eBook Gratuit

APPRENEZ Angular

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

#angular

Table des matières

À propos

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: angular

It is an unofficial and free Angular 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.

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: Commencer avec Angular

Remarques

Angular (communément appelé " **Angular 2+** " ou " **Angular 2** ") est une infrastructure Web frontale open source basée sur TypeScript, dirigée par Angular Team chez Google et par une communauté d'individus et de sociétés. parties du flux de travail du développeur lors de la création d'applications Web complexes. Angular est une réécriture complète de la même équipe qui a construit AngularJS . [1]

Le framework est constitué de plusieurs bibliothèques , certaines centrales (@ angular / core par exemple) et d'autres optionnelles (@ angular / animations).

Vous écrivez des applications angulaires en composant des *modèles* HTML avec un balisage angularisé, en écrivant *des* classes de *composants* pour gérer ces modèles, en ajoutant une logique d'application dans les *services* et en intégrant des composants et des services dans des *modules* .

Ensuite, vous lancez l'application en *amorçant* le *module racine* . Angular prend le relais en présentant le contenu de votre application dans un navigateur et en répondant aux interactions des utilisateurs en fonction des instructions que vous avez fournies.

La partie la plus fondamentale du développement d'applications angulaires est sans doute les **composants** . Un composant est la combinaison d'un modèle HTML et d'une classe de composants qui contrôle une partie de l'écran. Voici un exemple de composant qui affiche une chaîne simple:

src / app / app.component.ts

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

@Component({
    selector: 'my-app',
    template: `<h1>Hello {{name}}</h1>`
})
export class AppComponent {
    name = 'Angular';
}
```

Chaque composant commence par une fonction de décorateur `@Component` qui prend un objet de métadonnées . L'objet métadonnées décrit comment le modèle HTML et la classe de composants fonctionnent ensemble.

La propriété `selector` indique à Angular d'afficher le composant dans une `<my-app>` dans le fichier *index.html* .

index.html (dans la balise `body`)

```
<my-app>Loading AppComponent content here ...</my-app>
```

La propriété template définit un message dans un en-tête `<h1>` . Le message commence par "Hello" et se termine par `{{name}}` , qui est une expression de liaison par interpolation angulaire. Au moment de l'exécution, Angular remplace `{{name}}` par la valeur de la propriété `name` du composant. La liaison d'interpolation est l'une des nombreuses fonctionnalités angulaires que vous découvrirez dans cette documentation. Dans l'exemple, modifiez la propriété `name` la classe de composants de `'Angular'` à `'World'` et voyez ce qui se passe.

Cet exemple est écrit en **TypeScript** , un sur-ensemble de JavaScript. Angular utilise TypeScript car ses types facilitent la productivité des développeurs avec les outils. En outre, **presque tout le support est pour TypeScript** et l'utilisation de **JavaScript simple** pour écrire votre application sera **difficile** . Écrire du code angulaire en JavaScript est cependant possible; Ce guide explique comment.

Plus d'informations sur l' **architecture** d'Angular peuvent être trouvées **ici**

Versions

Version	Date de sortie
5.0.0-beta.1 (dernière)	2017-07-27
4.3.2	2017-07-26
5.0.0-beta.0	2017-07-19
4.3.1	2017-07-19
4.3.0	2017-07-14
4.2.6	2017-07-08
4.2.5	2017-06-09
4.2.4	2017-06-21
4.2.3	2017-06-16
4.2.2	2017-06-12
4.2.1	2017-06-09
4.2.0	2017-06-08
4.2.0-rc.2	2017-06-01
4.2.0-rc.1	2017-05-26

Version	Date de sortie
4.2.0-rc.0	2017-05-19
4.1.3	2017-05-17
4.1.2	2017-05-10
4.1.1	2017-05-04
4.1.0	2017-04-26
4.1.0-rc.0	2017-04-21
4.0.3	2017-04-21
4.0.2	2017-04-11
4.0.1	2017-03-29
4.0.0	2017-03-23
4.0.0-rc.6	2017-03-23
4.0.0-rc.5	2017-03-17
4.0.0-rc.4	2017-03-17
2.4.10	2017-03-17
4.0.0-rc.3	2017-03-10
2.4.9	2017-03-02
4.0.0-rc.2	2017-03-02
4.0.0-rc.1	2017-02-24
2.4.8	2017-02-18
2.4.7	2017-02-09
2.4.6	2017-02-03
2.4.5	2017-01-25
2.4.4	2017-01-19
2.4.3	2017-01-11
2.4.2	2017-01-06

Version	Date de sortie
2.4.1	2016-12-21
2.4.0	2016-12-20
2.3.1	2016-12-15
2.3.0	2016-12-07
2.3.0-rc.0	2016-11-30
2.2.4	2016-11-30
2.2.3	2016-11-23
2.2.2	2016-11-22
2.2.1	2016-11-17
2.2.0	2016-11-14
2.2.0-rc.0	2016-11-02
2.1.2	2016-10-27
2.1.1	2016-10-20
2.1.0	2016-10-12
2.1.0-rc.0	2016-10-05
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
2.0.0-rc.3	2016-06-21
2.0.0-rc.2	2016-06-15
2.0.0-rc.1	2016-05-03

Version	Date de sortie
2.0.0-rc.0	2016-05-02

Examples

Installation d'angulaire à l'aide d'angles

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

Conditions préalables:

- Node.js 6.9.0 ou supérieur.
- npm v3 ou plus ou fil .
- Typings v1 ou plus.

Ouvrez un terminal et exécutez les commandes une par une:

`npm install -g typings` ou `yarn global add typings`

`npm install -g @angular/cli` ou `yarn global add @angular/cli`

La première commande installe globalement la bibliothèque de typages (et ajoute l'exécutable `typings` à PATH). Le second 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 ça, vous avez maintenant un exemple de projet simple réalisé avec Angular. Vous pouvez maintenant naviguer vers le lien affiché dans le terminal et voir ce qu'il exécute.

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
# You can add --flat if you don't want to create new folder for a component
ng g component my-generated-component --flat
# You can add --spec false if you don't want a test file to be generated (my-generated-
component.spec.ts)
ng g component my-generated-component --spec false
```

Il existe plusieurs types d’échafaudages possibles:

Type d'échafaudage	Usage
Module	`ng g module my-new-module`
Composant	`ng g component my-new-component`
Directif	`ng g directive my-new-directive`

Type d'échafaudage	Usage
Tuyau	`ng g pipe my-new-pipe`
Un service	`ng g service my-new-service`
Classe	`ng g class my-new-class`
Interface	`ng g interface my-new-interface`
Enum	`ng g enum my-new-enum`

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 et que vous souhaitez l'installer sur un serveur Web tel qu'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 fournit des tests unitaires intégrés, et chaque élément créé par angular-cli génère un test élémentaire de base, qui peut être utilisé. 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

Programme angulaire "Hello World"

Conditions préalables:

Mise en place de l'environnement de développement

Avant de commencer, nous devons configurer notre environnement.

- Installez Node.js et npm s'ils ne sont pas déjà sur votre machine.

  Vérifiez que vous exécutez au moins le noeud 6.9.x et npm 3.xx en exécutant node -v et npm -v dans une fenêtre de terminal / console. Les anciennes versions génèrent des erreurs, mais les nouvelles versions sont correctes.

- Installez la CLI angulaire globalement en utilisant `npm install -g @angular/cli` .

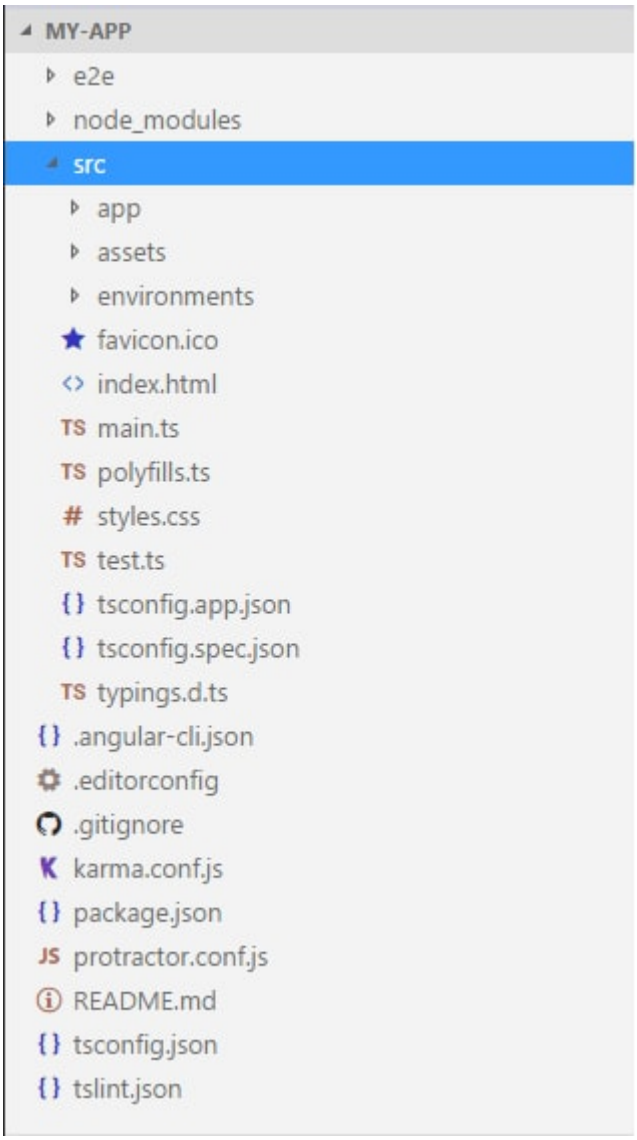
Etape 1: Créer un nouveau projet

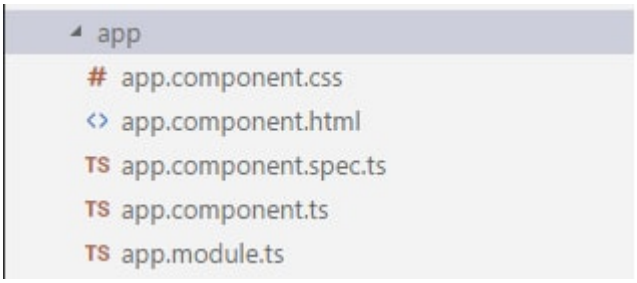
Ouvrez une fenêtre de terminal (ou une invite de commande Node.js dans Windows).

Nous créons un nouveau projet et une application squelette en utilisant la commande:

```
ng new my-app
```

Ici, le `ng` est pour angulaire. Nous obtenons une structure de fichier comme celle-ci.





Il y a beaucoup de fichiers. Nous n'avons pas à nous soucier de tous maintenant.

Étape 2: servir l'application

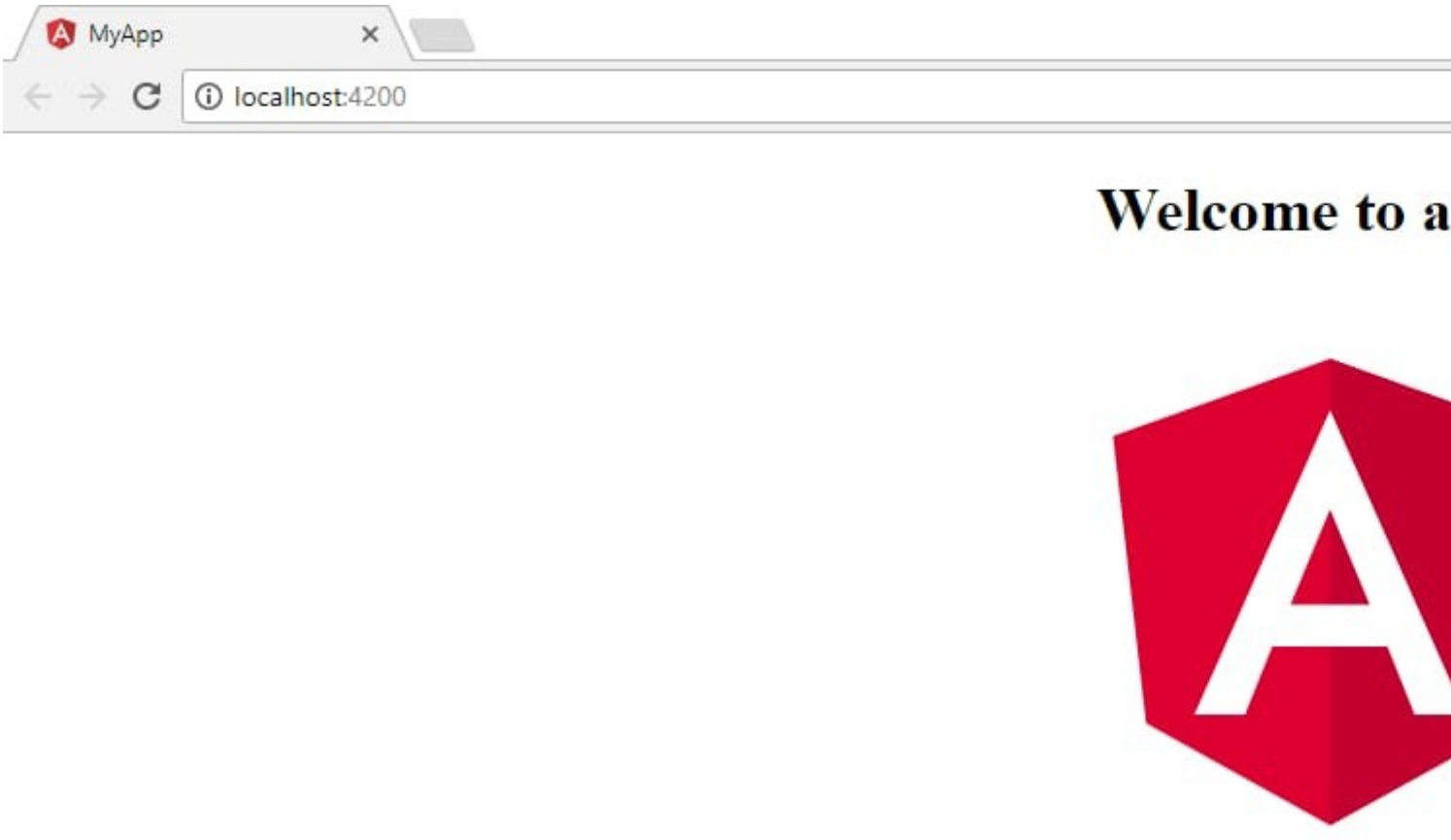
Nous lançons notre application en utilisant la commande suivante:

```
ng serve
```

Nous pouvons utiliser un drapeau `-open` (ou simplement `-o`) qui ouvrira automatiquement notre navigateur sur `http://localhost:4200/`

```
ng serve --open
```

Naviguez dans le navigateur jusqu'à l'adresse `http://localhost:4200/` . Cela ressemble à ceci:



Étape 3: Édition de notre premier composant angulaire

La CLI a créé le composant Angular par défaut pour nous. C'est le composant racine et il s'appelle `app-root` . On peut le trouver dans `./src/app/app.component.ts` .

Ouvrez le fichier de composant et modifiez la propriété title de `Welcome to app!!` à `Hello World` . Le navigateur se recharge automatiquement avec le titre révisé.

Code d'origine: Notez le `title = 'app';`

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

Angular CLI, 20 minutes ago | 1 author (Angular CLI)
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'app';
}
```

Code modifié: la valeur du `title` est modifiée.

```
import { Component } from '@angular/core';
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'Hello World';
}
```

De même, il y a un changement dans `./src/app/app.component.html` .

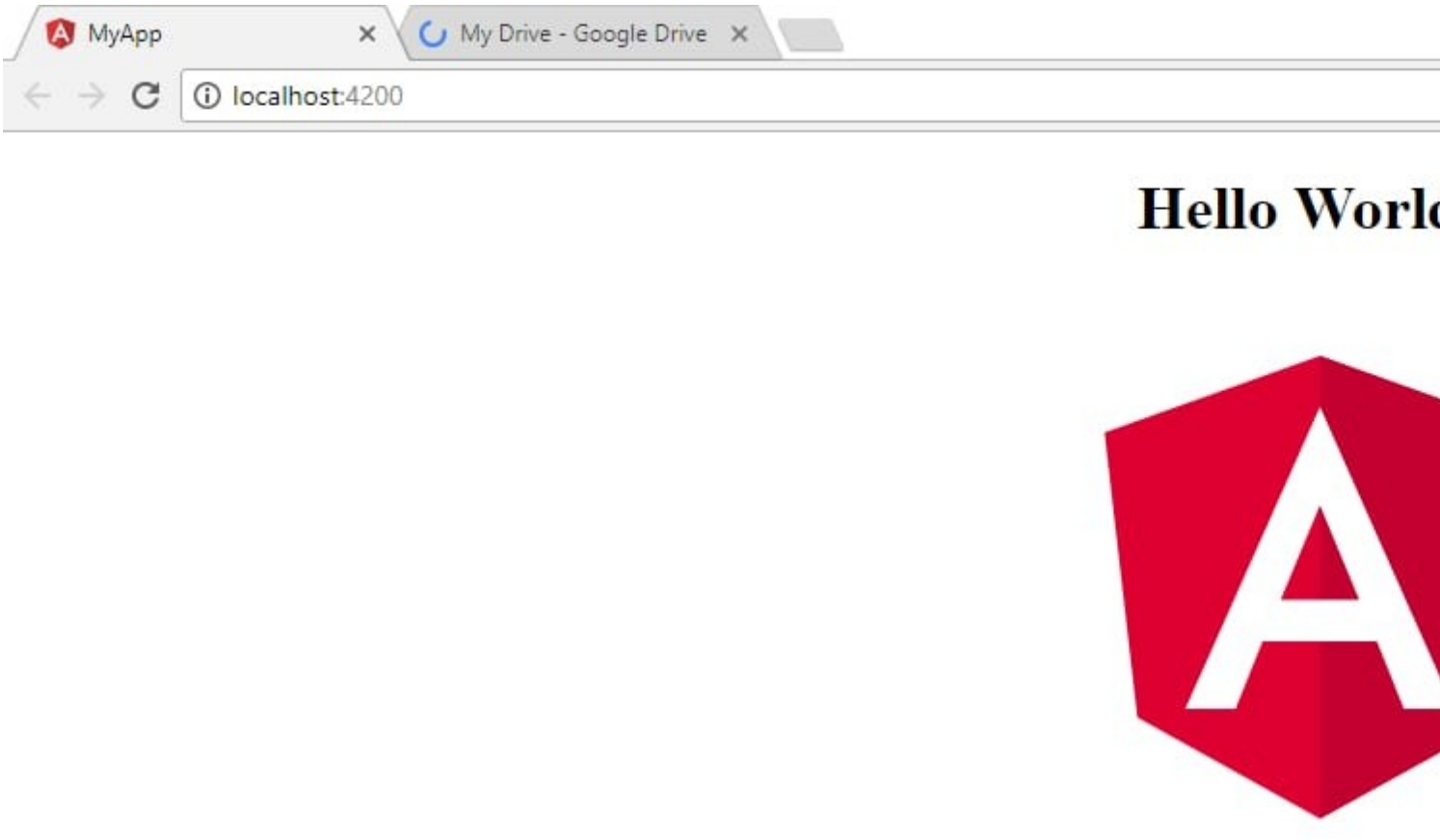
HTML original

```
<div style="text-align:center">
  <h1>
    Welcome to {{title}}!!
  </h1>
  <img width="300" src="
```

HTML modifié

```
<!--The content below is only a placeholder and can be replaced.-->
<div style="text-align:center">
  <h1>
     {{title}}!!
  </h1>
  <img width="300" src="
```

Notez que la valeur du `title` du `./src/app/app.component.ts` sera affichée. Le navigateur se recharge automatiquement lorsque les modifications sont effectuées. Cela ressemble à quelque chose comme ça.



Pour en savoir plus sur le sujet, visitez ce lien ici .

Lire Commencer avec Angular en ligne: https://riptutorial.com/fr/angular/topic/9754/commencer-avec-angular

Chapitre 2: Émetteur d'événement

Examples

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);
    }
}
```

Lire Émetteur d'événement en ligne: https://riptutorial.com/fr/angular/topic/9828/emetteur-d-evenement

Chapitre 3: Formes

Examples

Formes réactives

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 './app.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>
        <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
            };
        }
    }
}
```

Formulaires pilotés par modèle

Template - `signup.component.html`

```
<form #signUpForm="ngForm" (ngSubmit)="onSubmit()">

  <div class="title">
    Sign Up
  </div>

  <div class="input-field">
    <label for="username">username</label>
    <input
      type="text"
      pattern="\w{4,20}"
      name="username"
      required="required"
      [(ngModel)]="signUpRequest.username" />
  </div>

  <div class="input-field">
    <label for="email">email</label>
    <input
      type="email"
      pattern="^\S+@\S+$"
      name="email"
      required="required"
```

```
      [(ngModel)]="signUpRequest.email" />
  </div>

  <div class="input-field">
    <label for="password">password</label>
    <input
      type="password"
      pattern=".{6,30}"
      required="required"
      name="password"
      [(ngModel)]="signUpRequest.password" />
  </div>

  <div class="status">
    {{ status }}
  </div>

  <button [disabled]="!signUpForm.form.valid" type="submit">
    <span>Sign Up</span>
  </button>

</form>
```

Composant - `signup.component.ts`

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

import { SignUpRequest } from './signup-request.model';

@Component({
  selector: 'app-signup',
  templateUrl: './signup.component.html',
  styleUrls: ['./signup.component.css']
})
export class SignupComponent {

  status: string;
  signUpRequest: SignUpRequest;

  constructor() {
    this.signUpRequest = new SignUpRequest();
  }

  onSubmit(value, valid) {
    this.status = `User ${this.signUpRequest.username} has successfully signed up`;
  }

}
```

Modèle - `signup-request.model.ts`

```
export class SignUpRequest {

  constructor(
    public username: string="",
    public email: string="",
```

```
    public password: string=""
  ) {}

}
```

Module App - `app.module.ts`

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

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

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

Composant App - `app.component.html`

```
<app-signup></app-signup>
```

Lire Formes en ligne: https://riptutorial.com/fr/angular/topic/9825/formes

Chapitre 4: Le routage

Examples

Routage avec des enfants

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

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 -

REMARQUE : Assurez-vous d'avoir cette balise:

```
<base href='/'>
```

comme premier enfant sous votre balise head dans votre fichier index.html. Cet élément indique que votre dossier d'application est la racine de l'application. Angular saura alors comment organiser vos liens.

1. Vérifiez si vous pointez sur les dépendances de routage correctes / les plus récentes dans package.json (en utilisant la dernière version d'Angular) et que vous avez déjà effectué une `npm install` -

   ```
   "dependencies": {
       "@angular/router": "^4.2.5"
   }
   ```

2. Définissez l'itinéraire selon sa définition d'interface:

   ```
   interface Route {
     path?: string;
     pathMatch?: string;
     component?: Type<any>;
   }
   ```

3. Dans un fichier de routage (`routes/app.routing.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.

```
import { Routes, RouterModule } from '@angular/router';
import { ModuleWithProviders } from '@angular/core';
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 APP_ROUTES: Routes = [
    { 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_ROUTING: ModuleWithProviders = RouterModule.forRoot(APP_ROUTES);
```

4. Dans votre `app.module.ts` , placez ceci sous `@NgModule([])` sous `imports` :

```
// Alternatively, just import 'APP_ROUTES
import {APP_ROUTING} from '../routes/app.routing.ts';
@NgModule([
    imports: [
        APP_ROUTING
        // Or RouterModule.forRoot(APP_ROUTES)
    ]
])
```

5. Chargez / affichez les composants du routeur en fonction du chemin d'accès accédé. La directive `<router-outlet>` est utilisée pour indiquer angulaire où charger le composant.

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

@Component({
    selector: 'demo-app',
    template: `
        <div>
            <router-outlet></router-outlet>
        </div>
    `
})
export class AppComponent {}
```

6. Liez les autres itinéraires. Par défaut, `RouterOutlet` charge le composant pour lequel le chemin vide est spécifié dans les `Routes` . `RouterLink` directive `RouterLink` est utilisée avec la balise d'ancrage HTML pour charger les composants `RouterLink` aux itinéraires. `RouterLink` génère l'attribut href qui est utilisé pour générer des liens. Par exemple:

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

@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>
        `
})
export class AnotherComponent { }
```

Maintenant, nous sommes bons avec le routage vers des chemins statiques. `RouterLink` prend `RouterLink` charge le chemin dynamique en transmettant des paramètres supplémentaires avec le chemin.

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

@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>
  `
})
export class SecondComponent { }
```

`RouterLink` prend un tableau dans lequel le premier paramètre est le chemin de routage et les éléments suivants concernent les paramètres de routage dynamique.

Lire Le routage en ligne: https://riptutorial.com/fr/angular/topic/9827/le-routage

Chapitre 5: Partage de données entre composants

Introduction

L'objectif de cette rubrique est de créer des exemples simples de plusieurs manières selon lesquelles les données peuvent être partagées entre les composants via la liaison de données et le service partagé.

Remarques

Il y a toujours beaucoup de façons d'accomplir une tâche dans la programmation. N'hésitez pas à modifier les exemples actuels ou à en ajouter quelques-uns.

Examples

Envoi de données du composant parent à l'enfant via un service partagé

service.ts:

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

@Injectable()
export class AppState {

  public mylist = [];

}
```

parent.component.ts:

```
import {Component} from '@angular/core';
import { AppState } from './shared.service';

@Component({
  selector: 'parent-example',
  templateUrl: 'parent.component.html',
})
export class ParentComponent {
  mylistFromParent = [];

  constructor(private appState: AppState){
    this.appState.mylist;
  }

  add() {
    this.appState.mylist.push({"itemName":"Something"});
  }
```

```
}
```

parent.component.html:

```
<p> Parent </p>
  <button (click)="add()">Add</button>
<div>
  <child-component></child-component>
</div>
```

child.component.ts:

```
import {Component, Input } from '@angular/core';
import { AppState } from './shared.service';

@Component({
  selector: 'child-component',
  template: `
    <h3>Child powered by shared service</h3>
        {{mylist | json}}
  `,
})
export class ChildComponent {
  mylist: any;

  constructor(private appState: AppState){
    this.mylist = this.appState.mylist;
  }

}
```

Envoyer des données du composant parent au composant enfant via la liaison de données à l'aide de @Input

parent.component.ts:

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

@Component({
  selector: 'parent-example',
  templateUrl: 'parent.component.html',
})

export class ParentComponent {
  mylistFromParent = [];

  add() {
    this.mylistFromParent.push({"itemName":"Something"});
  }

}
```

parent.component.html:

```
<p> Parent </p>
  <button (click)="add()">Add</button>

<div>
  <child-component [mylistFromParent]="mylistFromParent"></child-component>
</div>
```

child.component.ts:

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

@Component({
  selector: 'child-component',
  template: `
    <h3>Child powered by parent</h3>
        {{mylistFromParent | json}}
  `,
})

export class ChildComponent {
  @Input() mylistFromParent = [];
}
```

Envoi de données d'un enfant à un parent via l'émetteur d'événement @Output

event-emitter.component.ts

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

@Component({
  selector: 'event-emitting-child-component',
  template: `<div *ngFor="let item of data">
              <div (click)="select(item)">
                {{item.id}} = {{ item.name}}
              </div>
             </div>
            `
})

export class EventEmitterChildComponent implements OnInit{

  data;

  @Output()
  selected: EventEmitter<string> = new EventEmitter<string>();

  ngOnInit(){
    this.data = [ { "id": 1, "name": "Guy Fawkes", "rate": 25 },
                  { "id": 2, "name": "Jeremy Corbyn", "rate": 20 },
                  { "id": 3, "name": "Jamie James", "rate": 12 },
                  { "id": 4, "name": "Phillip Wilson", "rate": 13 },
                  { "id": 5, "name": "Andrew Wilson", "rate": 30 },
                  { "id": 6, "name": "Adrian Bowles", "rate": 21 },
                  { "id": 7, "name": "Martha Paul", "rate": 19 },
                  { "id": 8, "name": "Lydia James", "rate": 14 },
                  { "id": 9, "name": "Amy Pond", "rate": 22 },
```

```
                  { "id": 10, "name": "Anthony Wade", "rate": 22 } ]
  }

  select(item) {
      this.selected.emit(item);

  }

}
```

event-receiver.component.ts:

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

@Component({
    selector: 'event-receiver-parent-component',
    template: `<event-emitting-child-component (selected)="itemSelected($event)">
                </event-emitting-child-component>
                <p *ngIf="val">Value selected</p>
                <p style="background: skyblue">{{ val | json}}</p>`
})

export class EventReceiverParentComponent{
  val;

  itemSelected(e){
    this.val = e;
  }
}
```

Envoi de données asynchrones de parent à enfant à l'aide de Observable et Subject

shared.service.ts:

```
import { Injectable }    from '@angular/core';
import { Headers, Http } from '@angular/http';

import 'rxjs/add/operator/toPromise';

import { Observable } from 'rxjs/Observable';
import { Observable } from 'rxjs/Rx';
import {Subject} from 'rxjs/Subject';


@Injectable()
export class AppState {

  private headers = new Headers({'Content-Type': 'application/json'});
  private apiUrl = 'api/data';

  // Observable string source
  private dataStringSource = new Subject<string>();

  // Observable string stream
  dataString$ = this.dataStringSource.asObservable();
```

```
  constructor(private http: Http) { }

  public setData(value) {
    this.dataStringSource.next(value);
  }

  fetchFilterFields() {
    console.log(this.apiUrl);
    return this.http.get(this.apiUrl)
               .delay(2000)
               .toPromise()
               .then(response => response.json().data)
               .catch(this.handleError);
  }

  private handleError(error: any): Promise<any> {
    console.error('An error occurred', error); // for demo purposes only
    return Promise.reject(error.message || error);
  }

}
```

parent.component.ts:

```
import {Component, OnInit} from '@angular/core';
import 'rxjs/add/operator/toPromise';
import { AppState } from './shared.service';

@Component({
  selector: 'parent-component',
  template: `
              <h2> Parent </h2>
              <h4>{{promiseMarker}}</h4>

              <div>
                <child-component></child-component>
              </div>
            `
})
export class ParentComponent implements OnInit {

  promiseMarker = "";

  constructor(private appState: AppState){ }

  ngOnInit(){
    this.getData();
  }

  getData(): void {
    this.appState
        .fetchFilterFields()
        .then(data => {
          // console.log(data)
          this.appState.setData(data);
          this.promiseMarker = "Promise has sent Data!";
        });
  }

}
```

child.component.ts:

```
import {Component, Input } from '@angular/core';
import { AppState } from './shared.service';


@Component({
  selector: 'child-component',
  template: `
    <h3>Child powered by shared service</h3>
        {{fields | json}}
  `,
})
export class ChildComponent {
  fields: any;

  constructor(private appState: AppState){
    // this.mylist = this.appState.get('mylist');

    this.appState.dataString$.subscribe(
      data => {
        // console.log("Subs to child" + data);
        this.fields = data;
      });

  }

}
```

Lire Partage de données entre composants en ligne:
https://riptutorial.com/fr/angular/topic/10836/partage-de-donnees-entre-composants

Chapitre 6: Pipes

Introduction

Les tuyaux sont très similaires aux filtres dans AngularJS en ce sens qu'ils permettent de transformer les données dans un format spécifié `|` est utilisé pour appliquer des tuyaux dans Angular.

Examples

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 { }
```

Plusieurs pipes personnalisées

Avoir des tuyaux différents est un cas très courant, où chaque tuyau fait une chose différente. L'ajout de chaque canal à chaque composant peut devenir un code répétitif.

Il est possible de regrouper tous les canaux fréquemment utilisés dans un `Module` et d'importer ce nouveau module dans n'importe quel composant ayant besoin des tubes.

lignes de rupture.ts

```
import { Pipe } from '@angular/core';
/**
 * pipe to convert the \r\n into <br />
 */
@Pipe({ name: 'br' })
export class BreakLine {
    transform(value: string): string {
        return value == undefined ? value :
             value.replace(new RegExp('\r\n', 'g'), '<br />')
              .replace(new RegExp('\n', 'g'), '<br />');
    }
}
```

majuscules.ts

```
import { Pipe } from '@angular/core';
/**
 * pipe to uppercase a string
 */
@Pipe({ name: 'upper' })
export class Uppercase{
    transform(value: string): string {
        return value == undefined ? value : value.toUpperCase( );
    }
}
```

pipes.module.ts

```
import { NgModule } from '@angular/core';
import { BreakLine } from './breakLine';
import { Uppercase} from './uppercase';

@NgModule({
    declarations: [
        BreakLine,
        Uppercase
    ],
```

```
    imports: [

    ],
    exports: [
        BreakLine,
        Uppercase
    ]
    ,
})
export class PipesModule {}
```

my.component.ts

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

@Component({
  selector: 'my-component',
  template: `{{ value | upper | br}}`
})
export class MyComponent {

    public value: string;

}
```

my.module.ts

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

import { MyComponent } from './my.component';
import { PipesModule} from './pipes.module';

@NgModule({
  imports: [
    BrowserModule,
    PipesModule,
  ],
  declarations: [
    MyComponent,
  ],
})
```

Lire Pipes en ligne: https://riptutorial.com/fr/angular/topic/9824/pipes

Chapitre 7: Pour boucle

Examples

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.

Lire Pour boucle en ligne: https://riptutorial.com/fr/angular/topic/9826/pour-boucle

Chapitre 8: RXJS et observables

Examples

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` abord `posts tags` .

```
loadData() : void {
    Observable.forkJoin(
        this.blogApi.getPosts(),
        this.blogApi.getTags()
    ).subscribe((([posts, tags]: [Post[], Tag[]]) => {
        this.posts = posts;
        this.tags = tags;
    }));
}
```

Demande de base

L'exemple suivant illustre une simple requête HTTP GET. `http.get()` renvoie un objet `Observable` auquel la méthode est `subscribe` . Celui-ci ajoute les données renvoyées au tableau des `posts` .

```
var posts = []

getPosts(http: Http): {
    this.http.get(`https://jsonplaceholder.typicode.com/posts`)
        .subscribe(response => {
            posts.push(response.json());
        });
}
```

Lire RXJS et observables en ligne: https://riptutorial.com/fr/angular/topic/9829/rxjs-et-observables

Crédits

S. No	Chapitres	Contributeurs
1	Commencer avec Angular	aholtry, Anup Kumar Gupta, BogdanC, Community, daddycool, Edric, Fahad Nisar, Faisal, Hendrik Brummermann, Philipp Kief, Tom
2	Émetteur d'événement	aholtry
3	Formes	aholtry, Saka7
4	Le routage	0mpurdy, aholtry, Edric
5	Partage de données entre composants	0mpurdy, BogdanC, JGFMK, Nehal
6	Pipes	aholtry, Amr ElAdawy
7	Pour boucle	aholtry
8	RXJS et observables	aholtry