => {

    let fixture = TestBed.createComponent(AppComponent);
    let app = fixture.debugElement.componentInstance;
    expect(app).toBeTruthy();
  }));
});

```

Test d'unité pour le composant d'espionnage en magasin

Il s'agit d'un test unitaire d'un composant qui contient *Store* en tant que dépendance. Ici, nous pouvons utiliser un magasin avec "l'état initial" par défaut tout en l'empêchant de distribuer des actions lorsque *store.dispatch ()* est appelé.

```

import {TestBed, async} from '@angular/core/testing';
import {AppComponent} from './app.component';
import {DumbComponentComponent} from './dumb-component/dumb-component.component';
import {SmartComponentComponent} from './smart-component/smart-component.component';
import {mainReducer} from './state-management/reducers/main-reducer';
import {StoreModule} from '@ngrx/store';
import {Store} from '@ngrx/store';
import {Observable} from 'rxjs';

describe('AppComponent', () => {
  beforeEach(() => {
    TestBed.configureTestingModule({
      declarations: [
        AppComponent,
        SmartComponentComponent,
        DumbComponentComponent,
      ],

```

```

    imports: [
      StoreModule.provideStore({mainReducer})
    ]
  });

});

it('should create the app', async(() => {
  let fixture = TestBed.createComponent(AppComponent);
  let app = fixture.debugElement.componentInstance;

  var mockStore = fixture.debugElement.injector.get(Store);
  var storeSpy = spyOn(mockStore, 'dispatch').and.callFake(function () {
    console.log('dispatching from the spy!');
  });

}));

});

```

Angular 2 - Mock Observable (composante service +)

un service

- J'ai créé le service post avec la méthode postRequest.

```

import {Injectable} from '@angular/core';
import {Http, Headers, Response} from "@angular/http";
import {PostModel} from "../PostModel";
import 'rxjs/add/operator/map';
import {Observable} from "rxjs";

@Injectable()
export class PostService {

  constructor(private _http: Http) {
  }

  postRequest(postModel: PostModel) : Observable<Response> {
    let headers = new Headers();
    headers.append('Content-Type', 'application/json');
    return this._http.post("/postUrl", postModel, {headers})
      .map(res => res.json());
  }
}

```

Composant

- J'ai créé le composant avec le paramètre result et la fonction postExample qui appellent postService.
- lorsque le paramètre post-requête réussie que résultat doit être "Succès" sinon "Echec"

```

import {Component} from '@angular/core';

```

```

import {PostService} from "../PostService";
import {PostModel} from "../PostModel";

@Component({
  selector: 'app-post',
  templateUrl: './post.component.html',
  styleUrls: ['./post.component.scss'],
  providers : [PostService]
})
export class PostComponent{

  constructor(private _postService : PostService) {

    let postModel = new PostModel();
    result : string = null;
    postExample(){
      this._postService.postRequest(this.postModel)
        .subscribe(
          () => {
            this.result = 'Success';
          },
          err => this.result = 'Fail'
        )
    }
  }
}

```

service de test

- lorsque vous souhaitez tester le service en utilisant http, vous devez utiliser mockBackend. et l'injecter.
- vous devez également injecter postService.

```

describe('Test PostService', () => {
  beforeEach(() => {
    TestBed.configureTestingModule({
      imports: [HttpModule],
      providers: [
        PostService,
        MockBackend,
        BaseRequestOptions,
        {
          provide: Http,
          deps: [MockBackend, BaseRequestOptions],
          useFactory: (backend: XHRBackend, defaultOptions: BaseRequestOptions) => {
            return new Http(backend, defaultOptions);
          }
        }
      ]
    });
  });

  it('sendPostRequest function return Observable', inject([PostService, MockBackend],
    (service: PostService, mockBackend: MockBackend) => {
      let mockPostModel = PostModel();

      mockBackend.connections.subscribe((connection: MockConnection) => {

```

```

    expect(connection.request.method).toEqual(RequestMethod.Post);
    expect(connection.request.url.indexOf('postUrl')).not.toEqual(-1);
    expect(connection.request.headers.get('Content-Type')).toEqual('application/json');
  });

  service
    .postRequest(PostModel)
    .subscribe((response) => {
      expect(response).toBeDefined();
    });
  });
});

```

composant de test

```

describe('testing post component', () => {
  let component: PostComponent;
  let fixture: ComponentFixture<postComponent>;

  let mockRouter = {
    navigate: jasmine.createSpy('navigate')
  };

  beforeEach(async(() => {
    TestBed.configureTestingModule({
      declarations: [PostComponent],
      imports: [RouterTestingModule.withRoutes([], ModalModule.forRoot()) ],
      providers: [PostService, MockBackend, BaseRequestOptions,
        {provide: Http, deps: [MockBackend, BaseRequestOptions],
          useFactory: (backend: XHRBackend, defaultOptions: BaseRequestOptions) => {
            return new Http(backend, defaultOptions);
          }
        },
        {provide: Router, useValue: mockRouter}
      ],
      schemas: [ CUSTOM_ELEMENTS_SCHEMA ]
    }).compileComponents();
  }));

  beforeEach(() => {
    fixture = TestBed.createComponent(PostComponent);
    component = fixture.componentInstance;
    fixture.detectChanges();
  });

  it('test postRequest success', inject([PostService, MockBackend], (service: PostService,
  mockBackend: MockBackend) => {
    fixturePostComponent = TestBed.createComponent(PostComponent);
    componentPostComponent = fixturePostComponent.componentInstance;
    fixturePostComponent.detectChanges();

    component.postExample();
    let postModel = new PostModel();
    let response = {
      'message' : 'message',

```

```

    'ok'      : true
  };
  mockBackend.connections.subscribe((connection: MockConnection) => {
    postComponent.result = 'Success'
    connection.mockRespond(new Response(
      new ResponseOptions({
        body: response
      })
    ))
  });
  service.postRequest(postModel)
    .subscribe((data) => {
      expect(component.result).toBeDefined();
      expect(PostComponent.result).toEqual('Success');
      expect(data).toEqual(response);
    });
  });
});
});

```

Magasin simple

simple.action.ts

```

import { Action } from '@ngrx/store';

export enum simpleActionTpye {
  add = "simpleAction_Add",
  add_Success = "simpleAction_Add_Success"
}

export class simpleAction {
  type: simpleActionTpye
  constructor(public payload: number) { }
}

```

simple.effects.ts

```

import { Effect, Actions } from '@ngrx/effects';
import { Injectable } from '@angular/core';
import { Action } from '@ngrx/store';
import { Observable } from 'rxjs';

import { simpleAction, simpleActionTpye } from './simple.action';

@Injectable()
export class simpleEffects {

  @Effect()
  addAction$: Observable<simpleAction> = this.actions$
    .ofType(simpleActionTpye.add)
    .switchMap((action: simpleAction) => {
      console.log(action);

      return Observable.of({ type: simpleActionTpye.add_Success, payload: action.payload
    })

    // if you have an api use this code
    // return this.http.post(url).catch().map(res=>{ type: simpleAction.add_Success,

```



```

payload:res))
    });
    constructor(private actions$: Actions) { }
}

```

simple.reducer.ts

```

import { Action, ActionReducer } from '@ngrx/store';

import { simpleAction, simpleActionType } from './simple.action';

export const simpleReducer: ActionReducer<number> = (state: number = 0, action: simpleAction):
number => {
  switch (action.type) {
    case simpleActionType.add_Success:
      console.log(action);
      return state + action.payload;
    default:
      return state;
  }
}

```

store / index.ts

```

import { combineReducers, ActionReducer, Action, StoreModule } from '@ngrx/store';
import { EffectsModule } from '@ngrx/effects';
import { ModuleWithProviders } from '@angular/core';
import { compose } from '@ngrx/core';

import { simpleReducer } from "../simple/simple.reducer";
import { simpleEffects } from "../simple/simple.effects";

export interface IAppState {
  sum: number;
}

// all new reducers should be define here
const reducers = {
  sum: simpleReducer
};

export const store: ModuleWithProviders = StoreModule.forRoot(reducers);
export const effects: ModuleWithProviders[] = [
  EffectsModule.forRoot([simpleEffects])
];

```

app.module.ts

```

import { BrowserModule } from '@angular/platform-browser'
import { NgModule } from '@angular/core';

import { effects, store } from "../Store/index";
import { AppComponent } from './app.component';

@NgModule({
  declarations: [

```

```

    AppComponent
  ],
  imports: [
    BrowserModule,
    // store
    store,
    effects
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }

```

app.component.ts

```

import { Component } from '@angular/core';

import { Store } from '@ngrx/store';
import { Observable } from 'rxjs';

import { IAppState } from './Store/index';
import { simpleActionTpye } from './Store/simple/simple.action';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'app';

  constructor(private store: Store<IAppState>) {
    store.select(s => s.sum).subscribe((res) => {
      console.log(res);
    })
    this.store.dispatch({
      type: simpleActionTpye.add,
      payload: 1
    })
    this.store.dispatch({
      type: simpleActionTpye.add,
      payload: 2
    })
    this.store.dispatch({
      type: simpleActionTpye.add,
      payload: 3
    })
  }
}

```

résultat 0 1 3 6

Lire Mocking @ ngrx / Store en ligne: <https://riptutorial.com/fr/angular2/topic/8038/mocking---ngrx--store>

Chapitre 51: Modèles

Introduction

Les modèles sont très similaires aux modèles dans Angular 1, même s'il existe de nombreuses petites modifications syntaxiques qui permettent de mieux comprendre ce qui se passe.

Exemples

Angular 2 Modèles

UN MODÈLE SIMPLE

Commençons par un modèle très simple qui montre notre nom et notre truc préféré:

```
<div>
  Hello my name is {{name}} and I like {{thing}} quite a lot.
</div>
```

{}: RENDU

Pour rendre une valeur, nous pouvons utiliser la syntaxe à double courbe standard:

```
My name is {{name}}
```

Les tubes, précédemment appelés «Filtres», transforment une valeur en une nouvelle valeur, comme la localisation d'une chaîne ou la conversion d'une valeur en virgule flottante en une représentation monétaire:

[: PROPRIÉTÉS DE LIAISON

Pour résoudre et lier une variable à un composant, utilisez la syntaxe []. Si nous avons `this.currentVolume` dans notre composant, nous le ferons passer à notre composant et les valeurs resteront synchronisées:

```
<video-control [volume]="currentVolume"></video-control>
(): HANDLING EVENTS
```

(): GESTION DES ÉVÉNEMENTS Pour écouter un événement sur un composant, nous utilisons la syntaxe ()

```
<my-component (click)="onClick($event)"></my-component>
```

[()]: LIAISON DE DONNÉES À DEUX VOIES

Pour conserver une liaison à jour avec les entrées de l'utilisateur et d'autres événements, utilisez

la syntaxe `[(())]`. Considérez-le comme une combinaison de gestion d'un événement et de liaison d'une propriété:

`<input [(ngModel)] = "myName">` La valeur `this.myName` de votre composant restera synchronisée avec la valeur d'entrée.

*** : L'ASTERISK**

Indique que cette directive traite ce composant comme un modèle et ne le dessine pas tel quel. Par exemple, `ngFor` prend notre et le tamponne pour chaque élément dans les éléments, mais il ne rend jamais notre initiale car il s'agit d'un modèle:

```
<my-component *ngFor="#item of items">  
</my-component>
```

Les autres directives similaires qui fonctionnent sur les modèles plutôt que sur les composants rendus sont `* ngIf` et `* ngSwitch`.

Lire Modèles en ligne: <https://riptutorial.com/fr/angular2/topic/9471/modeles>

Chapitre 52: Modules

Introduction

Les modules angulaires sont des conteneurs pour différentes parties de votre application.

Vous pouvez avoir des modules imbriqués, votre `app.module` est déjà en train d'imbriquer d'autres modules tels que `BrowserModule` et vous pouvez ajouter `RouterModule`, etc.

Exemples

Un module simple

Un module est une classe avec le décorateur `@NgModule`. Pour créer un module, nous ajoutons `@NgModule` passant quelques paramètres:

- `bootstrap` : le composant qui sera la racine de votre application. Cette configuration est uniquement présente sur votre module racine
- `declarations` : Ressources `declarations` le module. Lorsque vous ajoutez un nouveau composant, vous devez mettre à jour les déclarations (`ng generate component` fait automatiquement)
- `exports` : ressources le module exporte pouvant être utilisé dans d'autres modules
- `imports` : ressources utilisées par le module à partir d'autres modules (seules les classes de modules sont acceptées)
- `providers` : ressources pouvant être injectées (di) dans un composant

Un exemple simple:

```
import { AppComponent } from './app.component';
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';

@NgModule({
  bootstrap: [AppComponent]
  declarations: [AppComponent],
  exports: [],
  imports: [BrowserModule],
  providers: [],
})
export class AppModule { }
```

Modules d'imbrication

Les modules peuvent être imbriqués à l'aide du paramètre `imports` de `@NgModule` decorator.

Nous pouvons créer un `core.module` dans notre application qui contiendra des objets génériques, comme un `ReservePipe` (un canal qui inverse une chaîne) et regroupera ceux de ce module:

```
import { CommonModule } from '@angular/common';
import { NgModule } from '@angular/core';
import { ReversePipe } from '../reverse.pipe';

@NgModule({
  imports: [
    CommonModule
  ],
  exports: [ReversePipe], // export things to be imported in another module
  declarations: [ReversePipe],
})
export class CoreModule { }
```

Ensuite, dans le `app.module` :

```
import { CoreModule } from 'app/core/core.module';

@NgModule({
  declarations: [...], // ReversePipe is available without declaring here
                        // because CoreModule exports it
  imports: [
    CoreModule,        // import things from CoreModule
    ...
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Lire Modules en ligne: <https://riptutorial.com/fr/angular2/topic/10840/modules>

Chapitre 53: Modules de fonctionnalités

Exemples

Un module de fonctionnalités

```
// my-feature.module.ts
import { CommonModule } from '@angular/common';
import { NgModule }      from '@angular/core';

import { MyComponent } from './my.component';
import { MyDirective } from './my.directive';
import { MyPipe }       from './my.pipe';
import { MyService }   from './my.service';

@NgModule({
  imports:      [ CommonModule ],
  declarations: [ MyComponent, MyDirective, MyPipe ],
  exports:     [ MyComponent ],
  providers:   [ MyService ]
})
export class MyFeatureModule { }
```

Maintenant, dans votre racine (généralement `app.module.ts`):

```
// app.module.ts
import { NgModule }      from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';

import { AppComponent }  from './app.component';
import { MyFeatureModule } from './my-feature.module';

@NgModule({
  // import MyFeatureModule in root module
  imports:      [ BrowserModule, MyFeatureModule ],
  declarations: [ AppComponent ],
  bootstrap:   [ AppComponent ]
})
export class AppModule { }
```

Lire Modules de fonctionnalités en ligne: <https://riptutorial.com/fr/angular2/topic/6551/modules-de-fonctionnalites>

Chapitre 54: ngrx

Introduction

Ngrx est une bibliothèque puissante que vous pouvez utiliser avec **Angular2**. L'idée est de fusionner deux concepts qui fonctionnent bien ensemble pour avoir une **application réactive** avec un **conteneur d'état** prévisible: - [Redux] [1] - [RxJs] [2] Les principaux avantages: - Partager des données dans votre application entre vos composants va plus facile - Tester la logique de votre application consiste à tester des fonctions pures, sans aucune dépendance sur Angular2 (très simple!) [1]: <http://redux.js.org> [2]: <http://reactivex.io/rxjs>

Exemples

Exemple complet: Connexion / Déconnexion d'un utilisateur

Conditions préalables

Ce sujet **ne** concerne **pas** Redux et / ou Ngrx:

- Vous devez être à l'aise avec Redux
- Au moins comprendre les bases de RxJs et le motif observable

Tout d'abord, définissons un exemple dès le début et jouons avec du code:

En tant que développeur, je veux:

1. Avoir une interface `IUser` qui définit les propriétés d'un `User`
2. Déclarez les actions que nous utiliserons plus tard pour manipuler l' `User` dans le `Store`
3. Définir l'état initial du `UserReducer`
4. Créer le réducteur `UserReducer`
5. Importez notre `UserReducer` dans notre module principal pour créer le `Store`
6. Utilisez les données du `Store` pour afficher des informations à notre vue

Alerte Spoiler : Si vous voulez essayer la démo immédiatement ou lire le code avant même de commencer, voici une vue Plunkr ([vue d' intégration](#) ou [exécution](#)).

1) Définir `IUser` Interface

J'aime partager mes interfaces en deux parties:

- Les propriétés que nous obtiendrons d'un serveur
- Les propriétés que nous définissons uniquement pour l'interface utilisateur (si un bouton tourne par exemple)

Et voici l'interface `IUser` nous utiliserons:

user.interface.ts

```
export interface IUser {
  // from server
  username: string;
  email: string;

  // for UI
  isConnected: boolean;
  isConnecting: boolean;
};
```

2) Déclarez les actions pour manipuler l'`IUser`

Maintenant, nous devons réfléchir au type d'actions que nos *réducteurs* sont censés gérer. Disons ici:

user.actions.ts

```
export const UserActions = {
  // when the user clicks on login button, before we launch the HTTP request
  // this will allow us to disable the login button during the request
  USR_IS_CONNECTING: 'USR_IS_CONNECTING',
  // this allows us to save the username and email of the user
  // we assume those data were fetched in the previous request
  USR_IS_CONNECTED: 'USR_IS_CONNECTED',

  // same pattern for disconnecting the user
  USR_IS_DISCONNECTING: 'USR_IS_DISCONNECTING',
  USR_IS_DISCONNECTED: 'USR_IS_DISCONNECTED'
};
```

Mais avant d'utiliser ces actions, laissez-moi vous expliquer pourquoi nous allons avoir besoin d'un service pour nous envoyer **certaines** de ces actions:

Disons que nous voulons connecter un utilisateur. Nous allons donc cliquer sur un bouton de connexion et voici ce qui va se passer:

- Cliquez sur le bouton
- Le composant intercepte l'événement et appelle `userService.login`
- `userService.login` méthode `userService.login` dispatch un événement pour mettre à jour notre propriété `store: user.isConnected`
- Un appel HTTP est déclenché (nous utiliserons un `setTimeout` dans la démo pour simuler le **comportement asynchrone**)
- Une fois l'appel HTTP terminé, nous enverrons une autre action pour avertir notre magasin qu'un utilisateur est connecté.

user.service.ts

```

@Injectable()
export class UserService {
  constructor(public store$: Store<AppState>) { }

  login(username: string) {
    // first, dispatch an action saying that the user's trying to connect
    // so we can lock the button until the HTTP request finish
    this.store$.dispatch({ type: UserActions.USR_IS_CONNECTING });

    // simulate some delay like we would have with an HTTP request
    // by using a timeout
    setTimeout(() => {
      // some email (or data) that you'd have get as HTTP response
      let email = `${username}@email.com`;

      this.store$.dispatch({ type: UserActions.USR_IS_CONNECTED, payload: { username, email }
    });
  }, 2000);
}

  logout() {
    // first, dispatch an action saying that the user's trying to connect
    // so we can lock the button until the HTTP request finish
    this.store$.dispatch({ type: UserActions.USR_IS_DISCONNECTING });

    // simulate some delay like we would have with an HTTP request
    // by using a timeout
    setTimeout(() => {
      this.store$.dispatch({ type: UserActions.USR_IS_DISCONNECTED });
    }, 2000);
  }
}

```

3) Définir l'état initial du `UserReducer`

user.state.ts

```

export const UserFactory: IUser = () => {
  return {
    // from server
    username: null,
    email: null,

    // for UI
    isConnecting: false,
    isConnected: false,
    isDisconnecting: false
  };
};

```

4) Créez le réducteur `UserReducer`

Un réducteur prend 2 arguments:

- L'état actuel
- Une Action de type `Action<{type: string, payload: any}>`

Rappel: un réducteur doit être initialisé à un moment donné

Comme nous avons défini l'état par défaut de notre réducteur dans la partie 3), nous pourrions l'utiliser comme ça:

user.reducer.ts

```
export const UserReducer: ActionReducer<IUser> = (user: IUser, action: Action) => {
  if (user === null) {
    return userFactory();
  }

  // ...
}
```

Heureusement, il existe un moyen plus simple d'écrire cela en utilisant notre fonction `factory` pour renvoyer un objet et, dans le réducteur, utilisez une [valeur de paramètres par défaut \(ES6\)](#):

```
export const UserReducer: ActionReducer<IUser> = (user: IUser = UserFactory(), action: Action)
=> {
  // ...
}
```

Ensuite, nous devons gérer toutes les actions de notre réducteur: **CONSEIL** : utilisez la fonction [ES6 `Object.assign`](#) pour garder notre état immuable

```
export const UserReducer: ActionReducer<IUser> = (user: IUser = UserFactory(), action: Action)
=> {
  switch (action.type) {
    case UserActions.USR_IS_CONNECTING:
      return Object.assign({}, user, { isConnecting: true });

    case UserActions.USR_IS_CONNECTED:
      return Object.assign({}, user, { isConnecting: false, isConnected: true, username:
action.payload.username });

    case UserActions.USR_IS_DISCONNECTING:
      return Object.assign({}, user, { isDisconnecting: true });

    case UserActions.USR_IS_DISCONNECTED:
      return Object.assign({}, user, { isDisconnecting: false, isConnected: false });

    default:
      return user;
  }
};
```

5) Importer notre `UserReducer` dans notre module

principal pour construire le `Store`

app.module.ts

```
@NgModule({
  declarations: [
    AppComponent
  ],
  imports: [
    // angular modules
    // ...

    // declare your store by providing your reducers
    // (every reducer should return a default state)
    StoreModule.provideStore({
      user: UserReducer,
      // of course, you can put as many reducers here as you want
      // ...
    }),

    // other modules to import
    // ...
  ]
});
```

6) Utiliser les données de la `Store` pour afficher des informations à notre avis

Tout est maintenant prêt sur le côté logique et nous devons simplement afficher ce que nous voulons en deux composants:

- `UserComponent` : **[Composant Dumb]** Nous allons simplement passer l'objet utilisateur du magasin en utilisant la propriété `@Input` et le tube `async`. De cette façon, le composant ne recevra l'utilisateur qu'une fois qu'il sera disponible (et l'`user` sera de type `IUser` et non de type `Observable<IUser>` !)
- `LoginComponent` **[Composant intelligent]** Nous injecterons directement le `Store` dans ce composant et travaillerons uniquement sur l'`user` en tant `Observable`.

user.component.ts

```
@Component({
  selector: 'user',
  styles: [
    '.table { max-width: 250px; }',
    '.truthy { color: green; font-weight: bold; }',
    '.falsy { color: red; }'
  ],
  template: `
    <h2>User information :</h2>

    <table class="table">
```

```

<tr>
  <th>Property</th>
  <th>Value</th>
</tr>

<tr>
  <td>username</td>
  <td [class.truthy]="user.username" [class.falsy]="!user.username">
    {{ user.username ? user.username : 'null' }}
  </td>
</tr>

<tr>
  <td>email</td>
  <td [class.truthy]="user.email" [class.falsy]="!user.email">
    {{ user.email ? user.email : 'null' }}
  </td>
</tr>

<tr>
  <td>isConnecting</td>
  <td [class.truthy]="user.isConnecting" [class.falsy]="!user.isConnecting">
    {{ user.isConnecting }}
  </td>
</tr>

<tr>
  <td>isConnected</td>
  <td [class.truthy]="user.isConnected" [class.falsy]="!user.isConnected">
    {{ user.isConnected }}
  </td>
</tr>

<tr>
  <td>isDisconnecting</td>
  <td [class.truthy]="user.isDisconnecting" [class.falsy]="!user.isDisconnecting">
    {{ user.isDisconnecting }}
  </td>
</tr>
</table>
`
,
})
export class UserComponent {
  @Input() user;

  constructor() { }
}

```

login.component.ts

```

@Component({
  selector: 'login',
  template: `
    <form
      *ngIf="!(user | async).isConnected"
      #loginForm="ngForm"
      (ngSubmit)="login(loginForm.value.username)"
    >
      <input
        type="text"

```

```

        name="username"
        placeholder="Username"
        [disabled]="(user | async).isConnecting"
        ngModel
    >

    <button
        type="submit"
        [disabled]="(user | async).isConnecting || (user | async).isConnected"
    >Log me in</button>
</form>

<button
    *ngIf="(user | async).isConnected"
    (click)="logout()"
    [disabled]="(user | async).isDisconnecting"
    >Log me out</button>
,
})
export class LoginComponent {
    public user: Observable<IUser>;

    constructor(public store$: Store<AppState>, private userService: UserService) {
        this.user = store$.select('user');
    }

    login(username: string) {
        this.userService.login(username);
    }

    logout() {
        this.userService.logout();
    }
}

```

Comme `Ngrx` est une fusion des concepts de `Redux` et de `RxJs`, il peut être assez difficile de comprendre les détails au début. Mais il s'agit d'un modèle puissant qui vous permet, comme nous l'avons vu dans cet exemple, de disposer d'une *application réactive* et de partager facilement vos données. N'oubliez pas qu'il y a un [Plunkr](#) disponible et que vous pouvez le préparer pour faire vos propres tests!

J'espère que c'était utile même si le sujet est assez long, bravo!

Lire `ngrx` en ligne: <https://riptutorial.com/fr/angular2/topic/8086/ngrx>

Chapitre 55: npx-bootstrap personnalisé datepicker + entrée

Exemples

npx-bootstrap datepicker personnalisé

datepicker.component.html

```
<div (clickOutside)="onClickedOutside($event)" (blur)="onClickedOutside($event)">
  <div class="input-group date" [ngClass]="{'disabled-icon': disabledDatePicker == false}">
    <input (change)="changedDate()" type="text" [ngModel]="value" class="form-control"
    id="{{id}}" (focus)="openCloseDatepicker()" disabled="{{disabledInput}}" />
    <span id="openCloseDatepicker" class="input-group-addon"
    (click)="openCloseDatepicker()">
      <span class="glyphicon-calendar glyphicon"></span>
    </span>
  </div>

  <div class="dp-popup" *ngIf="showDatePicker">
    <datepicker [startingDay]="1" [startingDay]="dt" [minDate]="min" [(ngModel)]="dt"
    (selectionDone)="onSelectionDone($event)"></datepicker>
  </div>
</div>
```

datepicker.component.ts

```
import {Component, Input, EventEmitter, Output, OnChanges, SimpleChanges, ElementRef, OnInit}
from "@angular/core";
import {DatePipe} from "@angular/common";
import {NgModel} from "@angular/forms";
import * as moment from 'moment';

@Component({
  selector: 'custom-datepicker',
  templateUrl: 'datepicker.component.html',
  providers: [DatePipe, NgModel],
  host: {
    '(document:mousedown)': 'onClick($event)',
  }
})

export class DatepickerComponent implements OnChanges , OnInit{
  ngOnInit(): void {
    this.dt = null;
  }

  inputElement : ElementRef;
  dt: Date = null;
  showDatePicker: boolean = false;

  @Input() disabledInput : boolean = false;
```

```

@Input() disabledDatePicker: boolean = false;
@Input() value: string = null;
@Input() id: string;
@Input() min: Date = null;
@Input() max: Date = null;

@Output() dateModelChange = new EventEmitter();
constructor(el: ElementRef) {
  this.inputElement = el;
}

changedDate(){
  if(this.value === ''){
    this.dateModelChange.emit(null);
  }else if(this.value.split('/').length === 3){
    this.dateModelChange.emit(DatepickerComponent.convertToDate(this.value));
  }
}

clickOutside(event : Event){
  if(this.inputElement.nativeElement !== event.target) {
    console.log('click outside', event);
  }
}

onClick(event) {
  if (!this.inputElement.nativeElement.contains(event.target)) {
    this.close();
  }
}

ngOnChanges(changes: SimpleChanges): void {
  if (this.value !== null && this.value !== undefined && this.value.length > 0) {
    this.value = null;
    this.dt = null;
  }else {
    if(this.value !== null){
      this.dt = new Date(this.value);
      this.value = moment(this.value).format('MM/DD/YYYY');
    }
  }
}

private static transformDate(date: Date): string {
  return new DatePipe('pt-PT').transform(date, 'MM/dd/yyyy');
}

openCloseDatepicker(): void {
  if (!this.disabledDatePicker) {
    this.showDatepicker = !this.showDatepicker;
  }
}

open(): void {
  this.showDatepicker = true;
}

close(): void {
  this.showDatepicker = false;
}

private apply(): void {

```



```
    this.value = DatepickerComponent.transformDate(this.dt);
    this.dateModelChange.emit(this.dt);
  }

  onSelectionDone(event: Date): void {
    this.dt = event;
    this.apply();
    this.close();
  }

  onClickedOutside(event: Date): void {
    if (this.showDatepicker) {
      this.close();
    }
  }

  static convertToDate(val : string): Date {
    return new Date(val.replace('/', '-'));
  }
}
```

Lire [npx-bootstrap personnalisé datepicker + entrée en ligne](https://riptutorial.com/fr/angular2/topic/10549/npx-bootstrap-personnalise-datepicker-plus-entree):

<https://riptutorial.com/fr/angular2/topic/10549/npx-bootstrap-personnalise-datepicker-plus-entree>

Chapitre 56: Optimisation du rendu à l'aide de ChangeDetectionStrategy

Exemples

Défaut vs OnPush

Considérez le composant suivant avec une entrée `myInput` et une valeur interne appelée `someInternalValue`. Les deux sont utilisés dans le modèle d'un composant.

```
import {Component, Input} from '@angular/core';

@Component ({
  template:`
  <div>
    <p>{{myInput}}</p>
    <p>{{someInternalValue}}</p>
  </div>
  `
})
class MyComponent {
  @Input () myInput: any;

  someInternalValue: any;

  // ...
}
```

Par défaut, la propriété `changeDetection:` du composant decorator sera définie sur `ChangeDetectionStrategy.Default`; implicite dans l'exemple. Dans cette situation, toute modification apportée à l'une des valeurs du modèle déclenchera un nouveau rendu de `MyComponent`. En d'autres termes, si je change `myInput` ou `someInternalValue` angular 2 exercera une énergie et restituera le composant.

Supposons, cependant, que nous ne souhaitons que rendre à nouveau lorsque les entrées changent. Considérez le composant suivant avec `changeDetection:` défini sur `ChangeDetectionStrategy.OnPush`

```
import {Component, ChangeDetectionStrategy, Input} from '@angular/core';

@Component ({
  changeDetection: ChangeDetectionStrategy.OnPush
  template:`
  <div>
    <p>{{myInput}}</p>
    <p>{{someInternalValue}}</p>
  </div>
  `
})
class MyComponent {
  @Input () myInput: any;
```

```
someInternalValue: any;  
  
// ...  
}
```

En définissant le paramètre `changeDetection: sur ChangeDetectionStrategy.OnPush`, `MyComponent` ne sera rendu que lorsque ses entrées seront modifiées. Dans ce cas, `myInput` devra recevoir une nouvelle valeur de son parent pour déclencher un re-render.

Lire [Optimisation du rendu à l'aide de ChangeDetectionStrategy en ligne](https://riptutorial.com/fr/angular2/topic/2644/optimisation-du-rendu-a-l-aide-de-changedetectionstrategy):
<https://riptutorial.com/fr/angular2/topic/2644/optimisation-du-rendu-a-l-aide-de-changedetectionstrategy>

Chapitre 57: Ouvrier

Introduction

Nous verrons comment mettre en place un service de traitement angulaire, afin de permettre à notre application Web d'avoir des fonctionnalités hors ligne.

Un agent de service est un script spécial qui s'exécute en arrière-plan dans le navigateur et gère les demandes réseau à une origine donnée. Il est à l'origine installé par une application et reste résident sur la machine / le périphérique de l'utilisateur. Il est activé par le navigateur lorsqu'une page de son origine est chargée et a la possibilité de répondre aux requêtes HTTP lors du chargement de la page.

Exemples

Ajouter Service Worker à notre application

D'abord, si vous consultez mobile.angular.io le drapeau --mobile ne fonctionne plus.

Donc, pour commencer, nous pouvons créer un projet normal avec des angles angulaires.

```
ng new serviceWorking-example
cd serviceWorking-example
```

Maintenant, l'important, c'est de dire aux clients angulaires que nous voulons utiliser un technicien, nous devons faire:

```
ng set apps.0.serviceWorker = true
```

Si pour une raison quelconque vous n'avez pas installé @ angular / service-worker, vous verrez un message:

Votre projet est configuré avec serviceWorker = true, mais @ angular / service-worker n'est pas installé. Exécutez `npm install --save-dev @angular/service-worker` et réessayez, ou lancez `ng set apps.0.serviceWorker=false` dans votre fichier `.angular-cli.json`.

Vérifiez le fichier `.angular-cli.json` et vous devriez maintenant voir ceci: "serviceWorker": true

Lorsque cet indicateur est true, les versions de production seront configurées avec un agent de maintenance.

Un fichier `ngsw-manifest.json` sera généré (ou augmenté au cas où nous aurions créé un fichier `ngsw-manifest.json` à la racine du projet, ceci est généralement fait pour spécifier le routage, dans le futur cela sera probablement fait automatiquement) dans le `dist / root`, et le script du travailleur du service y sera copié. Un script court sera ajouté à `index.html` pour enregistrer le service worker.

Maintenant, si nous construisons l'application en mode de production `ng build --prod`

Et vérifiez `dist / folder`.

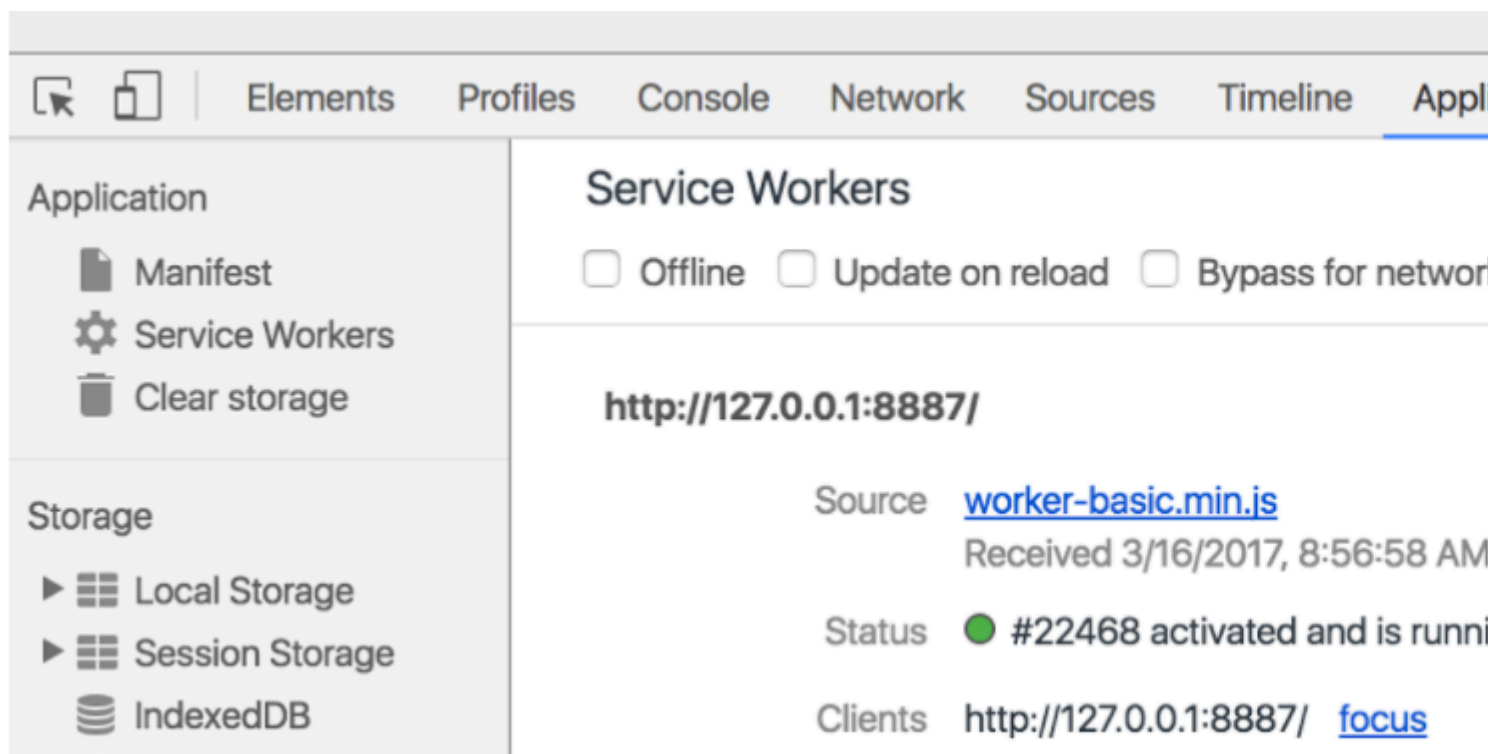
Vous verrez trois nouveaux fichiers là-bas:

- `worker-basic.min.js`
- `sw-register.HASH.bundle.js`
- `ngsw-manifest.json`

De plus, `index.html` inclut maintenant ce script `sw-register`, qui enregistre un agent de service angulaire (ASW) pour nous.

Actualisez la page dans votre navigateur (servi par le serveur Web pour Chrome)

Ouvrez les outils de développement. Allez à l'application -> Service Workers



Bon maintenant le Service Worker est opérationnel!

Maintenant, notre application devrait charger plus rapidement et nous devrions être en mesure d'utiliser l'application en mode hors connexion.

Maintenant, si vous activez le mode hors connexion dans la console chrome, vous devriez voir que notre application dans <http://localhost:4200/index.html> fonctionne sans connexion à Internet.

Mais dans <http://localhost:4200/> nous avons un problème et il ne se charge pas, ceci est dû au cache de contenu statique qui ne sert que les fichiers listés dans le manifeste.

Par exemple, si le manifeste déclare une URL de `/index.html`, le cache répondra aux demandes

adressées à /index.html, mais une requête à / ou / some / route sera envoyée au réseau.

C'est là que le plug-in de redirection de route entre en jeu. Il lit une configuration de routage à partir du manifeste et redirige les itinéraires configurés vers une route d'indexation spécifiée.

Actuellement, cette section de configuration doit être écrite à la main (19-7-2017). À terme, il sera généré à partir de la configuration de la route présente dans la source de l'application.

Donc, si maintenant nous créons ou ngsw-manifest.json à la racine du projet

```
{
  "routing": {
    "routes": {
      "/": {
        "prefix": false
      }
    },
    "index": "/index.html"
  }
}
```

Et nous construisons à nouveau notre application, maintenant que nous allons à <http://localhost:4200/>, nous devrions être redirigés vers <http://localhost:4200/index.html>.

Pour plus d'informations sur le routage, lisez la [documentation officielle ici](#)

Vous trouverez ici plus de documentation sur les travailleurs du service:

<https://developers.google.com/web/fundamentals/getting-started/primers/service-workers>

https://docs.google.com/document/d/19S5ozevWighny788nI99worpcIMDnwWVmaJDGf_RoDY/edit#

Et ici, vous pouvez voir une autre manière d'implémenter le service en utilisant la bibliothèque SW de précache:

<https://coryryan.com/blog/fast-offline-angular-apps-with-service-workers>

Lire Ouvrier en ligne: <https://riptutorial.com/fr/angular2/topic/10809/ouvrier>

Chapitre 58: Pipes

Introduction

Le tuyau `|` character est utilisé pour appliquer des canaux dans Angular 2. Les tubes sont très similaires aux filtres d'AngularJS car ils permettent de transformer les données dans un format spécifié.

Paramètres

Fonction / Paramètre	Explication
@Pipe ({name, pure})	les métadonnées pour la conduite, doivent immédiatement précéder la classe de conduite
nom: <i>chaîne</i>	ce que vous utiliserez à l'intérieur du modèle
pur: <i>booléen</i>	la valeur par défaut est true, cochez cette case pour que votre tube soit réévalué plus souvent
transformer (valeur, args []?)	la fonction appelée pour transformer les valeurs du modèle
valeur: <i>tout</i>	la valeur que vous souhaitez transformer
args: <i>any []</i>	les arguments dont vous avez besoin inclus dans votre transformation. Marquer des arguments facultatifs avec le? opérateur comme si transform (value, arg1, arg2?)

Remarques

Cette rubrique couvre [Angular2 Pipes](#) , un mécanisme de transformation et de mise en forme des données dans les modèles HTML dans une application Angular2.

Exemples

Chaining Pipes

Les tuyaux peuvent être enchaînés.

```
<p>Today is {{ today | date:'fullDate' | uppercase}}.</p>
```

Tuyaux personnalisés

my.pipe.ts

```
import { Pipe, PipeTransform } from '@angular/core';

@Pipe({name: 'myPipe'})
export class MyPipe implements PipeTransform {

  transform(value:any, args?: any):string {
    let transformedValue = value; // implement your transformation logic here
    return transformedValue;
  }

}
```

my.component.ts

```
import { Component } from '@angular/core';

@Component({
  selector: 'my-component',
  template: `{{ value | myPipe }}`
})
export class MyComponent {

  public value:any;

}
```

my.module.ts

```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';

import { MyComponent } from './my.component';
import { MyPipe } from './my.pipe';

@NgModule({
  imports: [
    BrowserModule,
  ],
  declarations: [
    MyComponent,
    MyPipe
  ],
})
export class MyModule { }
```

Tuyaux Intégrés

Angular2 est livré avec quelques tuyaux intégrés:

Tuyau	Usage	Exemple
<code>DatePipe</code>	<code>date</code>	<code>{{ dateObj date }} // output is 'Jun 15, 2015'</code>
<code>UpperCasePipe</code>	<code>uppercase</code>	<code>{{ value uppercase }} // output is 'SOMETEXT'</code>
<code>LowerCasePipe</code>	<code>lowercase</code>	<code>{{ value lowercase }} // output is 'sometext'</code>
<code>CurrencyPipe</code>	<code>currency</code>	<code>{{ 31.00 currency:'USD':true }} // output is '\$31'</code>
<code>PercentPipe</code>	<code>percent</code>	<code>{{ 0.03 percent }} //output is %3</code>

Il y en a d'autres Regardez [ici](#) pour leur documentation.

Exemple

hotel-reservation.component.ts

```
import { Component } from '@angular/core';

@Component({
  moduleId: module.id,
  selector: 'hotel-reservation',
  templateUrl: './hotel-reservation.template.html'
})
export class HotelReservationComponent {
  public fName: string = 'Joe';
  public lName: string = 'SCHMO';
  public reservationMade: string = '2016-06-22T07:18-08:00'
  public reservationFor: string = '2025-11-14';
  public cost: number = 99.99;
}
```

hotel-reservation.template.html

```
<div>
  <h1>Welcome back {{fName | uppercase}} {{lName | lowercase}}</h1>
  <p>
    On {reservationMade | date} at {reservationMade | date:'shortTime'} you
    reserved room 205 for {reservationDate | date} for a total cost of
    {cost | currency}.
  </p>
</div>
```

Sortie

```
Welcome back JOE schmo
On Jun 26, 2016 at 7:18 you reserved room 205 for Nov 14, 2025 for a total cost of
$99.99.
```

Déboguer avec JsonPipe

JsonPipe peut être utilisé pour déboguer l'état de tout interne donné.

Code

```
@Component({
  selector: 'json-example',
  template: `<div>
    <p>Without JSON pipe:</p>
    <pre>{{object}}</pre>
    <p>With JSON pipe:</p>
    <pre>{{object | json}}</pre>
  </div>`
})
export class JsonPipeExample {
  object: Object = {foo: 'bar', baz: 'qux', nested: {xyz: 3, numbers: [1, 2, 3, 4, 5]}};
}
```

Sortie

```
Without JSON Pipe:
object
With JSON pipe:
{object:{foo: 'bar', baz: 'qux', nested: {xyz: 3, numbers: [1, 2, 3, 4, 5]}}
```

Pipe personnalisée disponible à l'échelle mondiale

Pour rendre une application personnalisée disponible au niveau de l'application, lors du démarrage de l'application, étendez PLATFORM_PIPES.

```
import { bootstrap } from '@angular/platform-browser-dynamic';
import { provide, PLATFORM_PIPES } from '@angular/core';

import { AppComponent } from './app.component';
import { MyPipe } from './my.pipe'; // your custom pipe

bootstrap(AppComponent, [
  provide(PLATFORM_PIPES, {
    useValue: [
      MyPipe
    ],
    multi: true
  })
]);
```

Tutoriel ici: <https://scotch.io/tutorials/create-a-globally-available-custom-pipe-in-angular-2>

Créer un tuyau personnalisé

app / pipes.pipe.ts

```
import { Pipe, PipeTransform } from '@angular/core';
```

```

@Pipe({name: 'truthy'})
export class Truthy implements PipeTransform {
  transform(value: any, truthy: string, falsey: string): any {
    if (typeof value === 'boolean'){return value ? truthy : falsey;}
    else return value
  }
}

```

app / my-component.component.ts

```

import { Truthy } from './pipes.pipe';

@Component({
  selector: 'my-component',
  template: `
    <p>{{value | truthy:'enabled':'disabled' }}</p>
  `,
  pipes: [Truthy]
})
export class MyComponent{ }

```

Déballez les valeurs asynchrones avec un tuyau asynchrone

```

import { Component } from '@angular/core';
import { Observable } from 'rxjs/Observable';
import 'rxjs/add/observable/of';

@Component({
  selector: 'async-stuff',
  template: `
    <h1>Hello, {{ name | async }}</h1>
    Your Friends are:
    <ul>
      <li *ngFor="let friend of friends | async">
        {{friend}}
      </li>
    </ul>
  `,
})
class AsyncStuffComponent {
  name = Promise.resolve('Misko');
  friends = Observable.of(['Igor']);
}

```

Devient:

```

<h1>Hello, Misko</h1>
Your Friends are:
<ul>
  <li>
    Igor
  </li>
</ul>

```

Extension d'un tuyau existant

```

import { Pipe, PipeTransform } from '@angular/core';
import { DatePipe } from '@angular/common'

@Pipe({name: 'ifDate'})
export class IfDate implements PipeTransform {
  private datePipe: DatePipe = new DatePipe();

  transform(value: any, pattern?:string) : any {
    if (typeof value === 'number') {return value}
    try {
      return this.datePipe.transform(value, pattern)
    } catch(err) {
      return value
    }
  }
}

```

Pipes Stateful

Angular 2 propose deux types de tuyaux différents - sans état et avec état. Les tuyaux sont sans état par défaut. Cependant, nous pouvons implémenter des canaux avec état en définissant la propriété `pure` sur `false`. Comme vous pouvez le voir dans la section des paramètres, vous pouvez spécifier un `name` et déclarer si le tube doit être pur ou non, c'est-à-dire avec état ou sans état. Alors que les données transitent par un canal sans état (qui est une fonction pure) qui **ne se** souvient de rien, les données peuvent être gérées et mémorisées par des canaux avec état.

`AsyncPipe` fourni par Angular 2 est un bon exemple de pipe avec état.

Important

Notez que la plupart des tuyaux doivent tomber dans la catégorie des tuyaux sans état. C'est important pour des raisons de performance, car Angular peut optimiser les tuyaux sans état pour le détecteur de changement. Donc, utilisez les tuyaux avec état avec précaution. En général, l'optimisation des tubes dans Angular 2 améliore considérablement les performances par rapport aux filtres dans Angular 1.x. Dans Angular 1, le cycle de digestion devait toujours réexécuter tous les filtres, même si les données n'avaient pas du tout changé. Dans Angular 2, une fois que la valeur d'un tube a été calculée, le détecteur de changement ne sait plus exécuter ce canal, à moins que l'entrée ne change.

Mise en place d'un tuyau à états

```

import {Pipe, PipeTransform, OnDestroy} from '@angular/core';

@Pipe({
  name: 'countdown',
  pure: false
})
export class CountdownPipe implements PipeTransform, OnDestroy {
  private interval: any;
  private remainingTime: number;

  transform(value: number, interval: number = 1000): number {
    if (!parseInt(value, 10)) {
      return null;
    }
  }
}

```

```

}

if (typeof this.remainingTime !== 'number') {
  this.remainingTime = parseInt(value, 10);
}

if (!this.interval) {
  this.interval = setInterval(() => {
    this.remainingTime--;

    if (this.remainingTime <= 0) {
      this.remainingTime = 0;
      clearInterval(this.interval);
      delete this.interval;
    }
  }, interval);
}

return this.remainingTime;
}

ngOnDestroy(): void {
  if (this.interval) {
    clearInterval(this.interval);
  }
}
}

```

Vous pouvez ensuite utiliser le tuyau comme d'habitude:

```

{{ 1000 | countdown:50 }}
{{ 300 | countdown }}

```

Il est important que votre pipe implémente également l'interface `OnDestroy` afin que vous puissiez nettoyer une fois votre pipe détruite. Dans l'exemple ci-dessus, il est nécessaire d'effacer l'intervalle pour éviter les fuites de mémoire.

Tuyau dynamique

Scénario d'utilisation: une vue de tableau se compose de différentes colonnes avec un format de données différent qui doit être transformé avec différents canaux.

table.component.ts

```

...
import { DYNAMIC_PIPEES } from '../pipes/dynamic.pipe.ts';

@Component({
  ...
  pipes: [DYNAMIC_PIPEES]
})
export class TableComponent {
  ...

  // pipes to be used for each column
  table.pipes = [ null, null, null, 'humanizeDate', 'statusFromBoolean' ],

```

```

table.header = [ 'id', 'title', 'url', 'created', 'status' ],
table.rows = [
  [ 1, 'Home', 'home', '2016-08-27T17:48:32', true ],
  [ 2, 'About Us', 'about', '2016-08-28T08:42:09', true ],
  [ 4, 'Contact Us', 'contact', '2016-08-28T13:28:18', false ],
  ...
]
...
}

```

dynamic.pipe.ts

```

import {
  Pipe,
  PipeTransform
} from '@angular/core';
// Library used to humanize a date in this example
import * as moment from 'moment';

@Pipe({name: 'dynamic'})
export class DynamicPipe implements PipeTransform {

  transform(value:string, modifier:string) {
    if (!modifier) return value;
    // Evaluate pipe string
    return eval('this.' + modifier + '(\'' + value + '\')')
  }

  // Returns 'enabled' or 'disabled' based on input value
  statusFromBoolean(value:string):string {
    switch (value) {
      case 'true':
      case '1':
        return 'enabled';
      default:
        return 'disabled';
    }
  }

  // Returns a human friendly time format e.g: '14 minutes ago', 'yesterday'
  humanizeDate(value:string):string {
    // Humanize if date difference is within a week from now else returns 'December 20,
    2016' format
    if (moment().diff(moment(value), 'days') < 8) return moment(value).fromNow();
    return moment(value).format('MMMM Do YYYY');
  }
}

export const DYNAMIC_PIPES = [DynamicPipe];

```

table.component.html

```

<table>
  <thead>
    <td *ngFor="let head of data.header">{{ head }}</td>
  </thead>
  <tr *ngFor="let row of table.rows; let i = index">
    <td *ngFor="let column of row">{{ column | dynamic:table.pipes[i] }}</td>
  </tr>
</table>

```

```
</tr>
</table>
```

Résultat

ID	Page Title	Page URL	Created	Status
1	Home	home	4 minutes ago	Enabled
2	About Us	about	Yesterday	Enabled
4	Contact Us	contact	Yesterday	Disabled

Tester un tuyau

Étant donné un tuyau qui inverse une chaîne

```
import { Pipe, PipeTransform } from '@angular/core';

@Pipe({ name: 'reverse' })
export class ReversePipe implements PipeTransform {
  transform(value: string): string {
    return value.split('').reverse().join('');
  }
}
```

Il peut être testé en configurant le fichier de spécifications comme ceci

```
import { TestBed, inject } from '@angular/core/testing';

import { ReversePipe } from './reverse.pipe';

describe('ReversePipe', () => {
  beforeEach(() => {
    TestBed.configureTestingModule({
      providers: [ReversePipe],
    });
  });

  it('should be created', inject([ReversePipe], (reversePipe: ReversePipe) => {
    expect(reversePipe).toBeTruthy();
  }));

  it('should reverse a string', inject([ReversePipe], (reversePipe: ReversePipe) => {
    expect(reversePipe.transform('abc')).toEqual('cba');
  }));
});
```

Lire Pipes en ligne: <https://riptutorial.com/fr/angular2/topic/1165/pipes>

Chapitre 59: redux angulaire

Exemples

De base

app.module.ts

```
import {appStoreProviders} from "../app.store";
providers : [
  ...
  appStoreProviders,
  ...
]
```

app.store.ts

```
import {InjectionToken} from '@angular/core';
import {createStore, Store, compose, StoreEnhancer} from 'redux';
import {AppState, default as reducer} from "../app.reducer";

export const AppStore = new InjectionToken('App.store');

const devtools: StoreEnhancer<AppState> =
  window['devToolsExtension'] ?
  window['devToolsExtension']() : f => f;

export function createAppStore(): Store<AppState> {
  return createStore<AppState>(
    reducer,
    compose(devtools)
  );
}

export const appStoreProviders = [
  {provide: AppStore, useFactory: createAppStore}
];
```

app.reducer.ts

```
export interface AppState {
  example : string
}

const rootReducer: Reducer<AppState> = combineReducers<AppState>({
  example : string
});

export default rootReducer;
```

store.ts


```

export interface IAppState {
  example?: string;
}

export const INITIAL_STATE: IAppState = {
  example: null,
};

export function rootReducer(state: IAppState = INITIAL_STATE, action: Action): IAppState {
  switch (action.type) {
    case EXAMPLE_CHANGED:
      return Object.assign(state, state, (<UpdateAction>action));
    default:
      return state;
  }
}

```

actions.ts

```

import {Action} from "redux";
export const EXAMPLE_CHANGED = 'EXAMPLE CHANGED';

export interface UpdateAction extends Action {
  example: string;
}

```

Obtenir l'état actuel

```

import * as Redux from 'redux';
import {Inject, Injectable} from '@angular/core';

@Injectable()
export class exampleService {
  constructor(@Inject(AppStore) private store: Redux.Store<AppState>) {}
  getExampleState(){
    console.log(this.store.getState().example);
  }
}

```

changer d'état

```

import * as Redux from 'redux';
import {Inject, Injectable} from '@angular/core';

@Injectable()
export class exampleService {
  constructor(@Inject(AppStore) private store: Redux.Store<AppState>) {}
  setExampleState(){
    this.store.dispatch(updateExample("new value"));
  }
}

```

actions.ts

```

export interface UpdateExapleAction extends Action {

```

```
    example?: string;
  }

export const updateExample: ActionCreator<UpdateExapleAction> =
  (newVal) => ({
    type: EXAMPLE_CHANGED,
    example: newVal
  });
```

Ajouter l'outil chrome redux

app.store.ts

```
import {InjectionToken} from '@angular/core';
import {createStore, Store, compose, StoreEnhancer} from 'redux';
import {AppState, default as reducer} from "../app.reducer";

export const AppStore = new InjectionToken('App.store');

const devtools: StoreEnhancer<AppState> =
  window['devToolsExtension'] ?
  window['devToolsExtension']() : f => f;

export function createAppStore(): Store<AppState> {
  return createStore<AppState>(
    reducer,
    compose(devtools)
  );
}

export const appStoreProviders = [
  {provide: AppStore, useFactory: createAppStore}
];
```

installer l'extension chrome de Redux DevTools

Lire redux angulaire en ligne: <https://riptutorial.com/fr/angular2/topic/10652/redux-angulaire>

Chapitre 60: Routage (3.0.0+)

Remarques

Il y a quelques astuces que nous pouvons faire avec le routeur (telles que la restriction de l'accès), mais celles-ci peuvent être abordées dans un didacticiel séparé.

Si vous avez besoin d'un nouvel itinéraire, modifiez simplement `app.routes.ts` et suivez les étapes suivantes:

1. Importer le composant
2. Ajouter au tableau des `routes`. Assurez-vous d'inclure un nouveau `path` et `component`.

Exemples

Bootstrap

Maintenant que les routes sont définies, nous devons informer notre application des itinéraires. Pour ce faire, démarrez le fournisseur que nous avons exporté dans l'exemple précédent.

Trouvez votre configuration bootstrap (devrait être dans `main.ts`, mais **votre kilométrage peut varier**).

```
//main.ts

import {bootstrap} from '@angular/platform-browser-dynamic';

//Import the App component (root component)
import { App } from './app/app';

//Also import the app routes
import { APP_ROUTES_PROVIDER } from './app/app.routes';

bootstrap(App, [
  APP_ROUTES_PROVIDER,
])
.catch(err => console.error(err));
```

Configuration du routeur

Maintenant que le routeur est configuré et que notre application sait comment gérer les itinéraires, nous devons afficher les composants réels que nous avons configurés.

Pour ce faire, configurez votre modèle / fichier HTML pour votre composant de **niveau supérieur (app)** comme suit:

```
//app.ts

import {Component} from '@angular/core';
```

```
import {Router, ROUTER_DIRECTIVES} from '@angular/router';

@Component({
  selector: 'app',
  templateUrl: 'app.html',
  styleUrls: ['app.css'],
  directives: [
    ROUTER_DIRECTIVES,
  ]
})
export class App {
  constructor() {
  }
}

<!-- app.html -->

<!-- All of your 'views' will go here -->
<router-outlet></router-outlet>
```

L'élément `<router-outlet></router-outlet>` changera le contenu de l'itinéraire. Un autre aspect intéressant de cet élément est qu'il ne *doit pas* nécessairement être le seul élément de votre code HTML.

Par exemple: Supposons que vous souhaitiez une barre d'outils sur chaque page qui reste constante entre les itinéraires, de la même manière que Stack Overflow. Vous pouvez imbriquer les `<router-outlet>` sous les éléments afin que seules certaines parties de la page changent.

Changer d'itinéraires (à l'aide de modèles et de directives)

Maintenant que les routes sont configurées, nous avons besoin d'un moyen de changer les routes.

Cet exemple montre comment modifier les itinéraires à l'aide du modèle, mais il est possible de modifier les itinéraires dans TypeScript.

Voici un exemple (sans engagement):

```
<a routerLink="/home">Home</a>
```

Si l'utilisateur clique sur ce lien, il sera acheminé vers `/home`. Le routeur sait comment gérer `/home`, il affichera donc le composant `Home`.

Voici un exemple de liaison de données:

```
<a *ngFor="let link of links" [routerLink]="link">{{link}}</a>
```

Ce qui nécessiterait un tableau appelé `links` pour exister, alors ajoutez ceci à `app.ts`:

```
public links[] = [
  'home',
  'login'
```

```
]
```

Cela `routerLink` le tableau et ajoutera un élément `<a>` avec la directive `routerLink` = la valeur de l'élément actuel dans le tableau, en créant ceci:

```
<a routerLink="home">home</a>
<a routerLink="login">login</a>
```

Cela est particulièrement utile si vous avez beaucoup de liens ou si les liens doivent être constamment modifiés. Nous avons laissé Angular gérer le travail acharné d'ajout de liens en lui fournissant simplement les informations dont il a besoin.

Actuellement, les `links[]` sont statiques, mais il est possible de les alimenter depuis une autre source.

Définir les itinéraires

REMARQUE: Cet exemple est basé sur la version 3.0.0.-beta.2 du @angular/router. Au moment de la rédaction de ce document, il s'agit de la dernière version du routeur.

Pour utiliser le routeur, définissez des itinéraires dans un nouveau fichier TypeScript comme celui-ci.

```
//app.routes.ts

import {provideRouter} from '@angular/router';

import {Home} from './routes/home/home';
import {Profile} from './routes/profile/profile';

export const routes = [
  {path: '', redirectTo: 'home'},
  {path: 'home', component: Home},
  {path: 'login', component: Login},
];

export const APP_ROUTES_PROVIDER = provideRouter(routes);
```

Dans la première ligne, nous importons `provideRouter` afin que nous puissions informer notre application des routes au cours de la phase de démarrage.

`Home` et `Profile` sont que deux composants à titre d'exemple. Vous devrez importer chaque `Component` vous avez besoin en tant que route.

Ensuite, exportez le tableau des routes.

`path` : chemin d'accès au composant. **VOUS N'AVEZ PAS BESOIN D'UTILISER '/'** Angular le fera automatiquement

`component` : composant à charger lors de l'accès à l'itinéraire

`redirectTo` : *Facultatif* . Si vous devez rediriger automatiquement un utilisateur lorsqu'il accède à un itinéraire particulier, indiquez-le.

Enfin, nous exportons le routeur configuré. `provideRouter` retournera un fournisseur que nous pouvons booster afin que notre application sache comment gérer chaque route.

Contrôle de l'accès à ou à partir d'une route

Le routeur angulaire par défaut permet la navigation vers et depuis n'importe quel itinéraire sans condition. Ce n'est pas toujours le comportement souhaité.

Dans un scénario où un utilisateur peut être autorisé, à titre conditionnel, à naviguer vers ou à partir d'une route, une fonction **Route Guard** peut être utilisée pour limiter ce comportement.

Si votre scénario correspond à l'un des suivants, envisagez d'utiliser une garde de route,

- L'utilisateur doit être authentifié pour accéder au composant cible.
- L'utilisateur doit être autorisé à accéder au composant cible.
- Le composant nécessite une requête asynchrone avant l'initialisation.
- Le composant nécessite une entrée utilisateur avant de quitter.

Comment fonctionnent les gardes de route

Les gardes de route fonctionnent en renvoyant une valeur booléenne pour contrôler le comportement de la navigation du routeur. Si *true* est renvoyé, le routeur continuera à naviguer vers le composant cible. Si *false* est renvoyé, le routeur refuse la navigation vers le composant cible.

Interfaces de garde de route

Le routeur prend en charge plusieurs interfaces de garde:

- *CanActivate* : se produit entre la navigation de route.
- *CanActivateChild* : se produit entre la navigation de l'itinéraire vers un itinéraire enfant.
- *CanDeactivate* : se produit lorsque vous vous éloignez de l'itinéraire actuel.
- *CanLoad* : se produit entre la navigation de l'itinéraire vers un module de fonctions chargé de manière asynchrone.
- *Résoudre* : permet d'effectuer une récupération de données avant l'activation de la route.

Ces interfaces peuvent être implémentées dans votre garde pour accorder ou supprimer l'accès à certains processus de la navigation.

Gardes de route synchrone vs asynchrone

Les gardes de route permettent aux opérations synchrones et asynchrones de contrôler conditionnellement la navigation.

Garde de route synchrone

Une protection de route synchrone renvoie un booléen, par exemple en calculant un résultat immédiat, afin de contrôler la navigation de manière conditionnelle.

```
import { Injectable }    from '@angular/core';
import { CanActivate }  from '@angular/router';

@Injectable()
export class SynchronousGuard implements CanActivate {
  canActivate() {
    console.log('SynchronousGuard#canActivate called');
    return true;
  }
}
```

Garde de route asynchrone

Pour un comportement plus complexe, une protection de route peut bloquer la navigation de manière asynchrone. Un protecteur d'itinéraire asynchrone peut renvoyer une observation ou une promesse.

Ceci est utile dans les situations telles que l'attente d'une entrée utilisateur pour répondre à une question, l'attente d'une sauvegarde réussie des modifications sur le serveur ou l'attente de recevoir des données extraites d'un serveur distant.

```
import { Injectable }    from '@angular/core';
import { CanActivate, Router, ActivatedRouteSnapshot, RouterStateSnapshot } from
 '@angular/router';
import { Observable }    from 'rxjs/Rx';
import { MockAuthenticationService } from './authentication/authentication.service';

@Injectable()
export class AsynchronousGuard implements CanActivate {
  constructor(private router: Router, private auth: MockAuthenticationService) {}

  canActivate(route: ActivatedRouteSnapshot,
state: RouterStateSnapshot): Observable<boolean>|boolean {
    this.auth.subscribe((authenticated) => {
      if (authenticated) {
        return true;
      }
      this.router.navigateByUrl('/login');
      return false;
    });
  }
}
```

```
    });  
  }  
}
```

Ajouter un garde à la configuration de l'itinéraire

Fichier *app.routes*

Les itinéraires protégés `canActivate` être `canActivate` sur `Guard`

```
import { provideRouter, Router, RouterConfig, CanActivate } from '@angular/router';  
  
//components  
import { LoginComponent } from '../login/login.component';  
import { DashboardComponent } from '../dashboard/dashboard.component';  
  
export const routes: RouterConfig = [  
  { path: 'login', component: LoginComponent },  
  { path: 'dashboard', component: DashboardComponent, canActivate: [AuthGuard] }  
]
```

Exportez les **APP_ROUTER_PROVIDERS** à utiliser dans le bootstrap de l'application

```
export const APP_ROUTER_PROVIDERS = [  
  AuthGuard,  
  provideRouter(routes)  
];
```

Utilisez Guard dans l'application bootstrap

Fichier *main.ts* (ou *boot.ts*)

Considérons les exemples ci-dessus:

1. **Créer la garde** (où la garde est créée) et
2. **Ajouter la garde à la configuration de la route**, (où la garde est configurée pour la route, alors **APP_ROUTER_PROVIDERS** est exporté), nous pouvons coupler le bootstrap à Guard comme suit

```
import { bootstrap } from '@angular/platform-browser-dynamic';  
import { provide } from '@angular/core';  
  
import { APP_ROUTER_PROVIDERS } from './app.routes';  
import { AppComponent } from './app.component';  
  
bootstrap(AppComponent, [  
  APP_ROUTER_PROVIDERS  
)  
.then(success => console.log(`Bootstrap success`))  
.catch(error => console.log(error));
```

Utiliser des résolveurs et des gardes

Nous utilisons une protection de niveau supérieur dans notre route config pour intercepter l'utilisateur actuel sur la première page et un résolveur pour stocker la valeur de `currentUser`, qui est notre utilisateur authentifié du serveur.

Une version simplifiée de notre implémentation se présente comme suit:

Voici notre itinéraire de haut niveau:

```
export const routes = [
  {
    path: 'Dash',
    pathMatch: 'prefix',
    component: DashCmp,
    canActivate: [AuthGuard],
    resolve: {
      currentUser: CurrentUserResolver
    },
    children: [...[
      {
        path: '',
        component: ProfileCmp,
        resolve: {
          currentUser: currentUser
        }
      }
    ]]
  }
];
```

Voici notre `AuthService`

```
import { Injectable } from '@angular/core';
import { Http, Headers, RequestOptions } from '@angular/http';
import { Observable } from 'rxjs/Rx';
import 'rxjs/add/operator/do';

@Injectable()
export class AuthService {
  constructor(http: Http) {
    this.http = http;
  }

  let headers = new Headers({ 'Content-Type': 'application/json' });
  this.options = new RequestOptions({ headers: headers });
}

fetchCurrentUser() {
  return this.http.get('/api/users/me')
    .map(res => res.json())
    .do(val => this.currentUser = val);
}
}
```

Voici notre `AuthGuard` :

```
import { Injectable } from '@angular/core';
import { CanActivate } from "@angular/router";
import { Observable } from 'rxjs/Rx';

import { AuthService } from '../services/AuthService';
```

```

@Injectable()
export class AuthGuard implements CanActivate {
  constructor(auth: AuthService) {
    this.auth = auth;
  }
  canActivate(route, state) {
    return Observable
      .merge(this.auth.fetchCurrentUser(), Observable.of(true))
      .filter(x => x == true);
  }
}

```

Voici notre `CurrentUserResolver` :

```

import { Injectable } from '@angular/core';
import { Resolve } from "@angular/router";
import { Observable } from 'rxjs/Rx';

import { AuthService } from '../services/AuthService';

@Injectable()
export class CurrentUserResolver implements Resolve {
  constructor(auth: AuthService) {
    this.auth = auth;
  }
  resolve(route, state) {
    return this.auth.currentUser;
  }
}

```

Lire Routage (3.0.0+) en ligne: <https://riptutorial.com/fr/angular2/topic/1208/routage--3-0-0plus->

Chapitre 61: Service EventEmitter

Exemples

Aperçu de la classe

```
class EventEmitter extends Subject {
  constructor(isAsync?: boolean)
  emit(value?: T)
  subscribe(generatorOrNext?: any, error?: any, complete?: any) : any
}
```

Composant de classe

```
@Component({
  selector: 'zippy',
  template: `
    <div class="zippy">
      <div (click)="toggle()">Toggle</div>
      <div [hidden]="!visible">
        <ng-content></ng-content>
      </div>
    </div>`})
export class Zippy {
  visible: boolean = true;
  @Output() open: EventEmitter<any> = new EventEmitter();
  @Output() close: EventEmitter<any> = new EventEmitter();
  toggle() {
    this.visible = !this.visible;
    if (this.visible) {
      this.open.emit(null);
    } else {
      this.close.emit(null);
    }
  }
}
```

Événements émouvants

```
<zippy (open)="onOpen($event)" (close)="onClose($event)"></zippy>
```

Attraper l'événement

Créer un service-

```
import {EventEmitter} from 'angular2/core';
export class NavService {
  navchange: EventEmitter<number> = new EventEmitter();
  constructor() {}
  emitNavChangeEvent(number) {
    this.navchange.emit(number);
  }
}
```

```

    }
    getNavChangeEmitter() {
        return this.navchange;
    }
}

```

Créer un composant pour utiliser le service-

```

import {Component} from 'angular2/core';
import {NavService} from '../services/NavService';

@Component({
  selector: 'obs-comp',
  template: `obs component, item: {{item}}`
})
export class ObservingComponent {
  item: number = 0;
  subscription: any;
  constructor(private navService:NavService) {}
  ngOnInit() {
    this.subscription = this.navService.getNavChangeEmitter()
      .subscribe(item => this.selectedNavItem(item));
  }
  selectedNavItem(item: number) {
    this.item = item;
  }
  ngOnDestroy() {
    this.subscription.unsubscribe();
  }
}

@Component({
  selector: 'my-nav',
  template: `
    <div class="nav-item" (click)="selectedNavItem(1)">nav 1 (click me)</div>
    <div class="nav-item" (click)="selectedNavItem(2)">nav 2 (click me)</div>
  `,
})
export class Navigation {
  item = 1;
  constructor(private navService:NavService) {}
  selectedNavItem(item: number) {
    console.log('selected nav item ' + item);
    this.navService.emitNavChangeEvent(item);
  }
}

```

Exemple en direct

Un exemple concret de ceci peut être trouvé [ici](#) .

Lire Service EventEmitter en ligne: <https://riptutorial.com/fr/angular2/topic/9159/service-eventemitter>

Chapitre 62: Services et injection de dépendance

Exemples

Exemple de service

services / my.service.ts

```
import { Injectable } from '@angular/core';

@Injectable()
export class MyService {
  data: any = [1, 2, 3];

  getData() {
    return this.data;
  }
}
```

L'inscription du fournisseur de services dans la méthode bootstrap rendra le service disponible globalement.

main.ts

```
import { bootstrap } from '@angular/platform-browser-dynamic';
import { AppComponent } from 'app.component.ts';
import { MyService } from 'services/my.service';

bootstrap(AppComponent, [MyService]);
```

Dans la version RC5, l'inscription du fournisseur de services global peut être effectuée à l'intérieur du fichier de module. Afin d'obtenir une instance unique de votre service pour l'ensemble de votre application, le service doit être déclaré dans la liste des fournisseurs du module ng de votre application. *app_module.ts*

```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { routing, appRoutingProviders } from './app-routes/app.routes';
import { HttpClientModule } from '@angular/http';

import { AppComponent } from './app.component';
import { MyService } from 'services/my.service';

import { routing } from './app-resources/app-routes/app.routes';

@NgModule({
  declarations: [ AppComponent ],
  imports: [ BrowserModule,
            routing,
            HttpClientModule ]
})
```

```

        RouterModule,
        HttpModule ],
    providers: [
        appRoutingProviders,
        MyService
    ],
    bootstrap: [AppComponent],
})
export class AppModule {}

```

Utilisation dans `MyComponent`

composants / my.component.ts

Approche alternative pour enregistrer les fournisseurs d'applications dans les composants d'application. Si nous ajoutons des fournisseurs au niveau du composant chaque fois que le composant est rendu, cela créera une nouvelle instance du service.

```

import { Component, OnInit } from '@angular/core';
import { MyService } from '../services/my.service';

@Component({
  ...
  providers:[MyService] //
})
export class MyComponent implements OnInit {
  data: any[];
  // Creates private variable myService to use, of type MyService
  constructor(private myService: MyService) { }

  ngOnInit() {
    this.data = this.myService.getData();
  }
}

```

Exemple avec `Promise.resolve`

services / my.service.ts

```

import { Injectable } from '@angular/core';

@Injectable()
export class MyService {
  data: any = [1, 2, 3];

  getData() {
    return Promise.resolve(this.data);
  }
}

```

`getData()` agit désormais comme un appel REST qui crée une promesse, qui est résolue immédiatement. Les résultats peuvent être `.then()` à la main dans `.then()` et des erreurs peuvent également être détectées. C'est une bonne pratique et une convention pour les méthodes asynchrones.

composants / my.component.ts

```
import { Component, OnInit } from '@angular/core';
import { MyService } from '../services/my.service';

@Component({...})
export class MyComponent implements OnInit {
  data: any[];
  // Creates private variable myService to use, of type MyService
  constructor(private myService: MyService) { }

  ngOnInit() {
    // Uses an "arrow" function to set data
    this.myService.getData().then(data => this.data = data);
  }
}
```

Tester un service

Étant donné un service qui peut se connecter à un utilisateur:

```
import 'rxjs/add/operator/toPromise';

import { Http } from '@angular/http';
import { Injectable } from '@angular/core';

interface LoginCredentials {
  password: string;
  user: string;
}

@Injectable()
export class AuthService {
  constructor(private http: Http) { }

  async signIn({ user, password }: LoginCredentials) {
    const response = await this.http.post('/login', {
      password,
      user,
    }).toPromise();

    return response.json();
  }
}
```

Il peut être testé comme ceci:

```
import { ConnectionBackend, Http, HttpModule, Response, ResponseOptions } from
 '@angular/http';
import { TestBed, async, inject } from '@angular/core/testing';

import { AuthService } from './auth.service';
import { MockBackend } from '@angular/http/testing';
import { MockConnection } from '@angular/http/testing';

describe('AuthService', () => {
  beforeEach(() => {
```

```

TestBed.configureTestingModule({
  imports: [HttpModule],
  providers: [
    AuthService,
    Http,
    { provide: ConnectionBackend, useClass: MockBackend },
  ]
});

it('should be created', inject([AuthService], (service: AuthService) => {
  expect(service).toBeTruthy();
}));

// Alternative 1
it('should login user if right credentials are passed', async(
  inject([AuthService], async (authService) => {
    const backend: MockBackend = TestBed.get(ConnectionBackend);
    const http: Http = TestBed.get(Http);

    backend.connections.subscribe((c: MockConnection) => {
      c.mockRespond(
        new Response(
          new ResponseOptions({
            body: {
              accessToken: 'abcdef',
            },
          }),
        ),
      );
    });

    const result = await authService.signIn({ password: 'ok', user: 'bruno' });

    expect(result).toEqual({
      accessToken: 'abcdef',
    });
  })
);

// Alternative 2
it('should login user if right credentials are passed', async () => {
  const backend: MockBackend = TestBed.get(ConnectionBackend);
  const http: Http = TestBed.get(Http);

  backend.connections.subscribe((c: MockConnection) => {
    c.mockRespond(
      new Response(
        new ResponseOptions({
          body: {
            accessToken: 'abcdef',
          },
        }),
      ),
    );
  });

  const authService: AuthService = TestBed.get(AuthService);

  const result = await authService.signIn({ password: 'ok', user: 'bruno' });

```



```

    expect(result).toEqual({
      accessToken: 'abcdef',
    });
  });

  // Alternative 3
  it('should login user if right credentials are passed', async (done) => {
    const authService: AuthService = TestBed.get(AuthService);

    const backend: MockBackend = TestBed.get(ConnectionBackend);
    const http: Http = TestBed.get(Http);

    backend.connections.subscribe((c: MockConnection) => {
      c.mockRespond(
        new Response(
          new ResponseOptions({
            body: {
              accessToken: 'abcdef',
            },
          }),
        ),
      );
    });

    try {
      const result = await authService.signIn({ password: 'ok', user: 'bruno' });

      expect(result).toEqual({
        accessToken: 'abcdef',
      });

      done();
    } catch (err) {
      fail(err);
      done();
    }
  });
});

```

Lire Services et injection de dépendance en ligne:

<https://riptutorial.com/fr/angular2/topic/4187/services-et-injection-de-dependance>

Chapitre 63: Sujets et observables angulaires RXJS avec requêtes API

Remarques

Faire des requêtes API avec le service HTTP Angular 2 et RxJS est très similaire à travailler avec des promesses dans Angular 1.x.

Utilisez la classe `Http` pour faire des requêtes. La classe `Http` expose les méthodes d'émission des requêtes HTTP `GET`, `POST`, `PUT`, `DELETE`, `PATCH`, les requêtes `HEAD` via les méthodes correspondantes. Il expose également une méthode de `request` générique pour émettre tout type de requête HTTP.

Toutes les méthodes de la classe `Http` renvoient une `Observable<Response>` à laquelle vous pouvez appliquer des [opérations RxJS](#). Vous appelez la méthode `.subscribe()` et transmettez une fonction à appeler lorsque des données sont renvoyées dans le flux `Observable`.

Le flux observable pour une requête ne contient qu'une seule valeur: la `Response`, et se termine / se règle lorsque la requête HTTP est terminée avec succès ou en cas d'erreur ou d'erreur en cas d'erreur.

Notez que les observables renvoyées par le module `Http` sont *froides*, ce qui signifie que si vous vous abonnez à l'observable plusieurs fois, la requête d'origine sera *exécutée* une fois pour chaque abonnement. Cela peut se produire si vous souhaitez consommer le résultat dans plusieurs composants de votre application. Pour les requêtes `GET`, cela peut provoquer des requêtes supplémentaires, mais cela peut créer des résultats inattendus si vous vous abonnez plus d'une fois à des requêtes `PUT` ou `POST`.

Exemples

Demande de base

L'exemple suivant illustre une simple requête HTTP `GET`. `http.get()` renvoie un objet `Observable` auquel la méthode `subscribe` est appliquée. Celui-ci ajoute les données renvoyées au tableau des `posts`.

```
var posts = []

getPosts(http: Http):void {
  this.http.get(`https://jsonplaceholder.typicode.com/posts`)
    .map(response => response.json())
    .subscribe(post => posts.push(post));
}
```

Encapsulation des requêtes API

Il peut être judicieux d'encapsuler la logique de gestion HTTP dans sa propre classe. La classe suivante expose une méthode pour obtenir des messages. Il appelle la méthode `http.get()` et appelle `.map` sur l'Observable renvoyé pour convertir l'objet `Response` objet `Post`.

```
import {Injectable} from "@angular/core";
import {Http, Response} from "@angular/http";

@Injectable()
export class BlogApi {

  constructor(private http: Http) {
  }

  getPost(id: number): Observable<Post> {
    return this.http.get(`https://jsonplaceholder.typicode.com/posts/${id}`)
      .map((response: Response) => {
        const srcData = response.json();
        return new Post(srcData)
      });
  }
}
```

L'exemple précédent utilise une classe `Post` pour contenir les données renvoyées, qui peuvent ressembler à ceci:

```
export class Post {
  userId: number;
  id: number;
  title: string;
  body: string;

  constructor(src: any) {
    this.userId = src && src.userId;
    this.id = src && src.id;
    this.title = src && src.title;
    this.body = src && src.body;
  }
}
```

Un composant peut maintenant utiliser la classe `BlogApi` pour récupérer facilement des données `Post` sans se préoccuper du fonctionnement de la classe `Http`.

Attendez plusieurs demandes

Un scénario courant consiste à attendre que plusieurs requêtes se terminent avant de continuer. Cela peut être accompli en utilisant la [méthode](#) `forkJoin`.

Dans l'exemple suivant, `forkJoin` est utilisé pour appeler deux méthodes qui renvoient des Observables. Le rappel spécifié dans la méthode `.subscribe` sera appelé lorsque les deux observables seront terminés. Les paramètres fournis par `.subscribe` correspondent à l'ordre donné dans l'appel à `.forkJoin`. Dans ce cas, `posts` `abond` `posts` `tags`.

```
loadData() : void {
  Observable.forkJoin(
```

```
    this.blogApi.getPosts(),
    this.blogApi.getTags()
  ).subscribe((([posts, tags]: [Post[], Tag[]]) => {
    this.posts = posts;
    this.tags = tags;
  }));
}
```

Lire Sujets et observables angulaires RXJS avec requêtes API en ligne:

<https://riptutorial.com/fr/angular2/topic/3577/sujets-et-observables-angulaires-rxjs-avec-requetes-api>

Chapitre 64: Test d'une application angulaire

2

Exemples

Installation du framework de test Jasmine

Le moyen le plus courant de tester les applications Angular 2 est d'utiliser le framework de test Jasmine. Jasmine vous permet de tester votre code dans le navigateur.

Installer

Pour commencer, tout ce dont vous avez besoin est le paquet `jasmine-core` (pas le `jasmine`).

```
npm install jasmine-core --save-dev --save-exact
```

Vérifier

Pour vérifier que Jasmine est correctement configuré, créez le fichier `./src/unit-tests.html` avec le contenu suivant et ouvrez-le dans le navigateur.

```
<!DOCTYPE html>
<html>
<head>
  <meta http-equiv="content-type" content="text/html; charset=utf-8">
  <title>Ng App Unit Tests</title>
  <link rel="stylesheet" href="../node_modules/jasmine-core/lib/jasmine-core/jasmine.css">
  <script src="../node_modules/jasmine-core/lib/jasmine-core/jasmine.js"></script>
  <script src="../node_modules/jasmine-core/lib/jasmine-core/jasmine-html.js"></script>
  <script src="../node_modules/jasmine-core/lib/jasmine-core/boot.js"></script>
</head>
<body>
  <!-- Unit Testing Chapter #1: Proof of life. -->
  <script>
    it('true is true', function () {
      expect(true).toEqual(true);
    });
  </script>
</body>
</html>
```

Mise en place de tests avec Gulp, Webpack, Karma et Jasmine

La première chose dont nous avons besoin est de dire à karma d'utiliser Webpack pour lire nos tests, dans une configuration que nous avons définie pour le moteur WebPack. Ici, j'utilise babel

parce que j'écris mon code dans ES6, vous pouvez changer cela pour d'autres saveurs, telles que Typescript. Ou j'utilise les gabarits Pug (anciennement Jade), pas nécessaire.

Pourtant, la stratégie reste la même.

Donc, ceci est une configuration webpack:

```
const webpack = require("webpack");
let packConfig = {
  entry: {},
  output: {},
  plugins: [
    new webpack.DefinePlugin({
      ENVIRONMENT: JSON.stringify('test')
    })
  ],
  module: {
    loaders: [
      {
        test: /\.js$/,
        exclude: /(node_modules|bower_components)/,
        loader: "babel",
        query: {
          presets: ["es2015", "angular2"]
        }
      },
      {
        test: /\.woff2?$|\.ttf$|\.eot$|\.svg$/,
        loader: "file"
      },
      {
        test: /\.scss$/,
        loaders: ["style", "css", "sass"]
      },
      {
        test: /\.pug$/,
        loader: 'pug-html-loader'
      }
    ],
    devtool : 'inline-cheap-source-map'
  };
module.exports = packConfig;
```

Et puis, nous avons besoin d'un fichier karma.config.js pour utiliser cette configuration webpack:

```
const packConfig = require("../webpack.config.js");
module.exports = function (config) {
  config.set({
    basePath: '',
    frameworks: ['jasmine'],
    exclude: [],
    files: [
      {pattern: './karma.shim.js', watched: false}
    ],
    preprocessors: {
      './karma.shim.js': ["webpack"]
    }
  });
};
```

```

    },
    webpack: packConfig,

    webpackServer: {noInfo: true},

    port: 9876,

    colors: true,

    logLevel: config.LOG_INFO,

    browsers: ['PhantomJS'],

    concurrency: Infinity,

    autoWatch: false,
    singleRun: true
  });
};

```

Jusqu'à présent, nous avons demandé à Karma d'utiliser webpack, et nous lui avons dit de commencer par un fichier appelé **karma.shim.js** . Ce fichier aura pour fonction de servir de point de départ pour Webpack. webpack lira ce fichier et utilisera l' **importation** et **demandera des** instructions pour rassembler toutes nos dépendances et exécuter nos tests.

Alors maintenant, regardons le fichier karma.shim.js:

```

// Start of ES6 Specific stuff
import "es6-shim";
import "es6-promise";
import "reflect-metadata";
// End of ES6 Specific stuff

import "zone.js/dist/zone";
import "zone.js/dist/long-stack-trace-zone";
import "zone.js/dist/jasmine-patch";
import "zone.js/dist/async-test";
import "zone.js/dist/fake-async-test";
import "zone.js/dist/sync-test";
import "zone.js/dist/proxy-zone";

import 'rxjs/add/operator/map';
import 'rxjs/add/observable/of';

Error.stackTraceLimit = Infinity;

import {TestBed} from "@angular/core/testing";
import { BrowserDynamicTestingModule, platformBrowserDynamicTesting} from "@angular/platform-browser-dynamic/testing";

TestBed.initTestEnvironment(
  BrowserDynamicTestingModule,
  platformBrowserDynamicTesting());

let testContext = require.context('../src/app', true, /\.spec\.js/);
testContext.keys().forEach(testContext);

```

Essentiellement, nous importons **TestBed** à partir de tests de base angulaires et nous **lançons**

l'environnement, car il ne doit être lancé qu'une seule fois pour tous nos tests. Ensuite, nous **parcourons le répertoire src / app de** manière récursive et lisons tous les fichiers qui se terminent par **.spec.js** et les transmettons à testContext afin qu'ils s'exécutent.

J'essaie généralement de mettre mes tests au même endroit que la classe. Personnellement, cela me permet d'importer des dépendances et des tests de refactorisation avec des classes. Mais si vous voulez placer vos tests ailleurs, comme dans le répertoire **src / test** par exemple, voici votre chance. changer la ligne avant dernière dans le fichier karma.shim.js.

Parfait. ce qui reste? ah, la tâche gulp qui utilise le fichier karma.config.js que nous avons créé ci-dessus:

```
gulp.task("karmaTests",function(done){
  var Server = require("karma").Server;
  new Server({
    configFile : "./karma.config.js",
    singleRun: true,
    autoWatch: false
  }, function(result){
    return result ? done(new Error(`Karma failed with error code ${result}`)):done();
  }).start();
});
```

Je suis en train de démarrer le serveur avec le fichier de configuration que nous avons créé, en lui disant de s'exécuter une fois et de ne pas surveiller les modifications. Je trouve que cela me convient mieux car les tests ne seront exécutés que si je suis prêt à les exécuter, mais bien sûr, si vous voulez des différences, vous savez où changer.

Et comme dernier exemple de code, voici une série de tests pour le tutoriel Angular 2, "Tour of Heroes".

```
import {
  TestBed,
  ComponentFixture,
  async
} from "@angular/core/testing";

import {AppComponent} from "./app.component";
import {AppModule} from "./app.module";
import Hero from "./hero/hero";

describe("App Component", function () {

  beforeEach(() => {
    TestBed.configureTestingModule({
      imports: [AppModule]
    });

    this.fixture = TestBed.createComponent(AppComponent);
    this.fixture.detectChanges();
  });

  it("Should have a title", async(() => {
    this.fixture.whenStable().then(() => {
      expect(this.fixture.componentInstance.title).toEqual("Tour of Heros");
    });
  }));
});
```



```

    });
  });

it("Should have a hero", async(()=> {
  this.fixture.whenStable().then(()=> {
    expect(this.fixture.componentInstance.selectedHero).toBeNull();
  });
}));

it("Should have an array of heros", async(()=>
  this.fixture.whenStable().then(()=> {
    const cmp = this.fixture.componentInstance;
    expect(cmp.heroes).toBeDefined("component should have a list of heroes");
    expect(cmp.heroes.length).toEqual(10, "heroes list should have 10 members");
    cmp.heroes.map((h, i)=> {
      expect(h instanceof Hero).toBeTruthy(`member ${i} is not a Hero instance.
${h}`)
    });
  }));

it("Should have one list item per hero", async(()=>
  this.fixture.whenStable().then(()=> {
    const ul = this.fixture.nativeElement.querySelector("ul.heroes");
    const li = Array.prototype.slice.call(
      this.fixture.nativeElement.querySelectorAll("ul.heroes>li"));
    const cmp = this.fixture.componentInstance;
    expect(ul).toBeTruthy("There should be an unnumbered list for heroes");
    expect(li.length).toEqual(cmp.heroes.length, "there should be one li for each
hero");
    li.forEach((li, i)=> {
      expect(li.querySelector("span.badge"))
        .toBeTruthy(`hero ${i} has to have a span for id`);
      expect(li.querySelector("span.badge").textContent.trim())
        .toEqual(cmp.heroes[i].id.toString(), `hero ${i} had wrong id displayed`);
      expect(li.textContent)
        .toMatch(cmp.heroes[i].name, `hero ${i} has wrong name displayed`);
    });
  }));

it("should have correct styling of hero items", async(()=>
  this.fixture.whenStable().then(()=> {
    const hero = this.fixture.nativeElement.querySelector("ul.heroes>li");
    const win = hero.ownerDocument.defaultView || hero.ownerDocument.parentWindow;
    const styles = win.getComputedStyle(hero);
    expect(styles["cursor"]).toEqual("pointer", "cursor should be pointer on hero");
    expect(styles["borderRadius"]).toEqual("4px", "borderRadius should be 4px");
  }));

it("should have a click handler for hero items", async(()=>
  this.fixture.whenStable().then(()=>{
    const cmp = this.fixture.componentInstance;
    expect(cmp.onSelect)
      .toBeDefined("should have a click handler for heros");
    expect(this.fixture.nativeElement.querySelector("input.heroName"))
      .toBeNull("should not show the hero details when no hero has been selected");
    expect(this.fixture.nativeElement.querySelector("ul.heroes li.selected"))
      .toBeNull("Should not have any selected heroes at start");

    spyOn(cmp, "onSelect").and.callThrough();
    this.fixture.nativeElement.querySelectorAll("ul.heroes li")[5].click();
  }));

```

```

    expect(cmp.onSelect)
      .toHaveBeenCalledWith(cmp.heroes[5]);
    expect(cmp.selectedHero)
      .toEqual(cmp.heroes[5], "click on hero should change hero");
  })
});
});

```

Il est **intéressant de noter** que **beforeEach ()** configure un module de test et crée le composant dans test, et comment nous appelons **detectChanges ()** afin que angular passe réellement par la double liaison et tout.

Notez que chaque test est un appel à **async ()** et qu'il attend toujours que promis **stable soit** résolu avant d'examiner le projecteur. Il a ensuite accès au composant via **componentInstance** et à l'élément via **nativeElement** .

Il y a un test qui vérifie le style correct. Dans le cadre du didacticiel, l'équipe angulaire démontre l'utilisation de styles à l'intérieur des composants. Dans notre test, nous utilisons **getComputedStyle ()** pour vérifier que les styles proviennent de l'endroit spécifié, mais nous avons besoin de l'objet Window pour cela, et nous l'obtenons de l'élément comme vous pouvez le voir dans le test.

Test du service HTTP

Généralement, les services appellent Api distant pour récupérer / envoyer des données. Mais les tests unitaires ne devraient pas faire appel au réseau. Angular utilise en `XHRBackend` classe `XHRBackend` pour effectuer des requêtes http. L'utilisateur peut remplacer cela pour modifier le comportement. Le module de test angulaire fournit les classes `MockBackend` et `MockConnection` qui peuvent être utilisées pour tester et confirmer les requêtes http.

`posts.service.ts` Ce service frappe un noeud final api pour extraire la liste des publications.

```

import { Http } from '@angular/http';
import { Injectable } from '@angular/core';
import { Observable } from 'rxjs/rx';

import 'rxjs/add/operator/map';

export interface IPost {
  userId: number;
  id: number;
  title: string;
  body: string;
}

@Injectable()
export class PostsService {
  posts: IPost[];

  private postsUri = 'http://jsonplaceholder.typicode.com/posts';

  constructor(private http: Http) {
  }
}

```

```

get(): Observable<IPost[]> {
  return this.http.get(this.postsUri)
    .map((response) => response.json());
}
}

```

posts.service.spec.ts Ici, nous allons tester le service ci-dessus en moquant les appels d'API http.

```

import { TestBed, inject, fakeAsync } from '@angular/core/testing';
import {
  HttpClientModule,
  XHRBackend,
  ResponseOptions,
  Response,
  RequestMethod
} from '@angular/http';
import {
  MockBackend,
  MockConnection
} from '@angular/http/testing';

import { PostsService } from './posts.service';

describe('PostsService', () => {
  // Mock http response
  const mockResponse = [
    {
      'userId': 1,
      'id': 1,
      'title': 'sunt aut facere repellat provident occaecati excepturi optio reprehenderit',
      'body': 'quia et suscipit\nsuscipit recusandae consequuntur expedita et cum\nreprehenderit molestiae ut ut quas totam\nnostrum rerum est autem sunt rem eveniet architecto'
    },
    {
      'userId': 1,
      'id': 2,
      'title': 'qui est esse',
      'body': 'est rerum tempore vitae\nsequi sint nihil reprehenderit dolor beatae ea dolores neque\nfugiat blanditiis voluptate porro vel nihil molestiae ut reiciendis\nqui aperiam non debitis possimus qui neque nisi nulla'
    },
    {
      'userId': 1,
      'id': 3,
      'title': 'ea molestias quasi exercitationem repellat qui ipsa sit aut',
      'body': 'et iusto sed quo iure\nvoluptatem occaecati omnis eligendi aut ad\nvoluptatem doloribus vel accusantium quis pariatur\nmolestiae porro eius odio et labore et velit aut'
    },
    {
      'userId': 1,
      'id': 4,
      'title': 'eum et est occaecati',
      'body': 'ullam et saepe reiciendis voluptatem adipisci\nsit amet autem assumenda provident rerum culpa\nquis hic commodi nesciunt rem tenetur doloremque ipsam iure\nquis sunt voluptatem rerum illo velit'
    }
  ];
};

```

```

beforeEach(() => {
  TestBed.configureTestingModule({
    imports: [HttpModule],
    providers: [
      {
        provide: XHRBackend,
        // This provides mocked XHR backend
        useClass: MockBackend
      },
      PostsService
    ]
  });
});

it('should return posts retrieved from Api', fakeAsync(
  inject([XHRBackend, PostsService],
    (mockBackend, postsService) => {
      mockBackend.connections.subscribe(
        (connection: MockConnection) => {
          // Assert that service has requested correct url with expected method
          expect(connection.request.method).toBe(RequestMethod.Get);

expect(connection.request.url).toBe('http://jsonplaceholder.typicode.com/posts');
          // Send mock response
          connection.mockRespond(new Response(new ResponseOptions({
            body: mockResponse
          })));
        });

      postsService.get()
        .subscribe((posts) => {
          expect(posts).toBe(mockResponse);
        });

    }));
});

```

Test des composants angulaires - De base

Le code composant est donné ci-dessous.

```

import { Component } from '@angular/core';

@Component({
  selector: 'my-app',
  template: '<h1>{{title}}</h1>'
})
export class MyAppComponent {
  title = 'welcome';
}

```

Pour les tests angulaires, angular fournit ses utilitaires de test avec le framework de test, ce qui aide à écrire le bon scénario de test en angulaire. Les utilitaires angulaires peuvent être importés de `@angular/core/testing`

```

import { ComponentFixture, TestBed } from '@angular/core/testing';

```

```

import { MyAppComponent } from './banner-inline.component';

describe('Tests for MyAppComponent', () => {

  let fixture: ComponentFixture<MyAppComponent>;
  let comp: MyAppComponent;

  beforeEach(() => {
    TestBed.configureTestingModule({
      declarations: [
        MyAppComponent
      ]
    });
  });

  beforeEach(() => {

    fixture = TestBed.createComponent(MyAppComponent);
    comp = fixture.componentInstance;

  });

  it('should create the MyAppComponent', () => {

    expect(comp).toBeTruthy();

  });

});

```

Dans l'exemple ci-dessus, il n'y a qu'un seul scénario de test qui explique le scénario de test pour l'existence d'un composant. Dans l'exemple ci-dessus, des utilitaires de test angulaire tels que `TestBed` et `ComponentFixture` sont utilisés.

`TestBed` est utilisé pour créer le module de test angulaire et nous configurons ce module avec la méthode `configureTestingModule` pour produire l'environnement de module pour la classe que nous voulons tester. Module de test à configurer avant l'exécution de chaque test `beforeEach` c'est pourquoi nous configurons le module de test dans la fonction `beforeEach`.

méthode `createComponent` de `TestBed` est utilisée pour créer l'instance du composant sous test. `createComponent` renvoie le `ComponentFixture`. L'appareil donne accès à l'instance du composant lui-même.

Lire Test d'une application angulaire 2 en ligne: <https://riptutorial.com/fr/angular2/topic/2329/test-d-une-application-angulaire-2>

Chapitre 65: test unitaire

Exemples

Test élémentaire

fichier de composant

```
@Component ({
  selector: 'example-test-compnent',
  template: '<div>
    <div>{{user.name}}</div>
    <div>{{user.fname}}</div>
    <div>{{user.email}}</div>
  </div>'
})

export class ExampleTestComponent implements OnInit{

  let user :User = null;
  ngOnInit(): void {
    this.user.name = 'name';
    this.user.fname= 'fname';
    this.user.email= 'email';
  }

}
```

Fichier de test

```
describe('Example unit test component', () => {
  let component: ExampleTestComponent ;
  let fixture: ComponentFixture<ExampleTestComponent >;

  beforeEach(async(() => {
    TestBed.configureTestingModule({
      declarations: [ExampleTestComponent]
    }).compileComponents();
  }));

  beforeEach(() => {
    fixture = TestBed.createComponent(ExampleTestComponent );
    component = fixture.componentInstance;
    fixture.detectChanges();
  });

  it('ngOnInit should change user object values', () => {
    expect(component.user).toBeNull(); // check that user is null on initialize
    component.ngOnInit(); // run ngOnInit

    expect (component.user.name).toEqual('name');
    expect (component.user.fname).toEqual('fname');
```

```
    expect(component.user.email).toEqual('email');  
  });  
});
```

Lire test unitaire en ligne: <https://riptutorial.com/fr/angular2/topic/8955/test-unitaire>

Chapitre 66: Tester ngModel

Introduction

Est un exemple de la façon dont vous pouvez tester un composant dans Angular2 qui a un ngModel.

Exemples

Test de base

```
import { BrowserModule } from '@angular/platform-browser';
import { CommonModule } from '@angular/common';
import { FormsModule } from '@angular/forms';
import { HttpClientModule } from '@angular/http';

import { Component, DebugElement } from '@angular/core';
import { dispatchEvent } from "@angular/platform-browser/testing/browser_util";
import { TestBed, ComponentFixture } from '@angular/core/testing';
import { By } from "@angular/platform-browser";

import { MyComponentModule } from 'ng2-my-component';
import { MyComponent } from './my-component';

describe('MyComponent:', () => {

  const template = `
    <div>
      <my-component type="text" [(ngModel)]="value" name="TestName" size="9" min="3" max="8"
placeholder="testPlaceholder" disabled=false required=false></my-component>
    </div>
  `;

  let fixture:any;
  let element:any;
  let context:any;

  beforeEach(() => {

    TestBed.configureTestingModule({
      declarations: [InlineEditorComponent],
      imports: [
        FormsModule,
        InlineEditorModule]
    });
    fixture = TestBed.overrideComponent(InlineEditorComponent, {
      set: {
        selector:"inline-editor-test",
        template: template
      }
    })
    .createComponent(InlineEditorComponent);
    context = fixture.componentInstance;
    fixture.detectChanges();
  });
});
```



```
it('should change value of the component', () => {
  let input = fixture.nativeElement.querySelector("input");
  input.value = "Username";
  dispatchEvent(input, 'input');
  fixture.detectChanges();

  fixture.whenStable().then(() => {
    //this button dispatch event for save the text in component.value
    fixture.nativeElement.querySelectorAll('button')[0].click();
    expect(context.value).toBe("Username");
  });
});
```

Lire Tester ngModel en ligne: <https://riptutorial.com/fr/angular2/topic/8693/tester-ngmodel>

Chapitre 67: Titre de la page

Introduction

Comment pouvez-vous changer le titre de la page

Syntaxe

- setTitle(newTitle: string): void;
- getTitle(): string;

Exemples

changer le titre de la page

1. Nous devons d'abord fournir un service de titres.
2. Utiliser setTitle

```
import {Title} from "@angular/platform-browser";
@Component({
  selector: 'app',
  templateUrl: './app.component.html',
  providers : [Title]
})

export class AppComponent implements {
  constructor( private title: Title) {
    this.title.setTitle('page title changed');
  }
}
```

Lire Titre de la page en ligne: <https://riptutorial.com/fr/angular2/topic/8954/titre-de-la-page>

Chapitre 68: Utiliser des composants Web natifs dans Angular 2

Remarques

Lorsque vous utilisez un composant Web dans votre modèle Angular 2, angular essaiera de trouver un composant avec un sélecteur correspondant à la balise personnalisée du composant Web.

La solution consiste à importer un "schéma d'éléments personnalisés" dans le module qui contient le composant. Cela rendra angulaire toute balise personnalisée qui ne correspond à aucun sélecteur de composant ng.

Exemples

Inclure le schéma des éléments personnalisés dans votre module

```
import { NgModule, CUSTOM_ELEMENTS_SCHEMA } from '@angular/core';
import { CommonModule } from '@angular/common';
import { AboutComponent } from './about.component';

@NgModule({
  imports: [ CommonModule ],
  declarations: [ AboutComponent ],
  exports: [ AboutComponent ],
  schemas: [ CUSTOM_ELEMENTS_SCHEMA ]
})

export class AboutModule { }
```

Utilisez votre composant web dans un modèle

```
import { Component } from '@angular/core';

@Component({
  selector: 'myapp-about',
  template: `<my-webcomponent></my-webcomponent>`
})
export class AboutComponent { }
```

Lire Utiliser des composants Web natifs dans Angular 2 en ligne:

<https://riptutorial.com/fr/angular2/topic/7414/utiliser-des-composants-web-natifs-dans-angular-2>

Chapitre 69: Utiliser des librairies tierces comme jQuery dans Angular 2

Introduction

Lors de la création d'applications à l'aide d'Angular 2.x, il est parfois nécessaire d'utiliser des bibliothèques tierces telles que jQuery, Google Analytics, les API JavaScript de Chat Integration, etc.

Exemples

Configuration en utilisant des angles angulaires

MNP

Si une bibliothèque externe comme `jQuery` est installée avec NPM

```
npm install --save jquery
```

Ajouter un chemin de script dans votre `angular-cli.json`

```
"scripts": [  
  "../node_modules/jquery/dist/jquery.js"  
]
```

Dossier Actifs

Vous pouvez également enregistrer le fichier de bibliothèque dans votre répertoire `assets/js` et l'inclure dans `angular-cli.json`

```
"scripts": [  
  "assets/js/jquery.js"  
]
```

Remarque

Enregistrez votre bibliothèque principale `jquery` et leurs dépendances comme `jquery-cycle-plugin` dans le répertoire `assets` et ajoutez-les tous les deux dans `angular-cli.json`, assurez-vous que l'ordre est respecté pour les dépendances.

Utiliser jQuery dans les composants Angular 2.x

Pour utiliser `jquery` dans vos composants Angular 2.x, déclarez une variable globale au sommet

Si vous utilisez `$` pour jQuery

```
declare var $: any;
```

Si vous utilisez `jQuery` pour jQuery

```
declare var jQuery: any
```

Cela permettra d'utiliser `$` ou `jQuery` dans votre composant Angular 2.x.

Lire Utiliser des librairies tierces comme jQuery dans Angular 2 en ligne:

<https://riptutorial.com/fr/angular2/topic/9285/utiliser-des-librairies-tierces-comme-jquery-dans-angular-2>

Chapitre 70: Zone.js

Exemples

Faire référence à NgZone

`NgZone` référence `NgZone` peut être injectée via l'injection de dépendance (DI).

my.component.ts

```
import { Component, NgOnInit, NgZone } from '@angular/core';

@Component({...})
export class Mycomponent implements NgOnInit {
  constructor(private _ngZone: NgZone) { }
  ngOnInit() {
    this._ngZone.runOutsideAngular(() => {
      // Do something outside Angular so it won't get noticed
    });
  }
}
```

Utiliser NgZone pour effectuer plusieurs requêtes HTTP avant d'afficher les données

`runOutsideAngular` peut être utilisé pour exécuter du code en dehors de Angular 2 afin qu'il ne déclenche pas inutilement la détection de modifications. Cela peut être utilisé par exemple pour exécuter plusieurs requêtes HTTP pour obtenir toutes les données avant de les rendre. Pour exécuter le code à nouveau à l'intérieur angulaire 2, `run` la méthode de `NgZone` peut être utilisé.

my.component.ts

```
import { Component, OnInit, NgZone } from '@angular/core';
import { Http } from '@angular/http';

@Component({...})
export class Mycomponent implements OnInit {
  private data: any[];
  constructor(private http: Http, private _ngZone: NgZone) { }
  ngOnInit() {
    this._ngZone.runOutsideAngular(() => {
      this.http.get('resource1').subscribe((data1:any) => {
        // First response came back, so its data can be used in consecutive request
        this.http.get(`resource2?id=${data1['id']}`).subscribe((data2:any) => {
          this.http.get(`resource3?id1=${data1['id']}&id2=${data2}`).subscribe((data3:any) =>
          {
            this._ngZone.run(() => {
              this.data = [data1, data2, data3];
            });
          });
        });
      });
    });
  }
}
```

```
    });  
  }  
}
```

Lire Zone.js en ligne: <https://riptutorial.com/fr/angular2/topic/4184/zone-js>

Crédits

S. No	Chapitres	Contributeurs
1	Démarrer avec Angular 2	acdcjunior , Alexander Ciesielski , beagleknight , Bean0341 , Bhoomi Bhalani , BogdanC , briantylor , cDecker32 , Christopher Moore , Community , daniellmb , drbishop , echonax , Ekin Yücel , elliott-j , etayluz , ettanany , Everettss , H. Pauwelyn , Harry , He11ion , Janco Boscan , Jim , Kaspars Bergs , Logan H , Madhu Ranjan , michaelbahr , Michal Pietraszko , Mihai , nick , Nicolas Irisarri , Peter , QoP , rickysullivan , Shahzad , spike , theblindprophet , user6939352
2	Ajout dynamique de composants à l'aide de <code>ViewContainerRef.createComponent</code>	amansoni211 , daniellmb , Günter Zöchbauer , jupiter24 , Khaled
3	Angulaire - ForLoop	aholtry , Anil Singh , Berseker59 , gerl , Johan Van de Merwe , ob1 , Pujan Srivastava , Stephen Leppik , Yoav Schniederman
4	Angulaire 2 - rapporteur	Yoav Schniederman
5	Angulaire-cli	BogdanC , Yoav Schniederman
6	Angular 2 Détection de changement et déclenchement manuel	Yoav Schniederman
7	Angular 2 formes de données pilotées	MatWaligora , ThunderRoid
8	Angular2 Animations	Yoav Schniederman
9	Angular2 CanActivate	Companjo , Yoav Schniederman
10	Angular2 Databinding	Yoav Schniederman
11	Angular2 en utilisant webpack	luukgruijs
12	Angular2 Entrée () sortie ()	Kaloyan , Yoav Schniederman
13	Angular2 fournit des données externes à App avant le bootstrap	Ajey
14	Angular2 Validations personnalisées	Arnold Wiersma , Norsk , Yoav Schniederman

15	Animation	Gaurav Mukherjee , Nate May
16	API Web Angular2 In Memory	Jaime Still
17	Baril	TechJhola
18	Bootstrap Module vide en angulaire 2	AryanJ-NYC , autoboxer , Berseker59 , Eric Jimenez , Krishan , Sanket , snorkpete
19	Chargement paresseux d'un module	Batajus , M4R1KU , Shannon Young , Syam Pradeep
20	CommandePar Pipe	Yoav Schniederma
21	Comment utiliser ngfor	Jorge , Yoav Schniederma
22	Comment utiliser ngif	Amit kumar , ob1 , ppovoski , samAlvin
23	Compilation AOT avec Angular 2	Anil Singh , Eric Jimenez , Harry , Robin Dijkhof
24	Composants	BrunoLM
25	Conception de matériau angulaire	Ketan Akbari , Shailesh Ladumor
26	Configuration de l'application ASP.net Core pour qu'elle fonctionne avec Angular 2 et TypeScript	Oleksii Aza , Sam
27	couverture de test angulaire-cli	ahmadalibaloch
28	Créer un paquet Angular 2+ NPM	BogdanC , Janco Boscan , vinagreti
29	Créer une bibliothèque Angular npm	Maciej Treder
30	Crochets de cycle de vie	Alexandre Junges , daniellmb , Deen John , muetzerich , Sbats , theblindprophet
31	CRUD dans Angular2 avec API Restful	bleakgadfly , Sefa
32	Débogage de l'application typographique Angular2 à l'aide du code Visual Studio	PSabuwala
33	Détection des événements de redimensionnement	Eric Jimenez
34	Directives	acdcjunior , Andrei Zhytkevich , borislemke , BrunoLM , daniellmb , Everettss , lexith , Stian Standahl , theblindprophet

35	Directives d'attribut affectant la valeur des propriétés sur le noeud hôte à l'aide du décorateur @HostBinding.	Max Karpovets
36	Directives et composants: @Input @Output	acdcjunior , dafyddPrys , Everettss , Joel Almeida , lexith , muetzerich , theblindprophet , ThomasP1988
37	Directives et services généralement intégrés	Jim , Sanket
38	Dropzone dans Angular2	Ketan Akbari
39	Exemple pour des routes telles que / route / subroute pour les URL statiques	Yoav Schniederman
40	Exemples de composants avancés	borislemke , smnbbvr
41	Http Interceptor	Everettss , Mihai , Mike Kovetsky , Nilz11 , Paul Marshall , peeskilllet , theblindprophet
42	Ignorer la désinfection pour les valeurs de confiance	Scrambo
43	Installer des plugins tiers avec angular-cli@1.0.0-beta.10	Alex Morales , Daredzik , filoxo , Kaspars Bergs , pd farhad
44	Interactions entre composants	H. Pauwelyn , Janco Boscan , LLL , Sefa
45	Le routage	aholtry , Apmis , AryanJ-NYC , borislemke , camwhite , Kaspars Bergs , LordTribual , Sachin S , theblindprophet
46	Mise à jour angulaire de 2 formulaires	Amit kumar , Anil Singh , Christopher Taylor , Highmastdon , Johan Van de Merwe , K3v1n , Manmeet Gill , mayur , Norsk , Sachin S , victoroniibukun , vijaykumar , Yoav Schniederman
47	Mise à jour des typings	kEpEx
48	Mise à niveau de la force brutale	Jim , Treveshan Naidoo
49	Mocking @ ngrx / Store	BrianRT , Hatem , Jim , Lucas , Yoav Schniederman
50	Modèles	Max Karpovets
51	Modules	BrunoLM

52	Modules de fonctionnalités	AryanJ-NYC , gsc
53	ngrx	Maxime
54	npx-bootstrap personnalisé datepicker + entrée	Yoav Schniederman
55	Optimisation du rendu à l'aide de ChangeDetectionStrategy	daniellmb , Eric Jimenez , Everettss
56	Ouvrier	Roberto Fernandez
57	Pipes	acdcjunior , Boris , borislemke , BrunoLM , Christopher Taylor , Chybie , daniellmb , Daredzik , elliott-j , Everettss , Fredrik Lundin , Jarod Moser , Jeff Cross , Jim , Kaspars Bergs , Leon Adler , Lexi , LordTribual , michaelbahr , Philipp Kief , theblindprophet
58	redux angulaire	Yoav Schniederman
59	Routage (3.0.0+)	Ai_boy , Alexis Le Gal , Everettss , Gerard Simpson , Kaspars Bergs , mast3rd3mon , meorfi , rivanov , SlashTag , smnbbv , theblindprophet , ThomasP1988 , Trent
60	Service EventEmitter	Abrar Jahin
61	Services et injection de dépendance	BrunoLM , Eduardo Carísio , Kaspars Bergs , Matrim , Roope Hakulinen , Syam Pradeep , theblindprophet
62	Sujets et observables angulaires RXJS avec requêtes API	daniellmb , Maciej Treder , Ronald Zarīts , Sam Storie , Sébastien Temprado , willydee
63	Test d'une application angulaire 2	Arun Redhu , michaelbahr , nick , Reza , Rumit Parakhiya
64	test unitaire	Yoav Schniederman
65	Tester ngModel	jesussegado
66	Titre de la page	Yoav Schniederman
67	Utiliser des composants Web natifs dans Angular 2	ugreen
68	Utiliser des bibliothèques tierces comme jQuery dans Angular 2	Ashok Vishwakarma

