Бесплатная электронная книга

УЧУСЬ

Angular

Free unaffiliated eBook created from
Stack Overflow contributors.

#angular

Около

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

глава 1: Начало работы с угловым

замечания

Угловая (обычно называемая « **Угловая 2+** » или « **Угловая 2** ») является основанной на TypeScript интерфейсом веб-интерфейса с открытым исходным кодом под управлением Angular Team в Google и сообществом частных лиц и корпораций для рассмотрения всех части рабочего процесса разработчика при создании сложных веб-приложений. Угловая - это полная перепись от той же команды, которая построила AngularJS . [1]

Структура состоит из нескольких библиотек , некоторые из которых являются ядром (например, угловым / ядром) и некоторыми необязательными (@ angular / animations).

Вы пишете угловые приложения, создавая HTML- *шаблоны* с помощью Angularized markup, пишите классы *компонентов* для управления этими шаблонами, добавляя логику приложения в *сервисах* , а также компоненты и услуги бокса в *модулях* .

Затем вы запускаете приложение, *загружая корневой модуль* . Угловая передача, представление содержимого приложения в браузере и реагирование на взаимодействие пользователя в соответствии с инструкциями, которые вы предоставили.

Возможно, наиболее важной частью разработки приложений с угловыми углами являются **компоненты** . Компонент представляет собой комбинацию HTML-шаблона и класса компонентов, который контролирует часть экрана. Ниже приведен пример компонента, который отображает простую строку:

SRC / приложение / app.component.ts

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

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

Каждый компонент начинается с `@Component` декоратора `@Component` которая принимает объект метаданных . Объект metadata описывает, как HTML-шаблон и класс компонента работают вместе.

Свойство `selector` указывает Angular на отображение компонента внутри пользовательского `<my-app>` в файле *index.html* .

index.html (внутри тега `body`)

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

Свойство template определяет сообщение внутри заголовка `<h1>` . Сообщение начинается с «Hello» и заканчивается на `{{name}}` , который является выражением привязки угловой интерполяции . Во время выполнения Angular заменяет `{{name}}` на значение свойства `name` компонента. Связывание с интерполяцией является одной из многих функций Angular, которые вы найдете в этой документации. В этом примере измените свойство `name` класса компонента на `'Angular'` на `'World'` и посмотрите, что произойдет.

Этот пример написан в **TypeScript** , надмножестве JavaScript. Angular использует TypeScript, потому что его типы упрощают поддержку производительности разработчика с помощью инструментария. Кроме того, **почти вся поддержка для TypeScript** и поэтому использование **простого JavaScript** для написания вашего приложения будет **затруднительным** . Написание Углового кода в JavaScript возможно, однако; это руководство объясняет, как это сделать.

Более подробную информацию об **архитектуре** Angular можно найти **здесь**

Версии

Версия	Дата выхода
5.0.0-beta.1 (последний)	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-PK-2	2017-06-01

Версия	Дата выхода
4.2.0-RC.1	2017-05-26
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-PK-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

Версия	Дата выхода
2.4.2	2017-01-06
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-PK-2	2016-06-15

Версия	Дата выхода
2.0.0-RC.1	2016-05-03
2.0.0-rc.0	2016-05-02

Examples

Установка углового с использованием углового кли

Этот пример представляет собой быструю настройку Angular и как создать быстрый пример проекта.

Предпосылки:

- Node.js 6.9.0 или выше.
- npm v3 или больше или пряжа .
- Типики v1 или выше.

Откройте терминал и выполните команды один за другим:

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

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

Первая команда устанавливает библиотеку типизации глобально (и добавляет `typings` исполняемой в PATH). Второй устанавливает @ **angular / cli** глобально, добавляя исполняемый файл `ng` к PATH.

Чтобы настроить новый проект

Перейдите с помощью терминала в папку, в которой вы хотите настроить новый проект.

Запустите команды:

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

Вот и все, у вас теперь есть простой пример, выполненный с помощью Angular. Теперь вы можете перейти к ссылке, отображаемой в терминале, и посмотреть, что она работает.

Чтобы добавить к существующему

проекту

Перейдите в корень вашего текущего проекта.

Выполните команду:

```
ng init
```

Это добавит необходимые строительные леса в ваш проект. Файлы будут созданы в текущем каталоге, поэтому обязательно запустите его в пустой директории.

Выполнение проекта локально

Чтобы видеть и взаимодействовать с вашим приложением во время работы в браузере, вы должны запустить локальный сервер разработки, в котором размещаются файлы для вашего проекта.

```
ng serve
```

Если сервер успешно запущен, он должен отображать адрес, на котором работает сервер. Обычно это:

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

Исходя из этого, этот локальный сервер разработки подключается с помощью Hot Module Reloading, поэтому любые изменения в html, машинописном тексте или css приведут к тому, что браузер будет автоматически перезагружен (но может быть отключен по желанию).

Создание компонентов, директив, труб и услуг

Команда `ng generate <scaffold-type> <name>` (или просто `ng g <scaffold-type> <name>`) позволяет автоматически генерировать угловые компоненты:

```
# 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
```

Существует несколько возможных типов лесов, которые могут генерировать угловые cli:

Тип лесов	использование
модуль	`ng g module my-new-module`
Составная часть	`ng g component my-new-component`
директива	`ng g directive my-new-directive`
труба	`ng g pipe my-new-pipe`
обслуживание	`ng g service my-new-service`
Учебный класс	`ng g class my-new-class`
Интерфейс	`ng g interface my-new-interface`
Enum	`ng g enum my-new-enum`

Вы также можете заменить имя типа своей первой буквой. Например:

`ng gm my-new-module` для создания нового модуля или `ng gc my-new-component` для создания компонента.

Строительство / Пакетирование

Когда вы все закончите создание своего приложения с угловым веб-сайтом, и вы хотите установить его на веб-сервере, таком как Apache Tomcat, все, что вам нужно сделать, это запустить команду сборки либо с установленным флагом, либо без него. Производство будет минимизировать код и оптимизировать производственные параметры.

```
ng build
```

или же

```
ng build --prod
```

Затем зайдите в корневой каталог проектов для папки `/dist` , которая содержит сборку.

Если вы хотите получить преимущества небольшого пакета продуктов, вы также можете использовать компиляцию шаблона Ahead-of-Time, которая удаляет компилятор шаблона из окончательной сборки:

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

Тестирование устройства

Угловое обеспечивает встроенное модульное тестирование, и каждый элемент, созданный угловым кли, генерирует базовый единичный тест, который может быть израсходован. Модульные тесты записываются с использованием жасмина и выполняются через Карму. Чтобы начать тестирование, выполните следующую команду:

```
ng test
```

Эта команда выполнит все тесты в проекте и будет повторно выполнять их каждый раз при изменении исходного файла, будь то тест или код из приложения.

Для получения дополнительной информации также посетите страницу angular-cli github

Угловая программа «Hello World»

Предпосылки:

Настройка среды разработки

Прежде чем мы начнем, мы должны настроить нашу среду.

- Установите Node.js и npm, если они еще не установлены на вашем компьютере.

  Убедитесь, что вы используете хотя бы узел 6.9.x и npm 3.xx, запустив узел -v и npm -v в окне терминала / консоли. Более старые версии вызывают ошибки, но более новые версии - это хорошо.

- Установите Угловую CLI глобально, используя `npm install -g @angular/cli` .

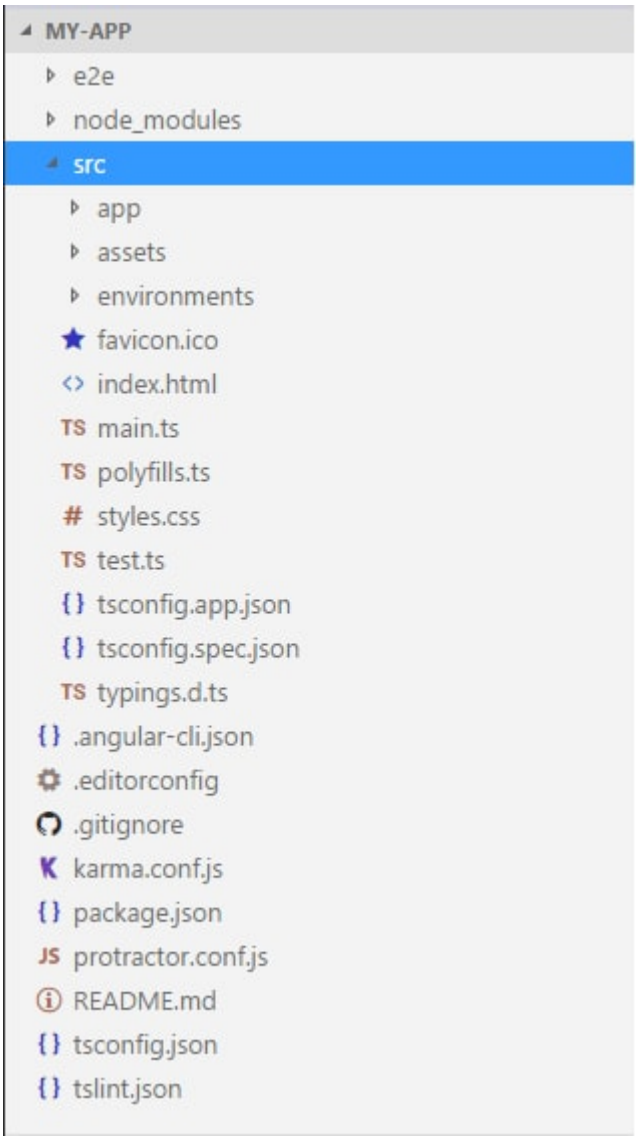
Шаг 1. Создание нового проекта.

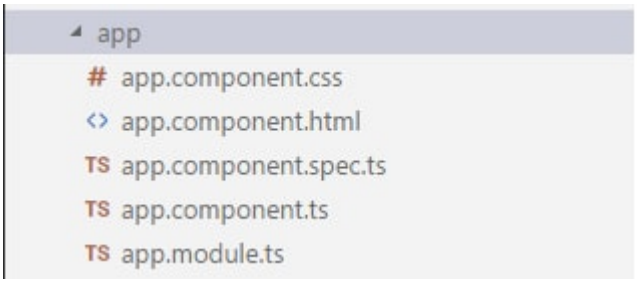
Откройте окно терминала (или командную строку Node.js в окнах).

Мы создаем новое приложение проекта и скелета, используя команду:

```
ng new my-app
```

Здесь `ng` для углового. Мы получаем файловую структуру примерно так.





Есть много файлов. Теперь нам не нужно беспокоиться обо всех них.

Шаг 2: Обслуживание приложения

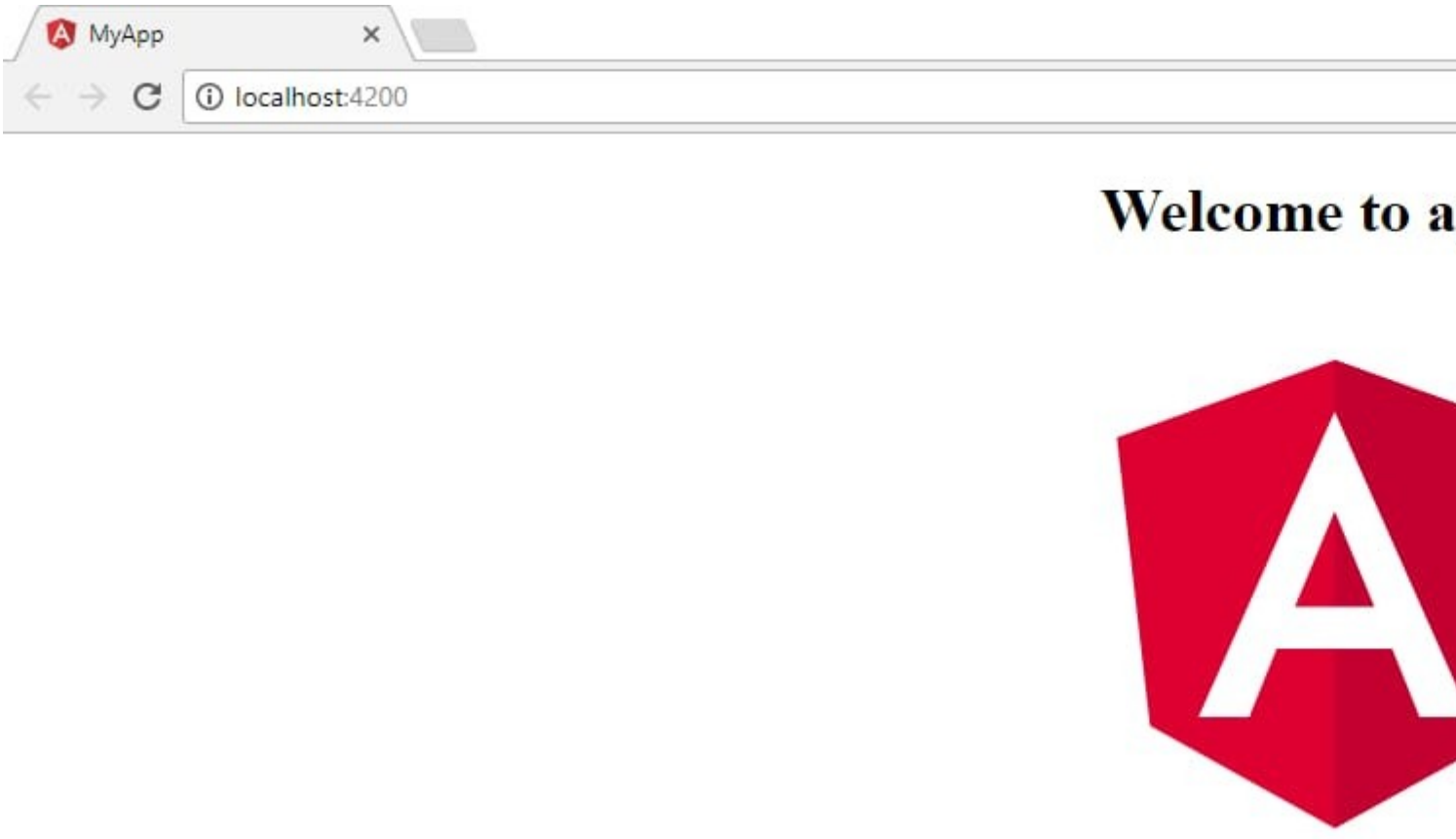
Мы запускаем наше приложение, используя следующую команду:

```
ng serve
```

Мы можем использовать флаг `-open` (или просто `-o`), который автоматически откроет наш браузер на `http://localhost:4200/`

```
ng serve --open
```

Перейдите в браузер по адресу `http://localhost:4200/` . Это выглядит примерно так:



Шаг 3: Редактирование нашего первого углового компонента

CLI создал для нас Угловой компонент по умолчанию. Это корневой компонент, и он называется `app-root` . Его можно найти в `./src/app/app.component.ts` .

Откройте файл компонента и измените свойство title из `Welcome to app!!` к `Hello World` . Браузер автоматически перезагружается с измененным заголовком.

Исходный код: Обратите внимание на `title = 'app';`

```ts
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';
}
```

Измененный код: изменяется значение `title` .

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

Аналогичным образом происходит изменение `./src/app/app.component.html` .

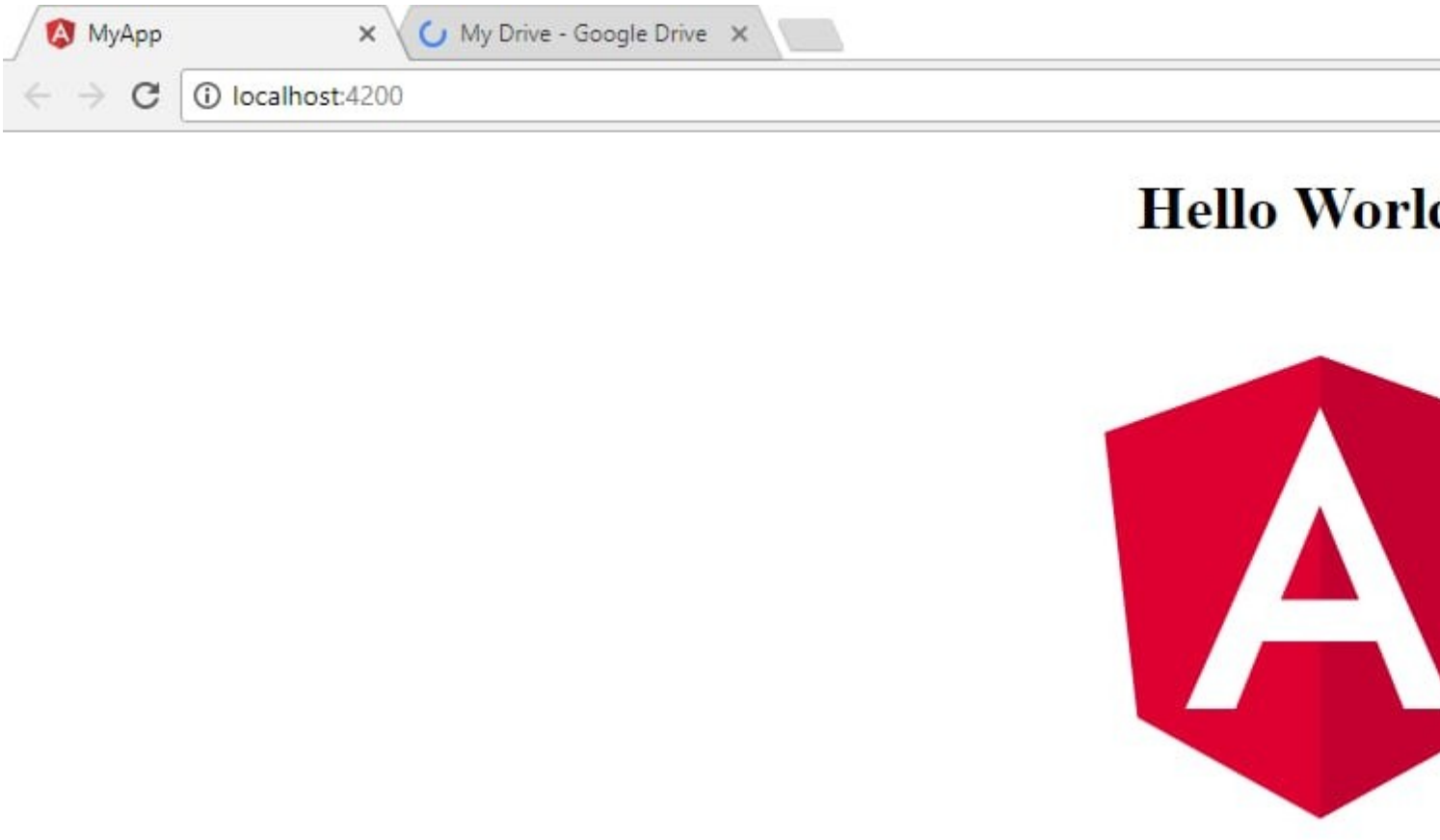
Оригинальный HTML

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

Измененный HTML

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

Обратите внимание, что будет отображаться значение `title` из `./src/app/app.component.ts` . Браузер автоматически перезагружается при выполнении изменений. Это выглядит примерно так.



Чтобы узнать больше по этой теме, перейдите по этой ссылке здесь .

Прочитайте Начало работы с угловым онлайн: https://riptutorial.com/ru/angular/topic/9754/начало-работы-с-угловым

глава 2: RXJS и наблюдаемые

Examples

Подождите несколько запросов

Один из распространенных сценариев - дождаться завершения ряда запросов до продолжения. Это может быть выполнено с использованием метода `forkJoin` .

В следующем примере `forkJoin` используется для вызова двух методов, возвращающих `Observables` . Обратный вызов, указанный в методе `.subscribe` будет вызываться, когда оба Observables завершены. Параметры, предоставленные `.subscribe` соответствуют порядку, указанному в вызове `.forkJoin` . В этом случае сначала `posts` `tags` .

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

Основной запрос

Следующий пример демонстрирует простой HTTP-запрос GET. `http.get()` возвращает `Observable` , который имеет метод `subscribe` . Это добавляет возвращаемые данные в массив `posts` .

```
var posts = []

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

Прочитайте RXJS и наблюдаемые онлайн: https://riptutorial.com/ru/angular/topic/9829/rxjs-и-наблюдаемые

глава 3: Для цикла

Examples

NgFor - разметка для цикла

Директива **NgFor** создает экземпляр шаблона один раз для каждого элемента из итерабельного. Контекст для каждого экземпляра-шаблона наследуется от внешнего контекста с заданной переменной цикла, установленной для текущего элемента из итерабельного.

Чтобы настроить алгоритм отслеживания по умолчанию, NgFor поддерживает **функцию trackBy** . **trackBy** принимает функцию, которая имеет два аргумента: index и item. Если **указано trackBy** , угловые дорожки изменяются по возвращаемому значению функции.

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

Дополнительные параметры : NgFor предоставляет несколько экспортированных значений, которые могут быть сглажены для локальных переменных:

- **index** будет установлен на текущую итерацию цикла для каждого контекста шаблона.
- **сначала** будет установлено логическое значение, указывающее, является ли элемент первым в итерации.
- **last** будет установлен на логическое значение, указывающее, является ли элемент последним в итерации.
- **даже** будет установлено логическое значение, указывающее, имеет ли этот элемент четный индекс.
- **нечетное** будет установлено в логическое значение, указывающее, имеет ли этот элемент нечетный индекс.

Прочитайте Для цикла онлайн: https://riptutorial.com/ru/angular/topic/9826/для-цикла

глава 4: маршрутизация

Examples

Маршрутизация с детьми

Я нашел, что это способ правильно встраивать маршруты детей внутри файла app.routing.ts или app.module.ts (в зависимости от ваших предпочтений). Этот подход работает при использовании WebPack или SystemJS.

В приведенном ниже примере показаны маршруты для дома, дома / счетчика и данных home / counter / fetch. Первый и последний маршруты являются примерами перенаправления. Наконец, в конце примера приведен правильный способ экспорта маршрута, который должен быть импортирован в отдельный файл. Напр. app.module.ts

Для дальнейшего объяснения, Угловая требует, чтобы у вас был беспроблемный маршрут в массиве children, который включает в себя родительский компонент, для представления родительского маршрута. Это немного запутанно, но если вы думаете о пустом URL-адресе для детского маршрута, он по существу будет равен тому же URL-адресу, что и родительский маршрут.

```
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 { }
```

Отличный пример и описание через Siraj

Основная маршрутизация

Маршрутизатор позволяет осуществлять навигацию с одного вида на другой на основе взаимодействия пользователя с приложением.

Ниже приводятся шаги по реализации базовой маршрутизации в Angular -

ПРИМЕЧАНИЕ . Убедитесь, что у вас есть этот тег:

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

как первый ребенок под вашим тегом заголовка в вашем файле index.html. Этот элемент указывает, что папка вашего приложения является корнем приложения. Угловой тогда будет знать, как организовать ваши ссылки.

1. Проверьте, указали ли вы правильные / последние зависимости маршрутизации в package.json (используя последнюю версию Angular) и что вы уже `npm install` -

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

2. Определите маршрут согласно определению интерфейса:

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

3. В файле маршрутизации (`routes/app.routing.ts`) импортируйте все компоненты, которые необходимо настроить для разных маршрутов маршрутизации. Пустой путь означает, что представление загружается по умолчанию. «:» в пути указывает динамический параметр, переданный загруженному компоненту.

```
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. В свой `app.module.ts` поместите это под `@NgModule([])` под `imports` :

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

5. Загружать / отображать компоненты маршрутизатора на основе доступа к пути. Директива `<router-outlet>` используется для указания угловой нагрузки, где загружается компонент.

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

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

6. Свяжите другие маршруты. По умолчанию `RouterOutlet` загрузит компонент , для которого пустой путь указан в `Routes` . Директива `RouterLink` используется с тэгом html anchor для загрузки компонентов, подключенных к маршрутам. `RouterLink` генерирует атрибут href, который используется для создания ссылок. Например:

```
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 { }
```

Теперь мы хорошо справляемся с маршрутизацией на статические пути. `RouterLink` поддерживает динамический путь, передавая дополнительные параметры вместе с этим путем.

```
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` берет массив, где первым параметром является путь для маршрутизации, а последующие элементы - для параметров динамической маршрутизации.

Прочитайте маршрутизация онлайн: https://riptutorial.com/ru/angular/topic/9827/маршрутизация

глава 5: Обмен данными между компонентами

Вступление

Цель этой темы - создать простые примеры нескольких способов обмена данными между компонентами посредством привязки данных и совместного использования.

замечания

В программировании всегда есть много способов решения одной задачи. Не стесняйтесь редактировать текущие примеры или добавлять свои собственные.

Examples

Отправка данных с родительского компонента на дочерний с помощью общей службы

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;
  }

}
```

Отправлять данные из родительского компонента в дочерний компонент посредством привязки данных с помощью @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 = [];
}
```

Отправка данных от дочернего к родительскому через эмитент события @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;
  }
}
```

Отправка данных асинхронно от родительского к дочернему с использованием Observable и 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;
      });

  }

}
```

Прочитайте Обмен данными между компонентами онлайн:
https://riptutorial.com/ru/angular/topic/10836/обмен-данными-между-компонентами

глава 6: Событие Эмитент

Examples

Захват события

Создайте сервис-

```
import {EventEmitter} from 'angular2/core';
export class NavService {
    navchange: EventEmitter<number> = new EventEmitter();
    constructor() {}
    emitNavChangeEvent(number) {
        this.navchange.emit(number);
    }
    getNavChangeEmitter() {
        return this.navchange;
    }
}
```

Создайте компонент для использования сервис-

```
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);
    }
}
```

Прочитайте Событие Эмитент онлайн: https://riptutorial.com/ru/angular/topic/9828/событие-эмитент

глава 7: трубы

Вступление

Трубы очень похожи на фильтры в AngularJS, поскольку они помогают преобразовать данные в заданный формат. Характер трубы | используется для применения труб в Угловом.

Examples

Пользовательские трубы

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 { }
```

Несколько пользовательских труб

Наличие разных труб - очень распространенный случай, когда каждая труба выполняет другую вещь. Добавление каждого канала к каждому компоненту может стать повторяющимся кодом.

Можно объединить все часто используемые трубы в один `Module` и импортировать этот новый модуль в любой компонент, который нуждается в трубах.

breaklines.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 />');
    }
}
```

uppercase.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,
  ],
})
```

Прочитайте трубы онлайн: https://riptutorial.com/ru/angular/topic/9824/трубы

глава 8: формы

Examples

Реактивные формы

app.module.ts

Добавьте их в свой файл app.module.ts для использования реактивных форм

```
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>
```

validators.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
            };
        }
    }
}
```

Шаблоны, управляемые формами

Шаблон - `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>
```

Компонент - `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`;
  }

}
```

Модель - `signup-request.model.ts`

```
export class SignUpRequest {

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

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

}
```

Модуль приложения - `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 { }
```

Компонент приложения - `app.component.html`

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

Прочитайте формы онлайн: https://riptutorial.com/ru/angular/topic/9825/формы

кредиты

S. No	Главы	Contributors
1	Начало работы с угловым	aholtry, Anup Kumar Gupta, BogdanC, Community, daddycool, Edric, Fahad Nisar, Faisal, Hendrik Brummermann, Philipp Kief, Tom
2	RXJS и наблюдаемые	aholtry
3	Для цикла	aholtry
4	маршрутизация	0mpurdy, aholtry, Edric
5	Обмен данными между компонентами	0mpurdy, BogdanC, JGFMK, Nehal
6	Событие Эмитент	aholtry
7	трубы	aholtry, Amr ElAdawy
8	формы	aholtry, Saka7