



Web-технологии

Основы работы с Angular

- Основы фреймворка Angular
 - структура приложения, компоненты
 - сервисы, иерархия сервисов
 - директивы, привязка
 - задание маршрутов, параметры маршрутизации
 - шаблонные переменные
 - работа с дочерними компонентами
 - валидация данных
 - взаимодействие с сервером

<https://angular.io/>

<https://metanit.com/web/angular2/>

<https://material.angular.io/>

<https://www.tutorialspoint.com/angular2/>

<https://www.w3schools.com/angular/>

https://developer.mozilla.org/en-US/docs/Learn/Tools_and_testing/Client-side_JavaScript_frameworks/Angular_getting_started

<https://angularplayground.it/>

<https://protractor.angular.io/>

https://stackblitz.com/


3


The online code editor f x +


← → 🔒 stackblitz.com The online code editor for web apps. Powered by Visual St... 📄 📌 S ↓


≡ ⚡ **StackBlitz** 🔍 🐙 SIGN IN


FRONT-END FRAMEWORKS & LIBRARIES


 **ANGULAR**
TYPESCRIPT


 **REACT**
JAVASCRIPT


 **REACT (TS)**
TYPESCRIPT

 **VUE 3**
JAVASCRIPT


 **SVELTE**
JAVASCRIPT


 **RXJS**
TYPESCRIPT


 **IONIC**
TYPESCRIPT

 **ANGULARJS**
JAVASCRIPT

BARE BONES STARTERS

 **STATIC**
HTML/JS/CSS

 **JAVASCRIPT**
BLANK PROJECT

 **TYPESCRIPT**
BLANK PROJECT

<https://stackblitz.com/fork/angular-ivy>

<https://codesandbox.io/s/angular>

4

The screenshot shows the CodeSandbox web interface for an Angular project. The browser address bar displays the URL `https://codesandbox.io`. The page title is "Angular - CodeSandbox". The interface includes a sidebar on the left with a file explorer showing the project structure: `src` (containing `app`, `assets`, `environments`, `index.html`, `main.ts`, `polyfills.ts`, `styles.css`, and `typings.d.ts`), `.angular-cli.json`, `package.json`, and `tsconfig.json`. Below the file explorer is a "Dependencies" section with a search bar and a list of installed packages: `@angular/animations ^12.2.0...`, `@angular/common ^12.2.0...`, and `@angular/core ^12.2.0...`. The main editor area displays the `main.ts` file with the following TypeScript code:

```
1 import { enableProdMode } from "@angular/core";
2 import { platformBrowserDynamic } from "@angular/platform-browser-dynamic";
3
4 import { AppModule } from "./app/app.module";
5 import { environment } from "./environments/environment";
6
7 if (environment.production) {
8   enableProdMode();
9 }
10
11 platformBrowserDynamic()
12   .bootstrapModule(AppModule)
13   .catch(err => console.log(err));
14
```

On the right side, there is a "Browser" preview window showing a "Welcome to CodeSandbox!" message and the Angular logo. Below the browser preview is a "Console" window with 1 message and a "Problems" window with 0 errors. The bottom status bar shows the file path `bb2167d81`, the cursor position `Ln 1, Col 1`, and the configuration `Spaces: 2 UTF-8 LF TypeScript 4.1.2`.

<https://angular.io/>

5

Установка Angular

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

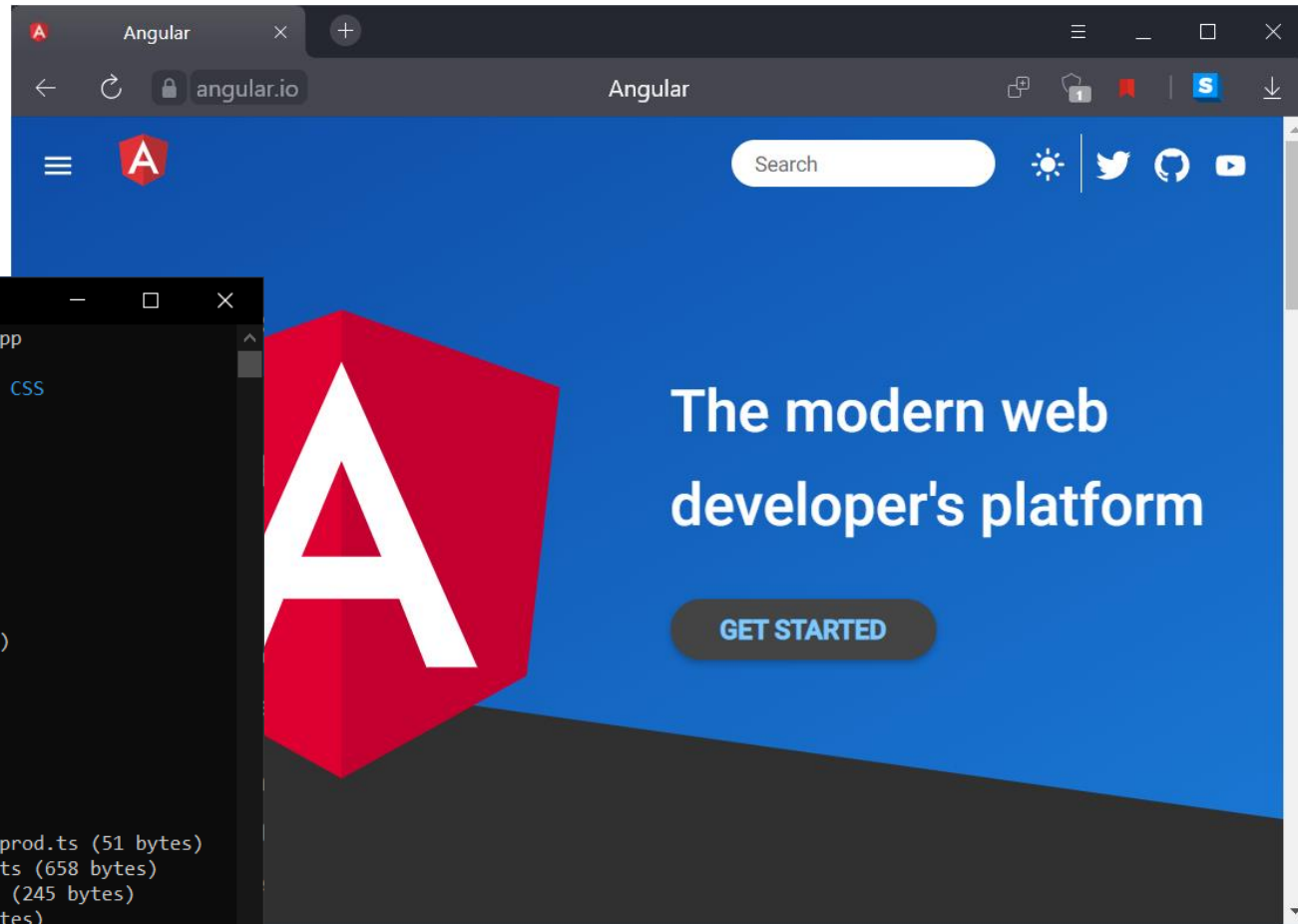
Проверка версии

```
ng version
```

Создание проекта

```
ng new my-app
```

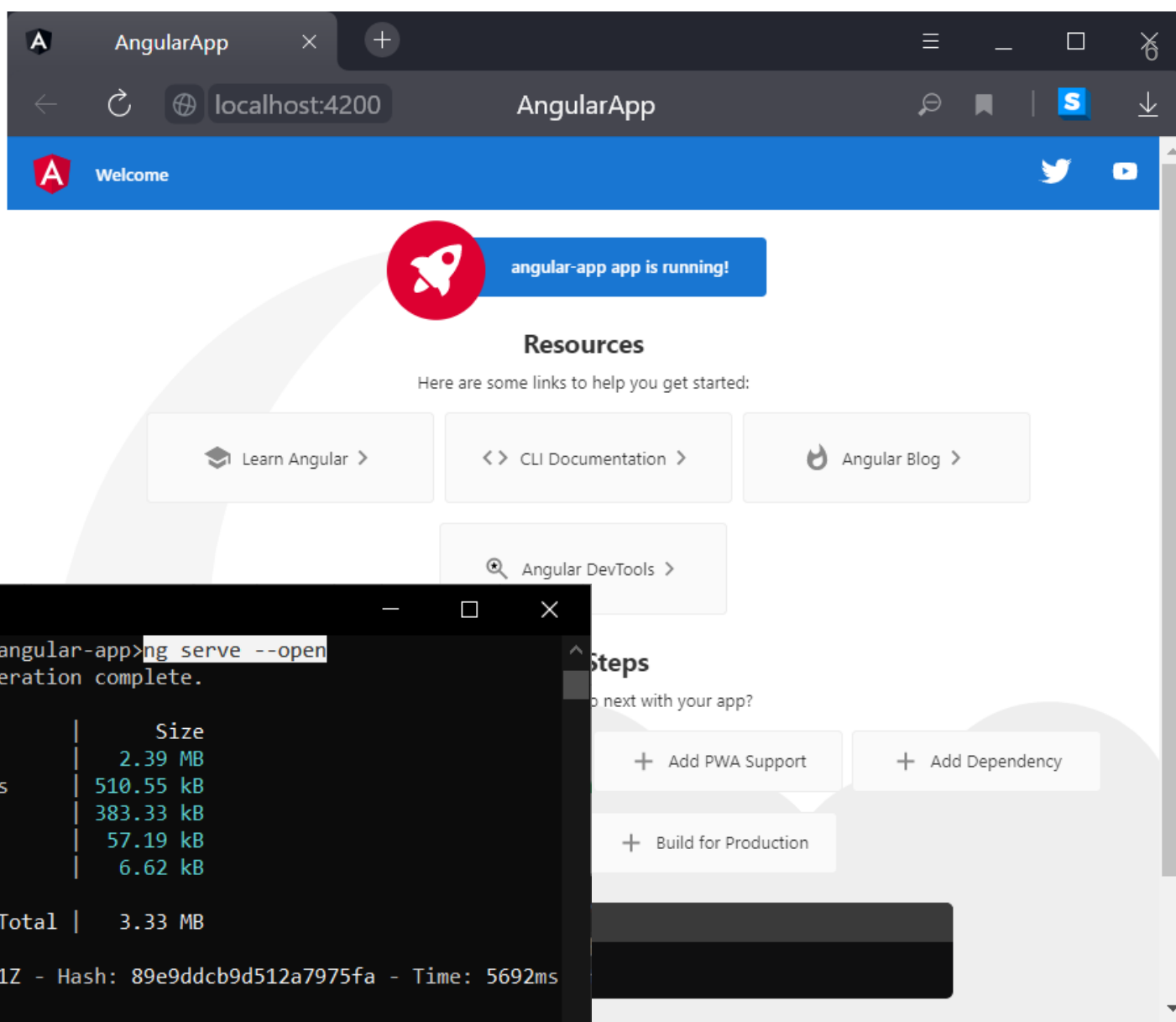
```
npm
C:\Users\serge\WebstormProjects>ng new angular-app
? Would you like to add Angular routing? Yes
? Which stylesheet format would you like to use? CSS
CREATE angular-app/angular.json (3069 bytes)
CREATE angular-app/package.json (1073 bytes)
CREATE angular-app/README.md (1056 bytes)
CREATE angular-app/tsconfig.json (783 bytes)
CREATE angular-app/.editorconfig (274 bytes)
CREATE angular-app/.gitignore (604 bytes)
CREATE angular-app/.browserslistrc (703 bytes)
CREATE angular-app/karma.conf.js (1428 bytes)
CREATE angular-app/tsconfig.app.json (287 bytes)
CREATE angular-app/tsconfig.spec.json (333 bytes)
CREATE angular-app/src/favicon.ico (948 bytes)
CREATE angular-app/src/index.html (296 bytes)
CREATE angular-app/src/main.ts (372 bytes)
CREATE angular-app/src/polyfills.ts (2820 bytes)
CREATE angular-app/src/styles.css (80 bytes)
CREATE angular-app/src/test.ts (788 bytes)
CREATE angular-app/src/assets/.gitkeep (0 bytes)
CREATE angular-app/src/environments/environment.prod.ts (51 bytes)
CREATE angular-app/src/environments/environment.ts (658 bytes)
CREATE angular-app/src/app/app-routing.module.ts (245 bytes)
CREATE angular-app/src/app/app.module.ts (393 bytes)
CREATE angular-app/src/app/app.component.html (24617 bytes)
CREATE angular-app/src/app/app.component.spec.ts (1088 bytes)
CREATE angular-app/src/app/app.component.ts (215 bytes)
CREATE angular-app/src/app/app.component.css (0 bytes)
\ Installing packages (npm)...
```



CLI = command line
interaction

Запуск

ng serve --open



```
Select Windows PowerShell
C:\Users\serge\WebstormProjects\angular-app>ng serve --open
√ Browser application bundle generation complete.

Initial Chunk Files | Names      | Size
vendor.js           | vendor     | 2.39 MB
polyfills.js        | polyfills  | 510.55 kB
styles.css, styles.js | styles    | 383.33 kB
main.js             | main       | 57.19 kB
runtime.js          | runtime    | 6.62 kB

| Initial Total | 3.33 MB

Build at: 2021-09-13T18:07:43.501Z - Hash: 89e9ddcb9d512a7975fa - Time: 5692ms

** Angular Live Development Server is listening on localhost:4200, open your browser on http://localhost:4200/ **

√ Compiled successfully.
```

Если всё правильно
указано в package.json:
npm start

Отладка: Angular DevTools

7

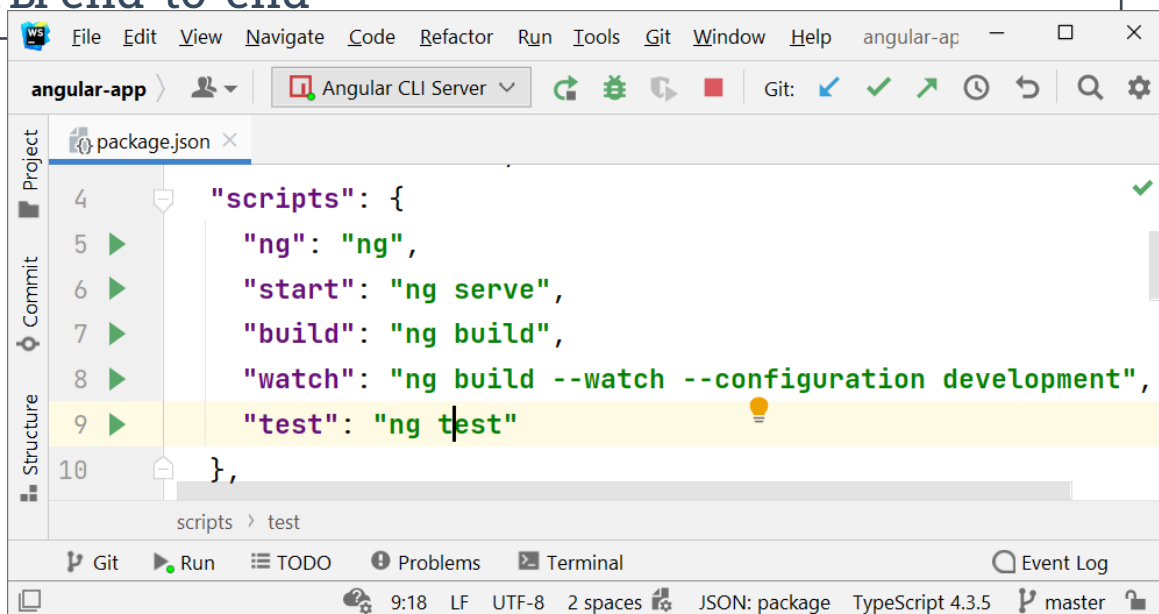
The image shows a screenshot of a web browser displaying the Angular DevTools extension page on the Chrome Web Store. The browser's address bar shows the URL `chrome.google.com` and the page title is "Angular DevTools - Интернет-магазин Chrome". The page features the Angular logo, the extension name "Angular DevTools", and the author "Автор: Angular". A blue "Установить" (Install) button is visible in the top right. Below the extension name, there are 80 star ratings and a link to "Инструменты разработчика" (Developer Tools). A navigation bar includes links for "Обзор" (Overview), "Меры по обеспечению конфиденциальности" (Privacy Policy), "Отзывы" (Reviews), "Поддержка" (Support), and "Похожие" (Similar).

Below the navigation bar, a preview of the Angular DevTools interface is shown. The interface has a dark theme and includes a "Components" panel on the left, a "Profiler" panel on the right, and a "Properties" panel on the right. The "Components" panel shows a tree view of the application's components, with "mat-toolbar" selected. The "Profiler" panel shows the "mat-toolbar" component's performance metrics. The "Properties" panel shows the component's properties, including `_color: primary`, `_document: HTMLDocument`, `_elementRef: {...}`, `_platform: {...}`, `_toolbarRows: {...}`, and `defaultColor: undefined`.

Утилиты Angular

8

Утилита	Задача
ng build	Компилирует приложение Angular app в build-директорию
ng serve	Собирает и обслуживает приложение, перестраивая его при изменении файлов
ng generate	Создает или перестраивает файлы, основываясь на схемах
ng test	Запускает модульные тесты на заданном проекте
ng e2e	Собирает и запускает Angular приложение, затем запускает тесты end-to-end



```
File Edit View Navigate Code Refactor Run Tools Git Window Help angular-ap - □ ×
angular-app > Angular CLI Server
package.json ×
4 "scripts": {
5   "ng": "ng",
6   "start": "ng serve",
7   "build": "ng build",
8   "watch": "ng build --watch --configuration development",
9   "test": "ng test"
10 },
scripts > test
Git Run TODO Problems Terminal Event Log
9:18 LF UTF-8 2 spaces JSON: package TypeScript 4.3.5 master
```


Особенности Angular2+

- **Angular** – фреймворк, поддерживаемый Microsoft
- Компоненты – обеспечивают деление приложения на составные части
- Сервисы – блоки кода, которые могут использоваться различными компонентами

Модули – блоки кода, решающие отдельные задачи
@NgModule

Компоненты – составные части модулей
@Component

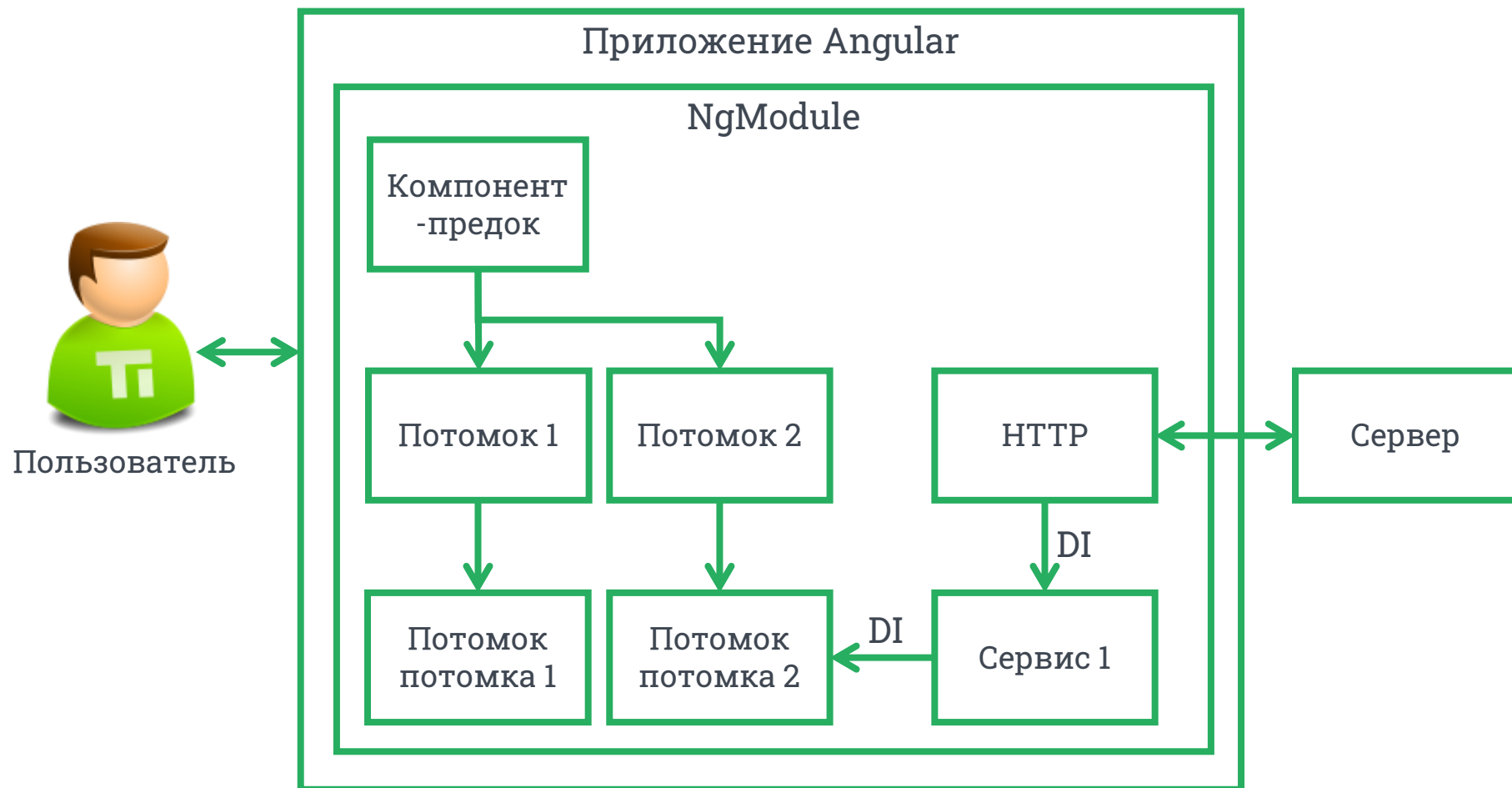
Шаблоны – формы для отображения
template

Метаданные – предоставление дополнительной информации

Сервисы – компоненты, доступные в рамках всего приложения
@Injectable

Пример архитектуры приложения

10



@NgModule
@Component
@Injectable

- модуль
- компонент
- сервис с возможностью DI

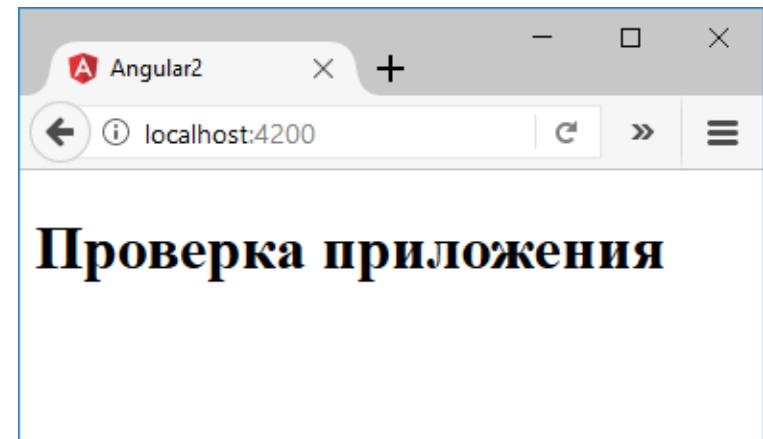
@Component. Hello World. {{title}}

11

app.components.ts

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

@Component({
  selector: 'app-root', // Имя в index.html
  template: `
    <h1>{{title}}</h1>
    ` // Шаблон для отображения
})
export class AppComponent {
  title = 'Проверка приложения'; // Название
}
```



@Component. CSS

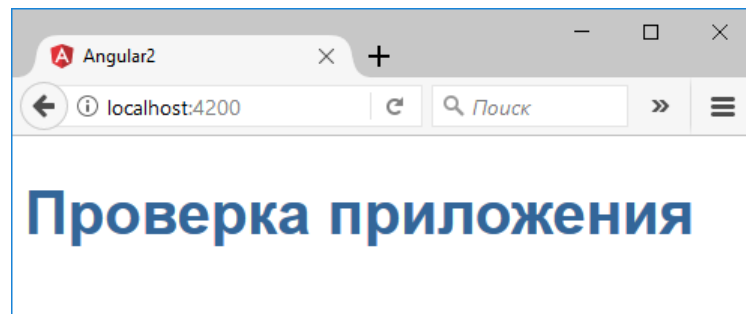
app.components.ts

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

@Component({
  selector: 'app-root', // Имя в index.html
  styleUrls: ['app.component.css'], // Стили
  template: `
    <h1>{{title}}</h1>
    ` // Шаблон для отображения
})
export class AppComponent {
  title = 'Проверка приложения'; // Название
}
```

app.component.css

```
h1 {
  color: #369;
  font-family: Arial, Helvetica, sans-serif;
  font-size: 250%;
}
```



@NgModule

13

app.module.ts

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

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

Все возможные свойства декоратора:

- **declarations** – компоненты, директивы, pipe
- **exports** – экспорт
- **imports** – импорт
- **providers** – сервисы, доступные во всём приложении
- **bootstrap** – корневой компонент

- **import** обеспечивает импорт функциональности существующих модулей
- декоратор **NgModule** используется для объявлений, импорта и задания начальной загрузки
- **BrowserModule** – модуль, используемый во всех web-приложениях
- **bootstrap** – определяет модули для начальной загрузки

// Модуль запуска приложения

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

// Основной компонент

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

// Запуск основного компонента

```
platformBrowserDynamic().bootstrapModule(AppModule)  
  .catch(err => console.error(err));
```

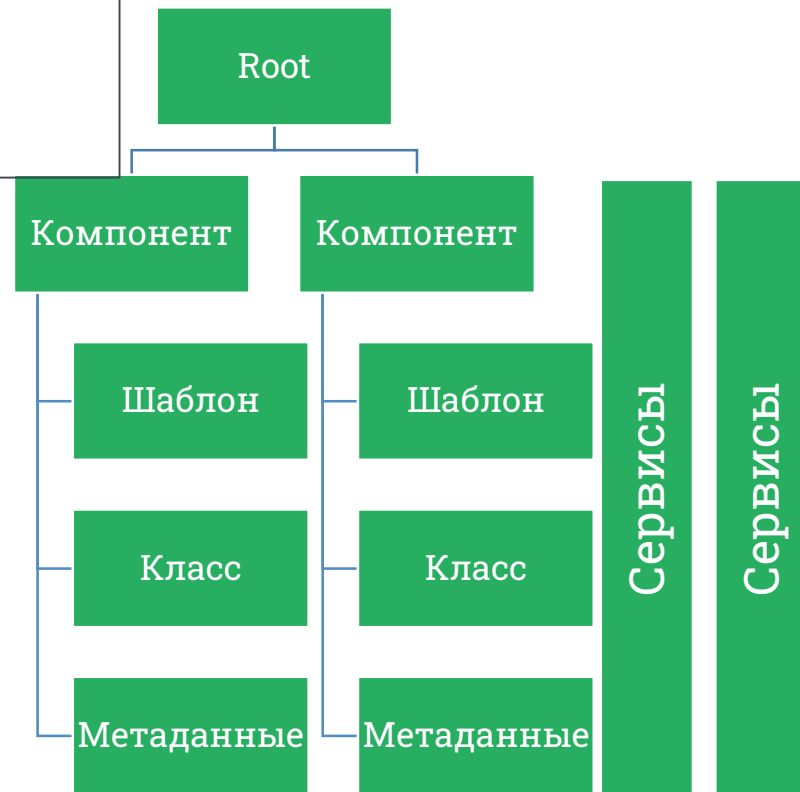
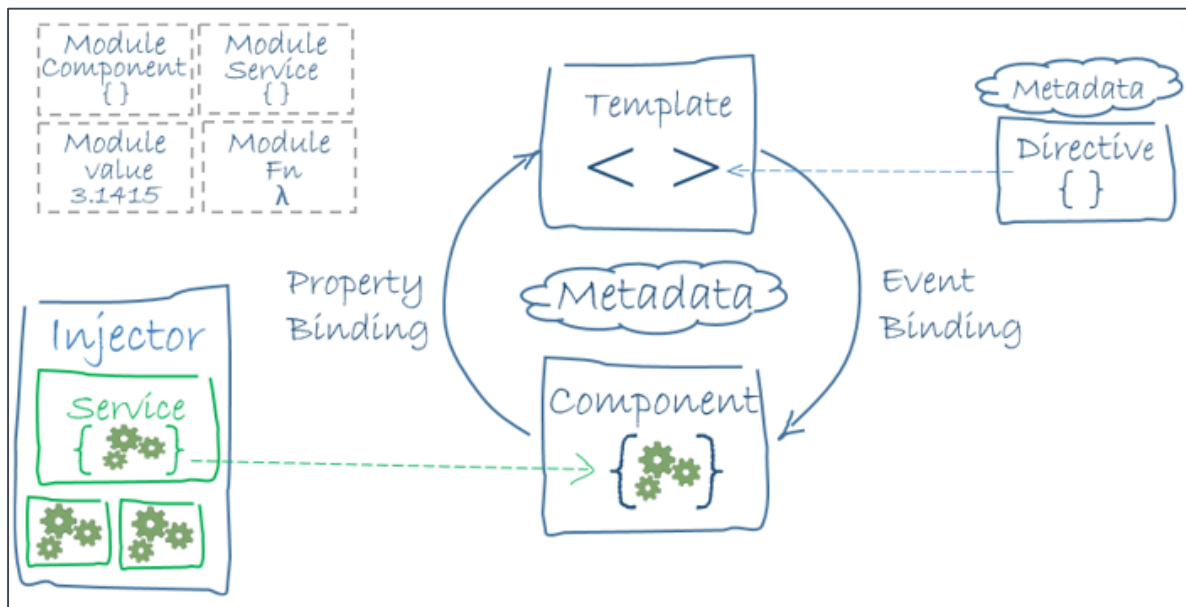
Корневая директория

Файл	Назначение
<code>src/</code>	Исходные коды
<code>e2e/</code>	Тесты end-to-end
<code>node_modules/</code>	Папка для модулей node.js
<code>.angular-cli.json</code>	Конфигурация Angular CLI
<code>.editorconfig</code>	Конфигурация редактора
<code>.gitignore</code>	Контроль коммитов в Git
<code>karma.conf.js</code>	Модульные тесты на движке Karma
<code>package.json</code>	Описание приложения
<code>protractor.conf.js</code>	Конфигурация тестов end-to-end на основе Protractor
<code>README.md</code>	Базовая документация
<code>tsconfig.json</code>	Конфигурация компилятора TypeScript
<code>tslint.json</code>	Конфигурация TSLint

Файл	Назначение
app/app.component.{ts,html,css,spec.ts}	Корневой компонент приложения
app/app.module.ts	Корневой модуль, который сообщает, как собирать приложение
assets/*	Для рисунков и других материалов
environments/*	Конфигурации целевой среды
favicon.ico	Фавикон
index.html	Главная страница
main.ts	Точка входа для приложения (язык – TypeScript)
polyfills.ts	Настройки для работы с различными браузерами
styles.css	Глобальные стили
test.ts	Точка входа для модульных тестов
tsconfig.{app spec}.json	Конфигурационные файлы компилятора

Структура приложения

17



Класс, шаблон, метаданные

Синтаксис класса

```
class classname {
    Propertyname: PropertyType = Value
}
```

Синтаксис шаблона

```
template: `
    <HTML code>
    class properties
`
```

```
templateUrl: 'путь_к_шаблону.html',
```

Синтаксис метаданных

1 @ИмяДекоратора (TS-класс)

2 **class** classname {
 @ИмяДекоратора (параметры)
 Propertyname: PropertyType = Value
}

Пример оформления шаблона, template

19

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

@Component({
  selector: 'app-root', // Имя в index.html
  styleUrls: ['app.component.css'], // Стили
  template: `
    <h1>{{title}}</h1>
    ` // Шаблон для отображения
})
export class AppComponent {
  title: string = 'Проверка приложения'; // Название
}
```

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

@Component({
  selector: 'app-root', // Имя в index.html
  styleUrls: ['app.component.css'], // Стили
  templateUrl: 'app.component.html'
})

export class AppComponent {
  title: string = 'Проверка приложения'; // Название
}
```

Альтернатива styleUrls
– свойство styles.

```
styles: [
  h1 { color: green; }
  p { font-size: 12px; }
]
```

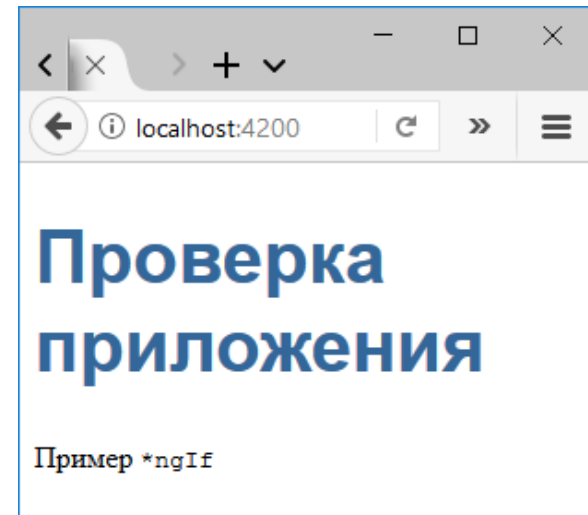
Селектор **:host**
ссылается на
элемент, в
котором
хостится
КОМПОНЕНТ

Директива ngIf

20

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

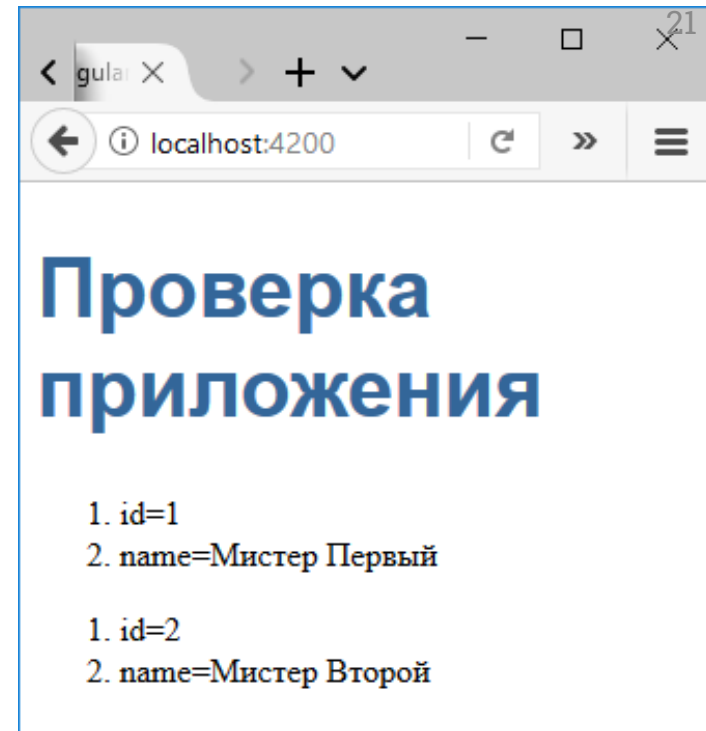
@Component({
  selector: 'app-root', // Имя в index.html
  styleUrls: ['app.component.css'], // Стили
  template: `
    <h1>{{title}}</h1>
    <div *ngIf = 'appStatus'>Пример <code>*ngIf</code></div>
  ` // Шаблон для отображения
})
export class AppComponent {
  title: string = 'Проверка приложения'; // Название
  appStatus: boolean = true; // Доп.переменная
}
```



Директива ngFor

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

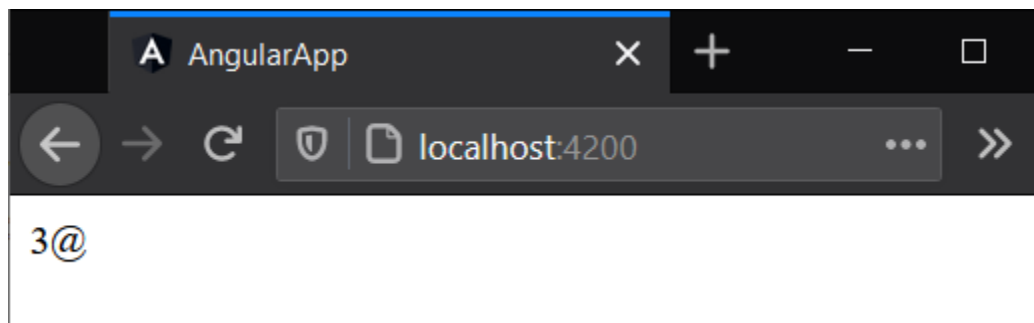
```
@Component({
  selector: 'app-root', // Имя в index.html
  styleUrls: ['app.component.css'], // Стили
  template: `
    <h1>{{title}}</h1>
    <div *ngFor = 'let item of appList'>
      <ol>
        <li>id={{item.id}}</li>
        <li>name={{item.name}}</li>
      </ol>
    </div>
  ` // Шаблон для отображения
})
export class AppComponent {
  title: string = 'Проверка приложения'; // Название
  appList: any[] = [{ // Доп. переменная
    'id': '1',
    'name': 'Мистер Первый'
  },
  {
    'id': '2',
    'name': 'Мистер Второй'
  }
];
}
```



Директива ngSwitch

22

```
import { Component } from '@angular/core';
@Component({
  selector: 'app-root', // switch..case..default
  template: `<p [ngSwitch]="value">
    <ng-template ngSwitchCase="1">{{value}}$</ng-template>
    <ng-template ngSwitchCase="2">{{value}}#</ng-template>
    <ng-template ngSwitchDefault>{{value}} @</ng-template>
  </p>`
})
export class AppComponent {
  value: number = 3; // Только число!
}
```



- [ngSwitch]
- ngSwitchCase
- ngSwitchDefault
- <ng-template>

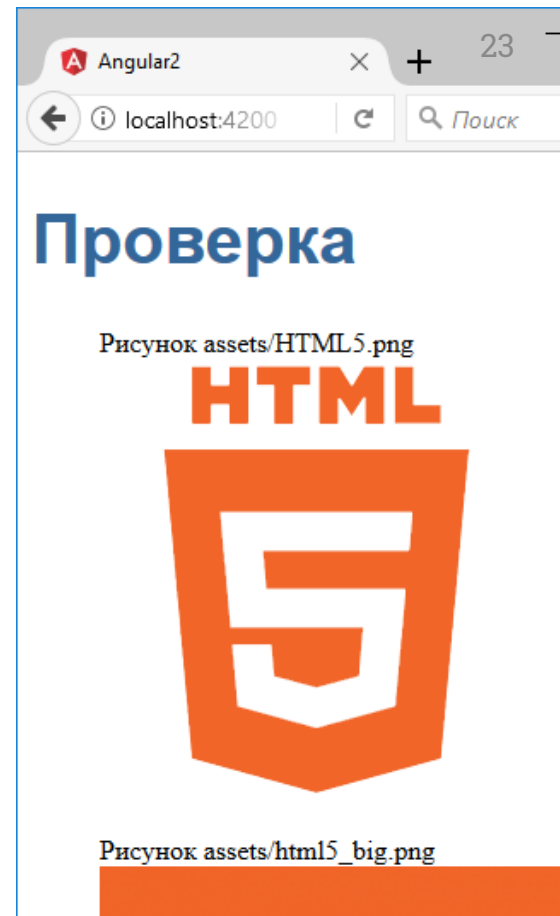
Альтернативная запись

```
template: `<p [ngSwitch]="value">
  <a *ngSwitchCase="1">{{value}}$</a>
  <a *ngSwitchCase="2">{{value}}#</a>
  <a *ngSwitchDefault>{{value}} @</a>
</p>`
```

Привязка - binding, [src]

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

@Component({
  selector: 'app-root', // Имя в index.html
  styleUrls: ['app.component.css'], // Стили
  template: `
    <h1>{{title}}</h1>
    <div *ngFor = 'let item of appList'>
      <ol>
        Рисунок {{item.url}}<br>
        <img [src]='item.url'> <!--Связывание-->
      </ol>
    </div>
  ` // Шаблон для отображения
})
export class AppComponent {
  title: string = 'Проверка'; // Название
  appList: any[] = [{ // Доп.переменная
    'id': '1',
    'url': 'assets/HTML5.png'
  },
  {
    'id': '2',
    'url': 'assets/html5_big.png'
  }
];
}
```



Если класс описан:

```
export class className {
  property: propertytype = value;
}
```

то привязка реализуется

`<htmltag htmlproperty='property'>`

Роутинг (1), компоненты

24

myCompA.component.ts

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

@Component ({
  selector: 'app-root',
  template: 'myCompA_template',
})
export class myCompA {}
```

myCompB.component.ts

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

@Component ({
  selector: 'app-root',
  template: 'myCompB_template',
})
export class myCompB {}
```

app.component.ts

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

@Component ({
  selector: 'app-root', // Имя в index.html
  styleUrls: ['app.component.css'], // Стили
  template: `
    <ul>
      <li><a [routerLink] = "['/myCompA']">myCompA</a></li>
      <li><a [routerLink] = "['/myCompB']">myCompB</a></li>
    </ul>
    <router-outlet></router-outlet>` // Шаблон для отображения
})
export class AppComponent {}
```

<router-outlet></router-outlet>
- место отображения
компонента

Альтернативное объявление

myCompA

Роутинг (2), Routes, forRoot, path

25

app.module.ts

```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import { myCompA } from './myCompA.component';
import { myCompB } from './myCompB.component';
import { RouterModule, Routes } from '@angular/router';
```

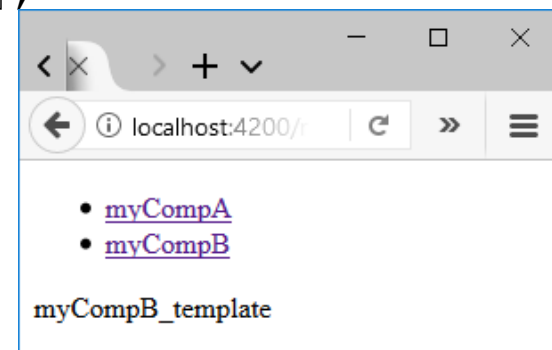
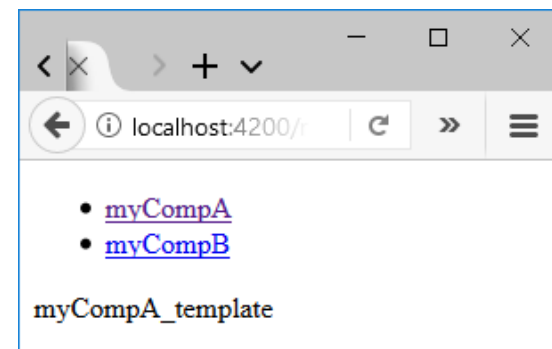
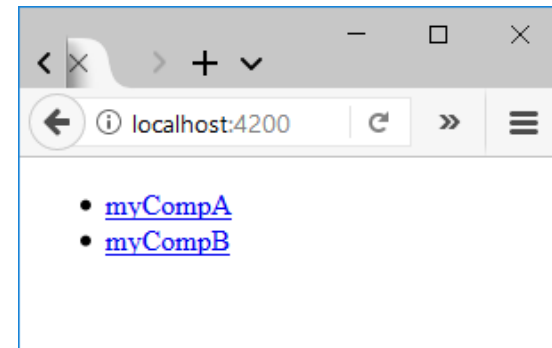
```
const appRoutes: Routes = [
  { path: 'myCompA', component: myCompA },
  { path: 'myCompB', component: myCompB },
];
```

```
@NgModule({
  declarations: [AppComponent, myCompA, myCompB],
  imports: [BrowserModule, RouterModule.forRoot(appRoutes)],
  providers: [],
  bootstrap: [AppComponent],
})
```

```
export class AppModule { }
```

- **Router** – маршрутизатор из **@angular/router**
- **<router-outlet>** – директива-заполнитель
- **Routes** – массив маршрутов
- **[routerLink]** – директива ссылки на маршрут
- **ActivatedRoute** – «активный» маршрут

- **forRoot()** – для корня
- **forChild()** – не для корня



Роутинг (3), navigate(), Router

26

app.component.ts

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

```
@Component({  
  selector: 'app-root',  
  templateUrl: './app.component.html',  
  styleUrls: ['./app.component.css']  
})  
export class AppComponent {  
  title = 'angular-app';  
  constructor(private _router: Router) {}  
  navigateToB() {  
    this._router.navigate(["/myCompB"])  
  }  
}
```

В шаблоне

```
<button (click)="navigateToB()">Click</button>
```

Роутинг (4), not found

app.module.ts

```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import { myCompA } from './myCompA.component';
import { myCompB } from './myCompB.component';
import { PageNotFound } from './NotFound.component';
import { RouterModule, Routes } from '@angular/router';

const appRoutes: Routes = [
  { path: 'myCompA', component: myCompA },
  { path: 'myCompB', component: myCompB },
  { path: '**', component: PageNotFound },
];

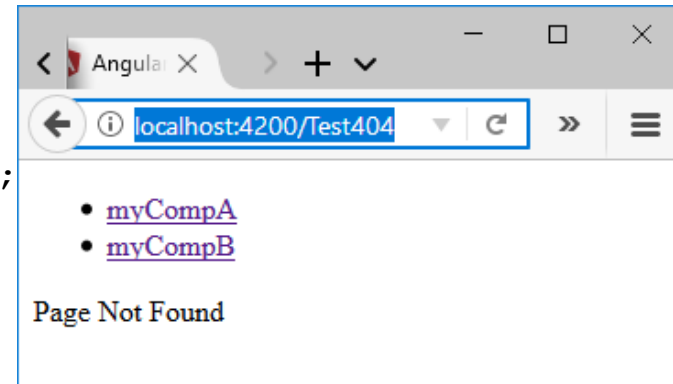
@NgModule({
  declarations: [AppComponent, myCompA, myCompB, PageNotFound],
  imports: [BrowserModule, RouterModule.forRoot(appRoutes)],
  providers: [],
  bootstrap: [AppComponent],
})
export class AppModule { }
```

NotFound.component.ts

27

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

@Component ({
  selector: 'app-root',
  template: 'Page Not Found',
})
export class PageNotFound { }
```



Если нужен указатель на
«корень», то path: ""

Роутинг (5), «дочерние» маршруты, children

```
const routes: Routes = [
  {
    path: 'first-component',
    component: FirstComponent, // Компонент <router-outlet> в шаблоне
    children: [
      {
        path: 'child-a', // дочерний путь A
        component: myCompA, // Компонент A
      },
      {
        path: 'child-b', // дочерний путь B
        component: myCompB, // Компонент B
      },
    ],
  },
];
```

Передача данных по маршруту, :id, params, routerLink, ActivatedRoute

myCompA.components.ts

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

```
@Component ({
  selector: 'app-root',
  template: 'myCompA_template {{myid}}',
})
```

```
export class myCompA {
  myid: string; // Параметр
  constructor(route: ActivatedRoute) {
    // Чтение параметра
    this.myid = route.snapshot.params["id"]
  }
}
```

Фрагмент app.module.ts

```
const appRoutes: Routes = [
  { path: 'myCompA/:id', component: myCompA },
  { path: 'myCompA', component: myCompA },
  { path: 'myCompB', component: myCompB },
];
```

Фрагмент шаблона

```
<a [routerLink] = "['/myCompA', '35']">myCompA</a>
```

Дополнительные
параметры для
строки запроса

```
<a [routerLink]= "['/myCompA', '35']"
[queryParams]= "{ 'type': 'phone',
'from': 100 }">
```

Для кода

```
this.router.navigate(
  ['/myCompA', "35"],
  {
    queryParams:{
      'type': 'phone',
      'from': '100'
    }
  }
);
```

URL при обращении:

http://localhost:4200/myCompA/35

Передача статических данных, data 30

myCompA.components.ts

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

@Component ({
  selector: 'app-root',
  template: 'myCompA_template {{myid}}, test: {{mytest}}',
})
export class myCompA {
  myid: string;
  mytest: string; // Для статики
  constructor(route: ActivatedRoute) {
    this.myid = route.snapshot.params["id"]
    if(route.snapshot.data[0])
      this.mytest = route.snapshot.data[0]["test"]
    else
      this.mytest = "no"
  }
}
```

Фрагмент app.module.ts

```
const appRoutes: Routes = [
  // Передача статических данных в data
  { path: 'myCompA:id', component: myCompA, data: [{test: "abcd"}] },
  { path: 'myCompA', component: myCompA },
  { path: 'myCompB', component: myCompB },
];
```

Резюме по маршрутам

- Маршруты настраиваются на уровне модулей
- Каждый маршрут имеет путь соответствующий компоненту
- Область, где отображается содержимое маршрута, определена **<router-outlet>** в шаблоне
- При навигации по маршруту используются
 - **routerLink**
 - метод **navigate()**
- Параметр для маршрута конфигурируется в свойстве **path** в конфигурации маршрута и передаётся в **routerLink** или **navigate()**
- Если маршрут принимает параметр, то компонент должен иметь конструктор с аргументом типа **ActivatedRoute**
- Маршруты-потомки конфигурируются с помощью свойства **children** интерфейса **Route**
- Приложение может показывать больше одного маршрута одновременно

Перенаправление

```
{ path: 'contact', redirectTo: '/about', pathMatch:'full'},
```

Навигация, (click)

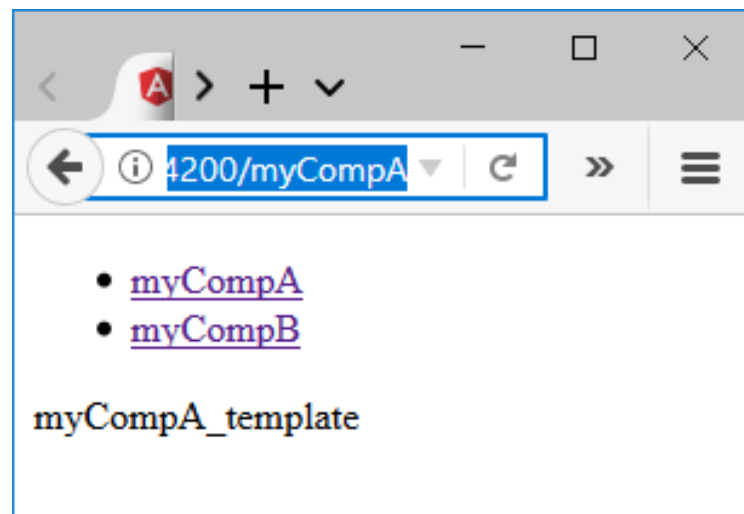
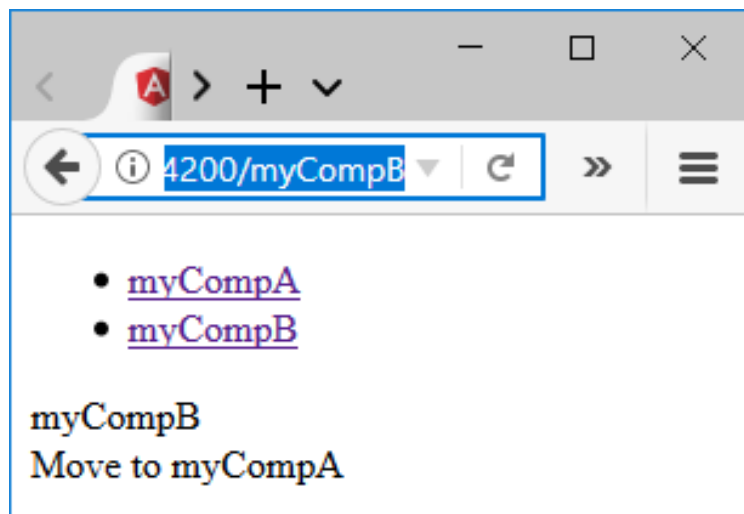
myCompB.component.ts

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

```
@Component ({
  selector: 'app-root',
  template: `myCompB<br>
  <a class = "button" (click) = "myFunc()">Move to myCompA</a>` ,
})
```

```
export class myCompB {
  constructor(private router: Router){}
  myFunc(): void {
    this.router.navigate([' /myCompA']);
  }
}
```

В качестве
параметра можно
передать \$event



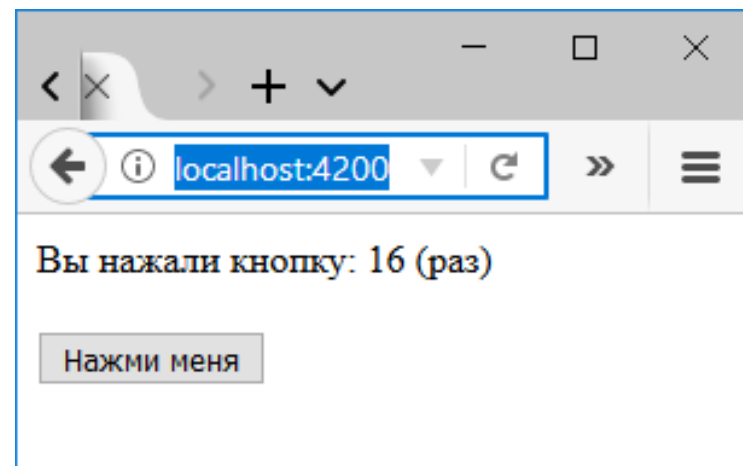
Обработка событий, clicked()

33

app.component.ts

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

```
@Component({  
  selector: 'app-root', // Имя в index.html  
  template: `  
    <div>  
      Вы нажали кнопку: {{counter}} (раз)  
    </div><br>  
    <button (click)="clicked()">Нажми меня</button>  
  `,  
})  
export class AppComponent {  
  counter: number = 0;  
  clicked(event) {  
    this.counter++;  
  }  
}
```



Привязка данных

- привязка событий для вызова функций, которые обрабатывают эти события
- привязка атрибутов для обновления текстовых значений атрибутов элементов HTML
- привязка свойств для обновления значений свойств элементов DOM
- привязка шаблонов для преобразования шаблонов представления
- двухсторонняя привязка данных с помощью ngModel

Связывание/привязка (binding)

35

Тип	Цель	Примеры
Property	Element property Component property Directive property	<code></code> <code><app-hero-detail [hero]="currentHero"></app-hero-detail></code> <code><div [ngClass]="{'special': isSpecial}"></div></code>
Event	Element event Component event Directive event	<code><button (click)="onSave()">Save</button></code> <code><app-detail (deleteRequest)="deleteHero()"></app-detail></code> <code><div (myClick)="clicked=\$event" clickable>click me</div></code>
Two-way	Event and property	<code><input [(ngModel)]="name"></code>
Attribute	Attribute (the exception)	<code><button [attr.aria-label]="help">help</button></code>
Class	class property	<code><div [class.special]="isSpecial">Special</div></code>
Style	style property	<code><button [style.color]="isSpecial ? 'red' : 'green'"></code>

«Односторонняя» привязка / \$event

app.component.ts

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

```
@Component ({
  selector: 'app-root',
  template: `
    <div>
      Внедрение переменной: {{developer}}<br>
      Внедрение элемента из массива:
      {{groups[0]}} , {{groups[1]}} , {{groups[2]}}
    </div><br>
    <div>
      Обновляемое поле:<br>
      <input [value]="name" (input)="name=$event.target.value">
      {{name}}
    </div>
  `
})
```

```
export class AppComponent {
  developer: string = "Сергей";
  groups: string[] = ["5381", "5303", "5304", "5382"];
  name: string="Текст, просто текст"
}
```

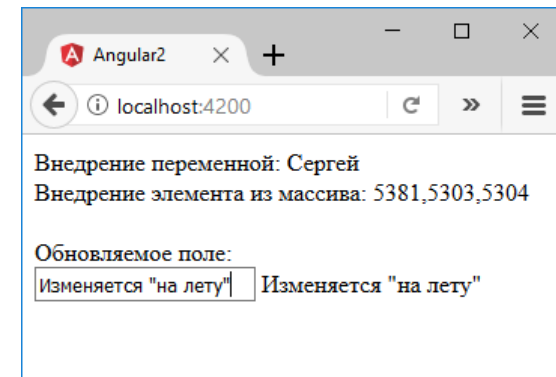
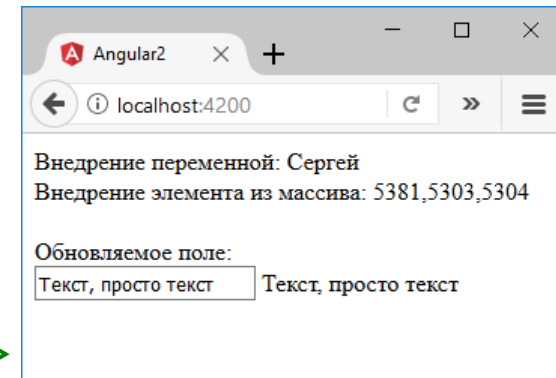
[value] = "name" - Привязка выражения name к свойству value элемента input

(input) = "expression" - декларативная связь с событием

name = \$event.target.value - выражение, которое будет выполнено при возникновении события

\$event - события

36



«Односторонняя» привязка, событие `keyup`

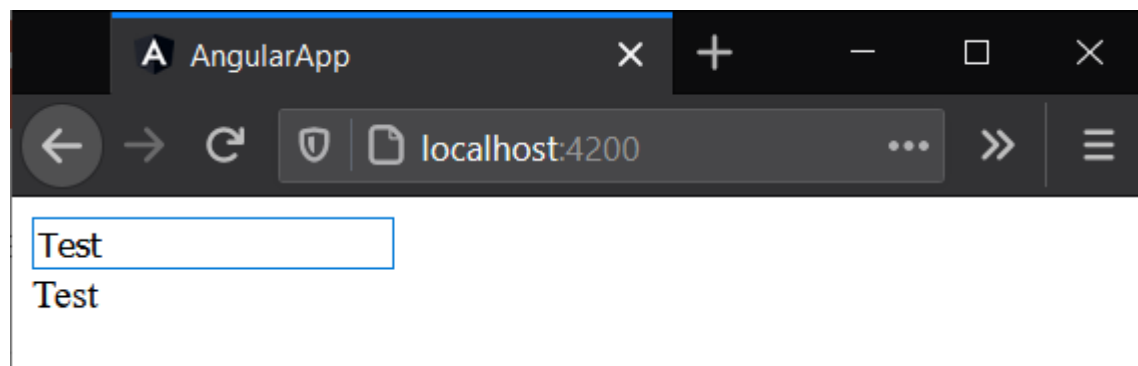
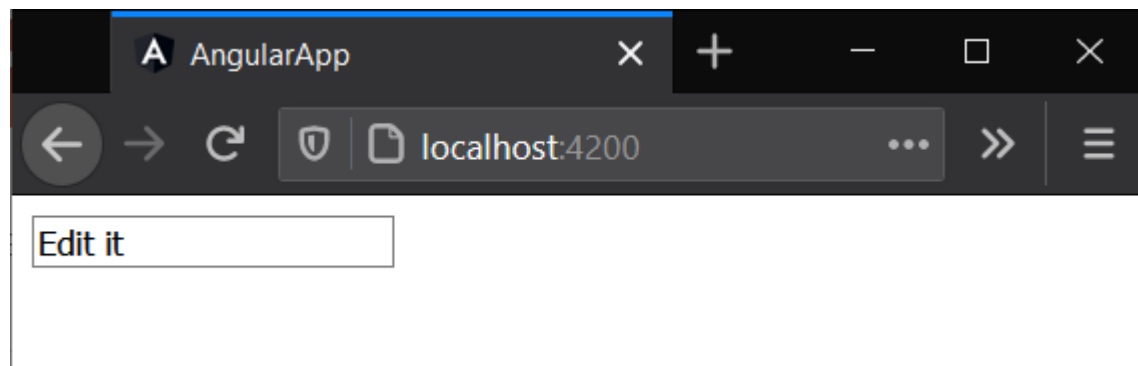
`app.component.ts`

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

```
@Component({
  selector: 'app-root',
  template: `
    <input #edit [value]="appValue" (keyup)="save(edit.value)">
    <br>{{result}}
  `
})
export class AppComponent {
  appValue = 'Edit it'
  result = ""
  save(val:string): void {
    this.result = val
  }
}
```

Важно!

Если в шаблоне **result** заменить на **appValue**, то всё равно в **save()** придётся присваивать значение



«Двусторонняя» привязка (1), ngModel

38

myItem.ts

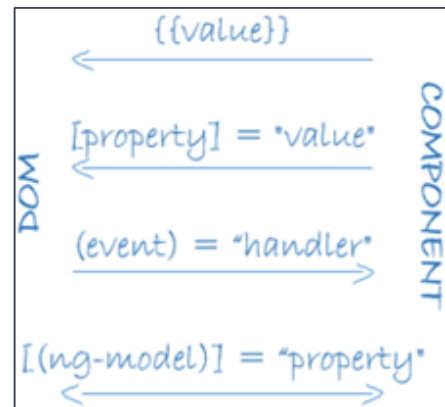
```
export class MyItem {  
  // Создание двух свойств id, title  
  constructor (  
    public myid: number,  
    public title: string  
  ) {}  
}
```

myitem-form.component.html

```
<div class="container">  
  <h1>Форма ввода данных</h1>  
  <form>  
    <div class="form-group">  
      <label for="myid">ID</label>  
      <input type="text" class="form-control" id="myid" required  
        [(ngModel)]="item.myid" name="id">  
    </div>  
  
    <div class="form-group">  
      <label for="title">Название</label>  
      <input type="text" class="form-control" id="title"  
        [(ngModel)]="item.title" name="title">  
    </div>  
  </form>  
</div>
```

myitem-form.component.ts

```
import { Component } from '@angular/core';  
import { MyItem } from './myItem';  
  
@Component ({  
  selector: 'myitem-form',  
  templateUrl: './myitem-form.component.html'  
})  
  
export class MyItemFormComponent {  
  item = new MyItem(1, 'ABC');  
}
```



Для корректной работы в модуле
нужен импорт

```
import {FormsModule} from '@angular/forms';
```

продолжение



«Двусторонняя» привязка (2), myitem-form

app.component.ts

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

```
@Component({
  selector: 'app-root', // Имя в index.html
  styleUrls: ['app.component.css'], // Стили
  template: '<myitem-form></myitem-form>'
})
```

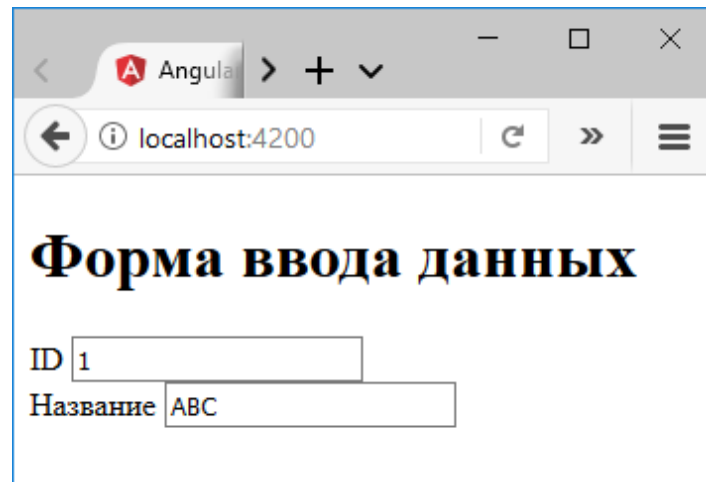
```
export class AppComponent {
}
```

app.module.ts

```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import { FormsModule } from '@angular/forms';
import { MyItemFormComponent } from './myitem-form.component';
```

```
@NgModule({
  declarations: [AppComponent, MyItemFormComponent],
  imports: [BrowserModule, FormsModule],
  providers: [],
  bootstrap: [AppComponent],
})
```

```
export class AppModule { }
```



Шаблонные переменные / #имя, lowercase

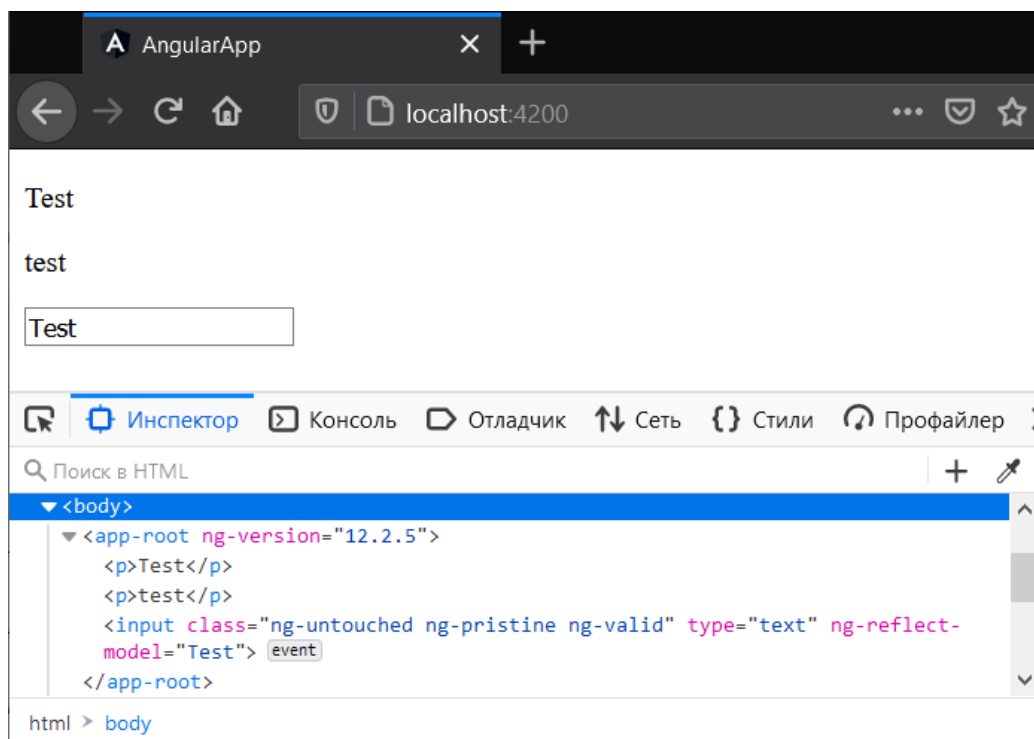
```
import { Component } from '@angular/core';
```

```
@Component({
  selector: 'app-root',
  template: `
    <p #userName>{{name}}</p> <!--Объявление шаблонной переменной-->
    <p>{{userName.textContent|lowercase}}</p> <!--Использование-->
    <input type="text" [(ngModel)]="name" /> <!--Источник изменений--> `
})
export class AppComponent {
  name:string="Test";
}
```

Для определения «шаблонных» переменных применяется знак решетки «#»

Обратите внимание на цепочку

- name:string [(ngModel)]
- #userName→name
- userName.textContent

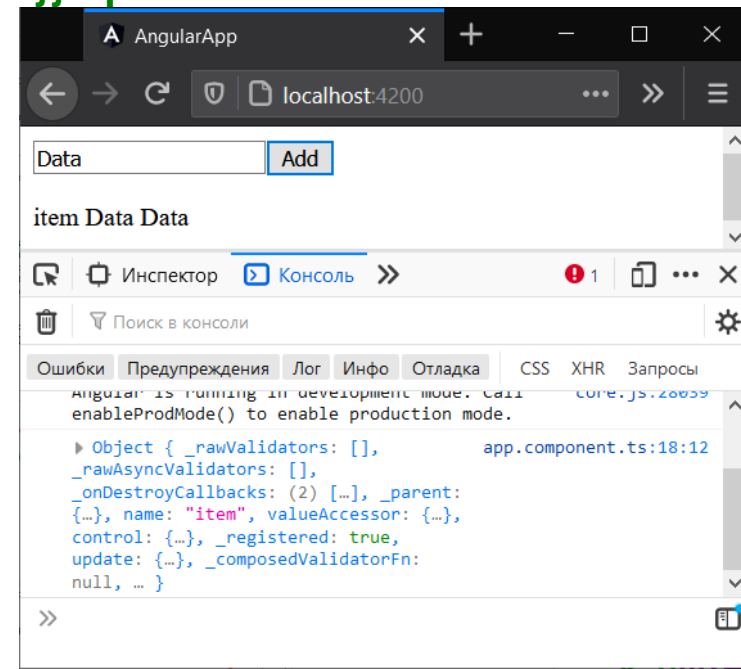


Изменение модели / #, NgModel

41

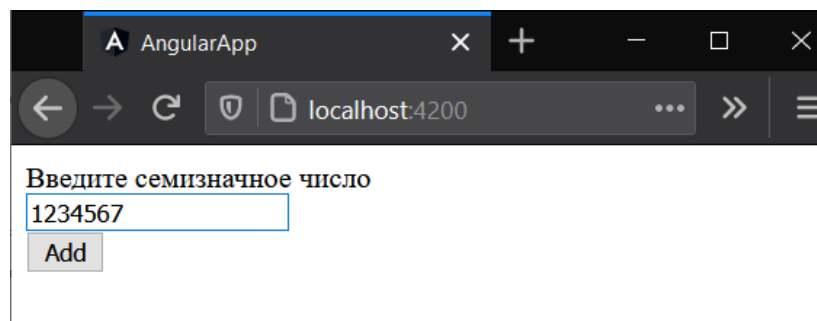
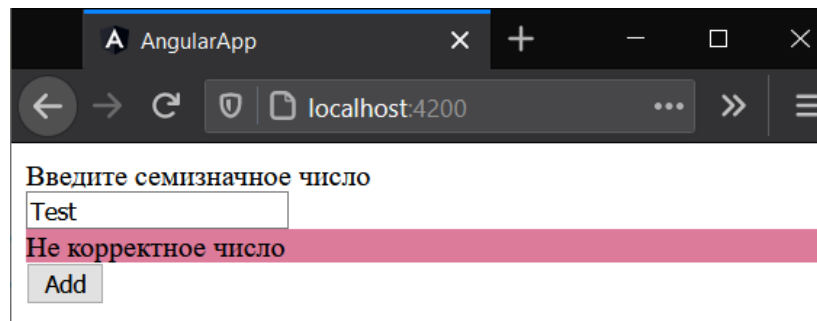
```
import { Component } from '@angular/core';
import { NgModel } from '@angular/forms';
@Component({
  selector: 'app-root',
  template: `
    <div>
      <form>
        <input [(ngModel)]="myValue" #myElem="ngModel" name="item">
        <button (click)="add(myElem)">Add</button>
      </form>
    </div>
    <p>{{myElem.name}} {{myElem.value}} {{myElem.model}}</p>
  `
})
export class AppComponent {
  myValue: string = "Test";
  add(data: NgModel) {
    console.log(data)
  }
}
```

Атрибут `<input>`
`(ngModelChange)="onMyChange()"`
позволяет контролировать ввод



Валидация / required, pattern, [hidden], .valid

```
import { Component } from '@angular/core';
@Component({
  selector: 'app-root',
  template: `
<div> <!-- Валидация с помощью required && pattern -->
  <label>Введите семизначное число</label><br>
  <input [(ngModel)]="myValue" #myElem="ngModel" name="item"
    required pattern="[0-9]{7}"><br>
  <div [hidden]="myElem.valid" class="wrong">Не корректное число</div>
  <button (click)="send()">Add</button>
</div>`,
  styles: [`.wrong { background-color: #D7698CE0 }`]
})
export class AppComponent {
  myValue: string = "Test";
  send() {
    console.log(this.myValue)
  }
}
```



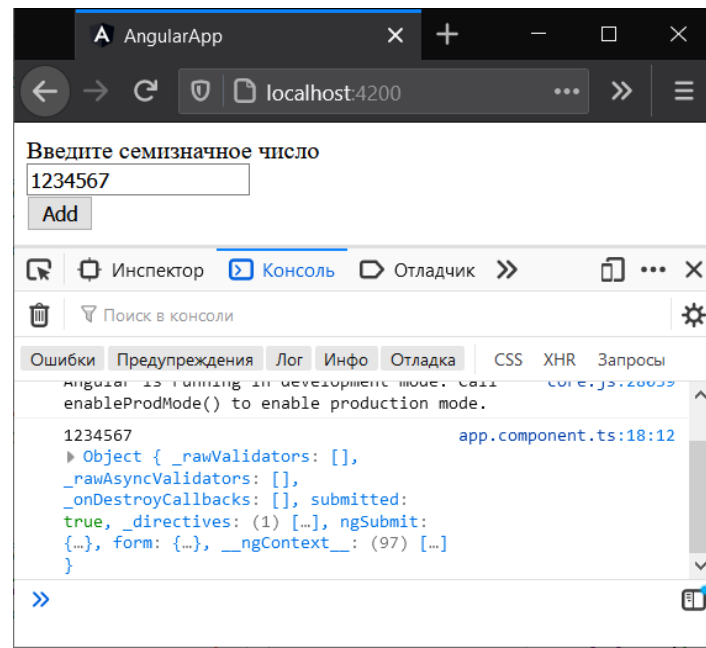
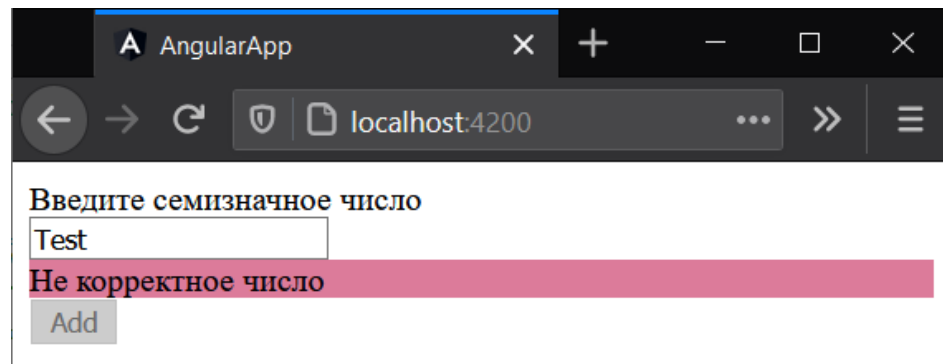
Валидация / NgForm, [disabled], .invalid

43

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

```
@Component({
  selector: 'app-root',
  template: `
    <form #myForm="ngForm" novalidate>
      <label>Введите семизначное число</label><br>
      <input [(ngModel)]="myValue" #myElem="ngModel" name="item"
        required pattern="[0-9]{7}"><br>
      <div [hidden]="myElem.valid" class="wrong">Не корректное число</div>
      <button (click)="send(myForm)" [disabled]="myForm.invalid">Add</button>
    </form>`,
  styles: [`.wrong { background-color: #D7698CE0 }`]
})
```

```
export class AppComponent {
  myValue: string = "Test";
  send(form: NgForm) {
    console.log(this.myValue, form)
  }
}
```




Контроль наблюдаемого потока

- Наблюдаемым потоком является объект, отправляющий элементы из некоторого источника данных (сокета, массива, событий интерфейса) по одному:
 - отправка следующего элемента
 - генерация ошибки
 - отправка сигнала завершения потока

let mySubscription: Subscription = someObservable.subscribe(myObserver);

- Отмена подписки:

mySubscription.unsubscribe();

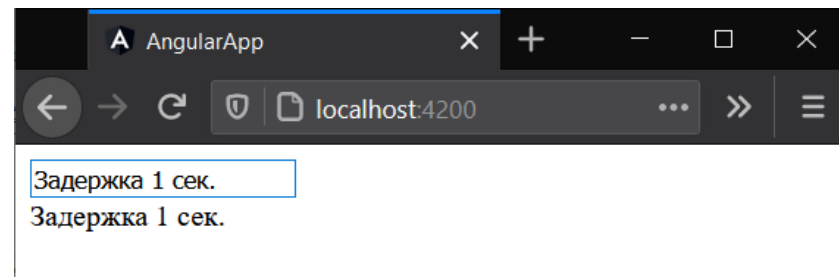


Потребуется дальше
для работы с
сервером

Подписка на изменение компонента / FormControl, pipe, debounceTime, subscribe

```
import { Component } from '@angular/core';
import { FormControl } from '@angular/forms';
import { debounceTime } from 'rxjs/operators';
```

```
@Component({
  selector: 'app-root',
  template: `
    <input type="text" [formControl]="inputValue">
    <br>{{myValue}}
  `,
  providers: []
})
export class AppComponent {
  inputValue: FormControl = new FormControl("");
  myValue: string = ""
  constructor() {
    this.inputValue.valueChanges // Контроль изменения в поле
      .pipe(debounceTime(1000)) // Задержка потока
      // Подписка на изменения
      .subscribe((value:any) => this.myValue = value)
  }
}
```



Данные появляются
с задержкой в 1 сек.

В **app.module.ts** добавлен импорт в **@NgModule**

```
import { FormsModule,
  ReactiveFormsModule } from
  '@angular/forms';
```

Объект класса **FormControl** содержит текущее значение **<input>**, информацию о статусе валидации, а также сведения о том, был ли он изменен

Внедрение зависимостей (сервисы)

DI – Dependency Injection

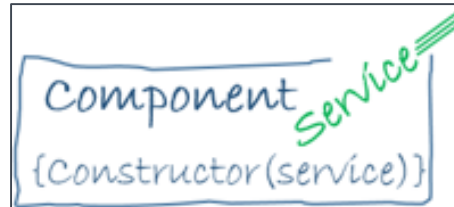
46

app.myInjectService.ts

```
import {Injectable} from '@angular/core';
@Injectable()
export class myAppService {
  getValue(): string {
    return 'Привет МИР';
  }
}
```

app.component.ts

```
import {Component} from '@angular/core';
import {myAppService} from '../app.myInjectService';
@Component({
  selector: 'app-root', // Имя в index.html
  styleUrls: ['app.component.css'], // Стили
  template: '<div>{{value}}</div>',
  providers: [myAppService]
})
export class AppComponent {
  value: string = '';
  constructor(private appService: myAppService) { }
  ngOnInit(): void {
    this.value = this.appService.getValue();
  }
}
```

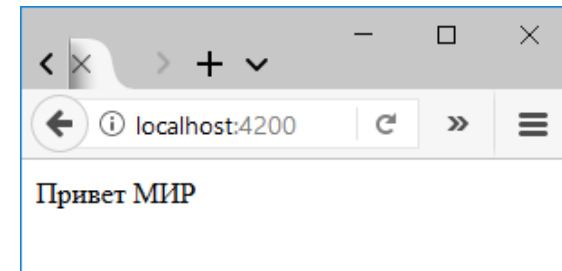


1. Импорт модуля **myInjectService** (**@Injectable**)

2. Регистрация сервиса, как поставщика данных (**providers**)

3. Создаём переменную **appService** в **AppComponent**, чтобы к ней можно было обращаться по всему приложению

3. Для примера в момент инициализации (**ngOnInit**) вызван метод **getValue** и результат помещен в **value**



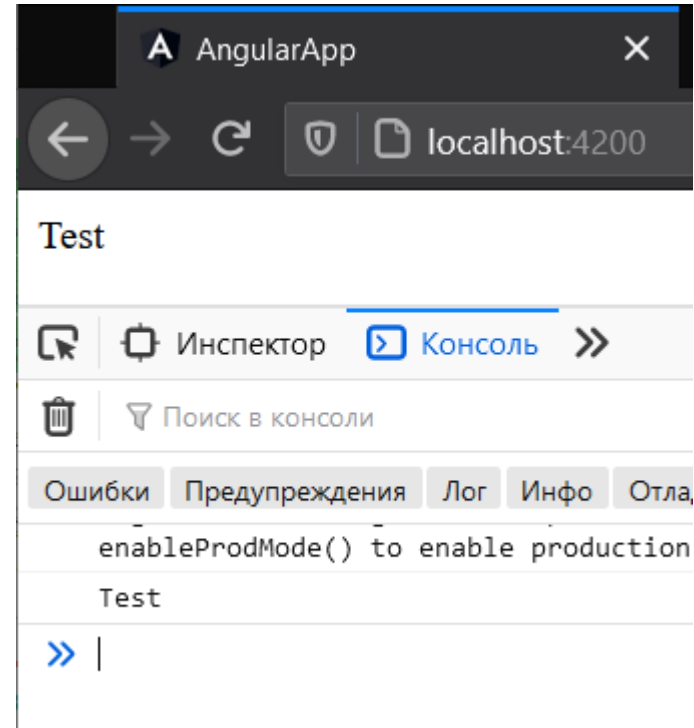
Опциональные сервисы, @Optional

47

app.component.ts

```
import { Component, Optional } from '@angular/core';
import { LogService } from './log.service';

@Component({
  selector: 'app-root',
  template: `
    <div>{{getData()}}</div>`,
  providers: [LogService] // Можно закомментировать
})
export class AppComponent {
  value: string = "Test";
  constructor(@Optional() private logService: LogService) {}
  getData() {
    // Если закомментировать providers,
    // тогда this.logService == false
    if(this.logService)
      this.logService.log(this.value)
    return this.value
  }
}
```



log.service.ts

```
import { Injectable } from '@angular/core';
@Injectable()
export class LogService {
  public log(value: string = "empty"): void {
    console.log(value);
  }
}
```

Иерархия сервисов / providedIn

- Пример

- Мы создали два одинаковых компонента, которые используют одинаковый сервис
 - Одинаковые компоненты используют независимые сервисы
 - Если необходим общий сервис, то его необходимо зарегистрировать в модуле

- Иерархия сервисов

1. Глобальный или корневой уровень (root) – один сервис на всё приложение

1 `@Injectable({providedIn: 'root'})`

2. Уровень модуля – один сервис на модуль

1 `@Injectable({providedIn: AppModule})` // Доступ в AppModule

2 `providers: [LogService]`

3. Уровень компонента – один сервис на компонент

1 `providers: [LogService]`

Параметризация вывода (стандартные каналы)

app.component.ts

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

```
@Component({
```

```
  selector: 'app-root',
```

```
  template: `
```

```
    <div>
```

```
      Имя: {{myname}}<br>
```

```
      Пример lowercase: {{myname|lowercase}}<br>
```

```
      Пример uppercase: {{myname|uppercase}}<br>
```

```
      Пример slice:1:2: {{myname|slice:1:2}}<br>
```

```
      Пример slice:2:4: {{myname|slice:2:4}}<br>
```

```
      Пример date: {{current|date:"dd.MM.yyyy"}}<br>
```

```
      Пример currency: {{index|currency}}<br>
```

```
      Пример percent: {{float|percent}}<br>
```

```
    </div>
```

```
  })
```

```
export class AppComponent {
```

```
  myname: string = "Сергей";
```

```
  current = new Date(2017, 10, 5);
```

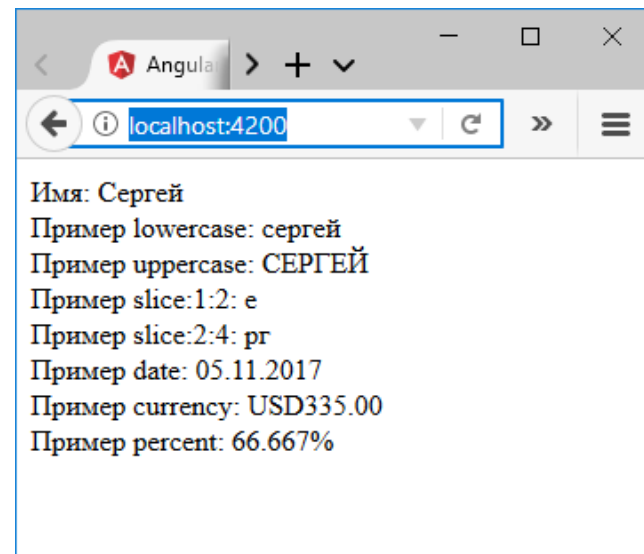
```
  index: number = 335;
```

```
  float: number = 2/3;
```

```
}
```

Стандартные каналы:

- **uppercase** – верхний регистр
- **date** – формат даты
- **currency** – формат валюты
- **async** – извлечение данных из обёртки



Канал @Pipe

app.component.ts

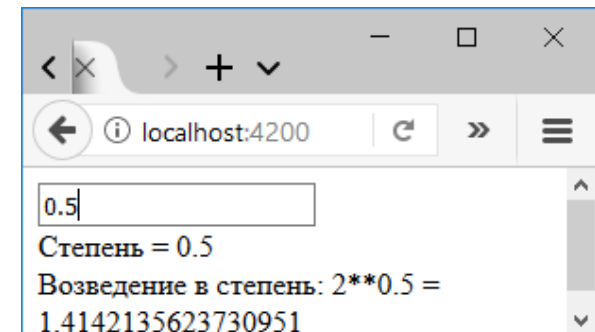
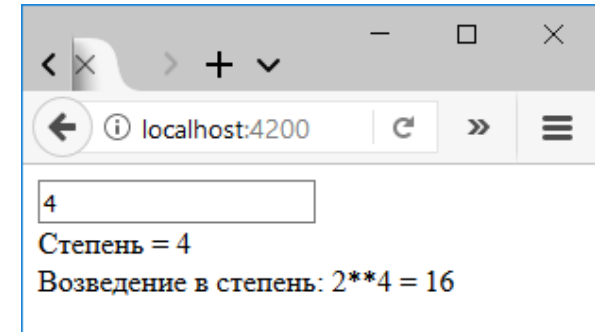
```
import {Component}
from '@angular/core';
@Component({
  selector: 'app-root',
  template: `
<div>
<input [value]="myValue"
(input)="myValue=$event.target.value"><br>
Степень = {{myValue}}<br>
Возведение в степень: 2**{{myValue}} = {{2|Power:myValue}}
</div>`
})
export class AppComponent {
  myValue: string = "4";
}
```

app.module.ts

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

power.pipe.ts

```
import {Pipe, PipeTransform} from '@angular/core';
@Pipe ({
  name: 'Power'
})
export class PowerPipe implements PipeTransform {
  transform(value: number, power: string): number {
    let myPower = parseFloat(power);
    return value ** myPower;
  }
}
```



- **ngOnChanges** – вызывается до метода **ngOnInit** при начальной установке свойств, которые связаны механизмом привязки, а также при любой их переустановке или изменении их значений.
 - В качестве параметра принимает объект класса **SimpleChanges**, который содержит предыдущие и текущие значения свойства.
- **ngOnInit** – вызывается один раз после установки свойств компонента, которые участвуют в привязке
- **ngDoCheck** – вызывается при каждой проверке изменений свойств компонента сразу после методов **ngOnChanges** и **ngOnInit**
 - **ngAfterContentInit** – вызывается один раз после метода **ngDoCheck** после вставки содержимого в представление компонента кода html
 - **ngAfterContentChecked** – вызывается при проверке изменений содержимого, которое добавляется в представление компонента. Вызывается после метода **ngAfterContentInit** и после каждого последующего вызова метода **ngDoCheck**
 - **ngAfterViewInit** – вызывается после инициализации представления компонента, а также представлений дочерних компонентов. Вызывается только один раз сразу после первого вызова метода **ngAfterContentChecked**
 - **ngAfterViewChecked** – вызывается после проверки на изменения в представлении компонента, а также проверки представлений дочерних компонентов. Вызывается после первого вызова **ngAfterViewInit** и после каждого последующего вызова **ngAfterContentChecked**
- **ngOnDestroy** – вызывается перед тем, как компонент будет удалён

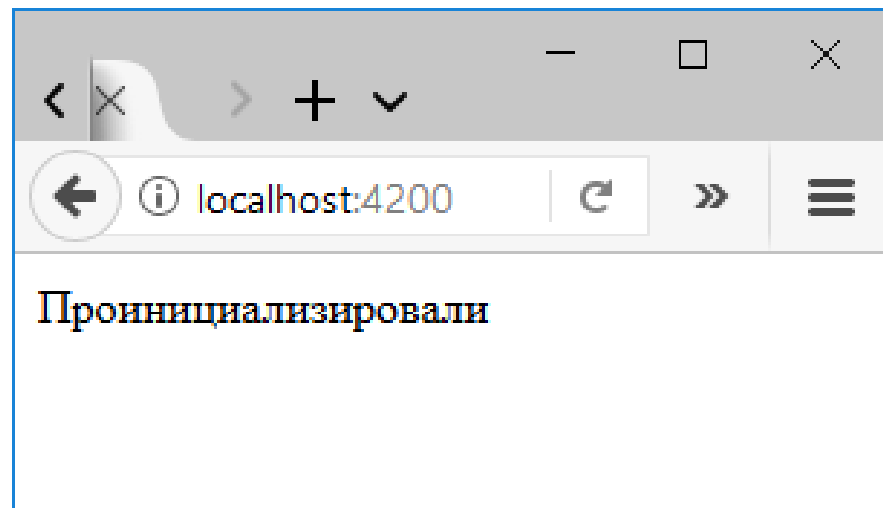
Пример ngOnInit()

app.component.ts

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

```
@Component ({  
  selector: 'app-root',  
  template: `  
    <div>  
      {{myValue}}  
    </div>  
  `,  
})
```

```
export class AppComponent {  
  myValue: string = "";  
  ngOnInit() {  
    this.myValue = "Проинициализировали"  
  }  
}
```



Обмен данными между компонентами, @Input, @Output

app.component.ts

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

```
@Component({
```

```
  selector: 'app-root',
```

```
  template: `
```

```
    <myCompB inB="{{ appValue }}" (onClick)="upCase($event)"></myCompB>
```

```
    <h3>{{ result }}</h3>
```

```
`
```

```
});
```

```
export class AppComponent {
```

```
  appValue = 'AppValue'
```

```
  result = ''
```

```
  upCase(eventVal: string): void {
```

```
    this.result = `App. Received '${eventVal}' from compB`
```

myCompB.component.ts

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

```
@Component({
```

```
  selector: 'myCompB',
```

```
  template: `
```

```
    <h3>myCompB received '{{inB}}' from App</h3>
```

```
    <button (click)="onClick.emit('Hi!')">Push me</button>
```

```
`
```

```
});
```

```
export class myCompB {
```

```
  @Input() inB: string = '';
```

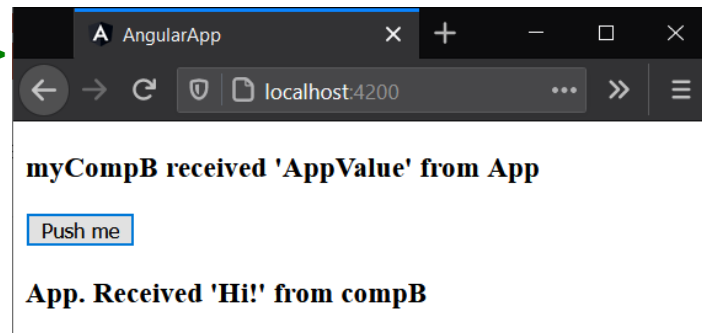
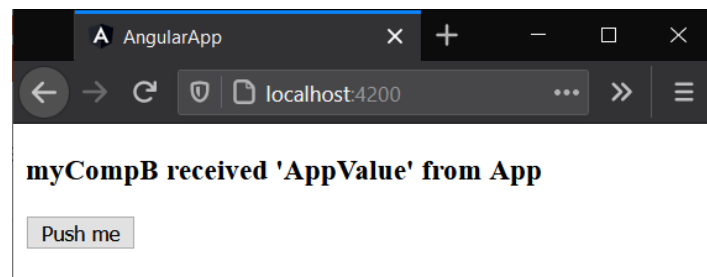
```
  @Output() onClick = new EventEmitter();
```

```
}
```

@Input – в дочерний из родительского

@Output – из дочернего в родительский

Из дочернего в родительский – события



Дочерние компоненты, <child-root>

54

child.component.ts

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

```
@Component ({
  selector: 'child-root',
  template: '<div> {{myValue}} </div> '
})
```

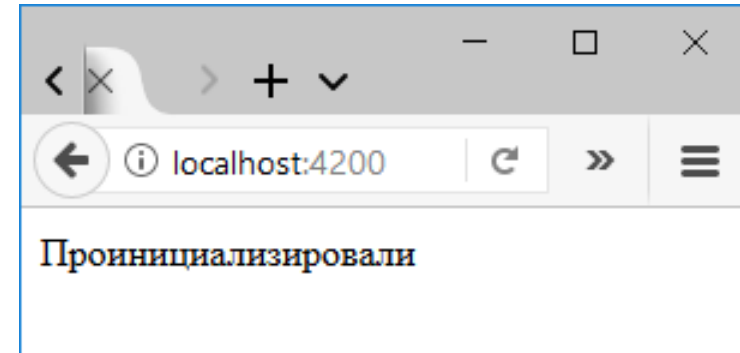
```
export class ChildComponent {
  myValue = '';
  ngOnInit() {
    this.myValue = "Проинициализировали"
  }
}
```

app.component.ts

```
import {Component} from '@angular/core';
import {ChildComponent}
  from './child.component';
```

```
@Component ({
  selector: 'app-root',
  template: `
    <child-root></child-root>
  `
})
```

```
export class AppComponent {}
```



app.module.ts

```
import { NgModule } from '@angular/core';
import { BrowserModule }
  from '@angular/platform-browser';
import { AppComponent }
  from './app.component';
import { ChildComponent }
  from './child.component';
```

```
@NgModule({
  declarations: [AppComponent,
                 ChildComponent],
  imports: [BrowserModule],
  providers: [],
  bootstrap: [AppComponent],
})
```

```
export class AppModule {}
```

Может
подключаться
как компонент
в другой
модуль

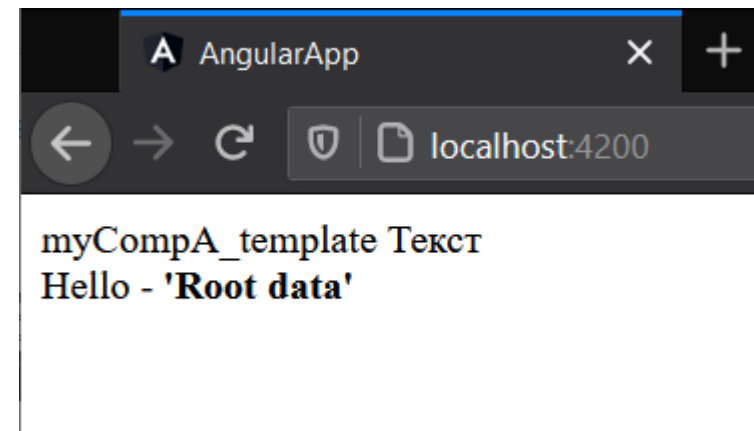
Передача данных потомку / <ng-content> 55

app.component.ts

```
import { Component } from '@angular/core';
@Component({
  selector: 'app-root',
  template: // Передача данных потомку
    `<comp-a>Hello - <b>{{data}}</b></comp-a>`
})
export class AppComponent {
  data:string = "Root data"
}
```

myCompA.component.ts

```
import { Component } from '@angular/core';
@Component({
  selector: 'comp-a',
  template: `
    myCompA_template {{mytext}}<br>
    <ng-content></ng-content>`, // Данные от родителя
})
export class myCompA {
  mytext: string = "Текст";
}
```



Вызов функций потомка / #имя

56

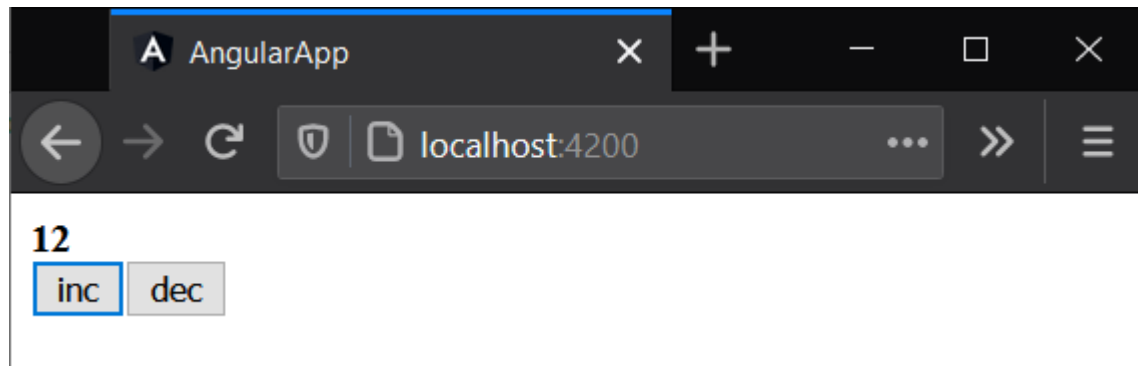
app.component.ts

```
import { Component } from '@angular/core';
@Component({
  selector: 'app-root',
  template: // Передача данных потомку
    `
```

- **compA** – экземпляр **myCompA** (я мог вместо **#compA** написать что угодно)
- **compA.inc()** – обращение к функции **inc()**
- **compA.dec()** – обращение к функции **dec()**

myCompA.component.ts

```
import { Component } from '@angular/core';
@Component({
  selector: 'comp-a',
  template: `<b>{{counter}}</b>`
})
export class myCompA {
  counter: number = 0;
  inc() { this.counter++; }
  dec() { this.counter--; }
}
```



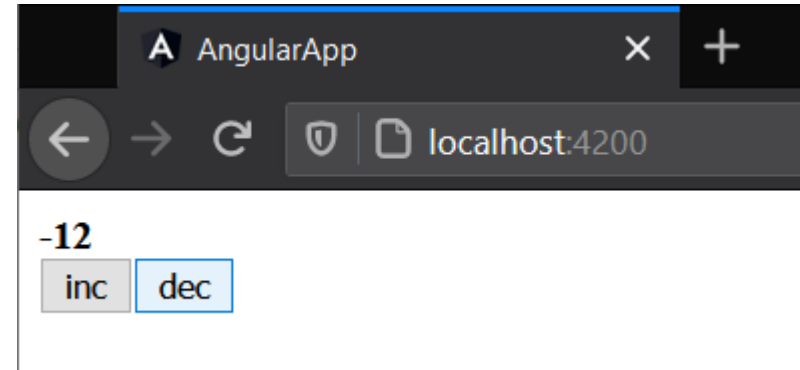
Вызов функций потомка / @ViewChild

57

app.component.ts

```
import { Component, ViewChild } from '@angular/core';
import { myCompA } from './myCompA.component';
@Component({
  selector: 'app-root',
  template:
    `</comp-a><br>
    <button (click)="incr()">inc</button>
    <button (click)="decr()">dec</button>`
})
export class AppComponent {
  // Наблюдатель за дочерним элементом
  @ViewChild(myCompA, {static: false})
  private compA: myCompA | undefined;
  incr() { this.compA?.incr(); }
  decr() { this.compA?.decr(); }
}
```

Здесь **static: true** –
динамическое
подключение, значение по
умолчанию – **false**



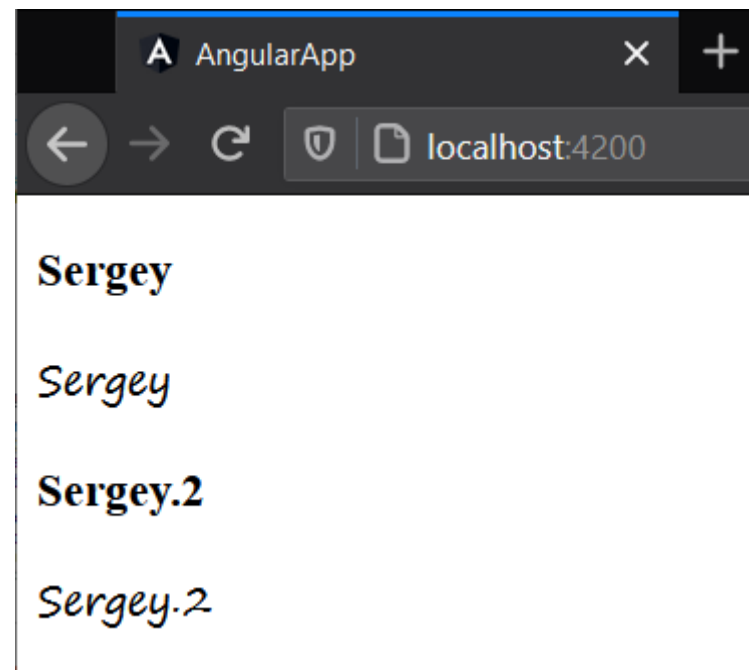
myCompA.component.ts

```
import { Component } from '@angular/core';
@Component({
  selector: 'comp-a',
  template: `<b>{{counter}}</b>`
})
export class myCompA {
  counter: number = 0;
  inc() { this.counter++; }
  decr() { this.counter--; }
}
```

Директива подключения классов [ngClass]

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

```
@Component({  
  selector: 'app-root',  
  template: // Статическое и динамическое подключение класса  
    `<h3 [ngClass]="{segoeFont:false}">{{name}}</h3>  
    <p [ngClass]="{segoeFont:isSegoe}">{{name}}</p>  
    <h3 [class.segoeFont]="false">{{name}}.2</h3>  
    <p [class.segoeFont]=[isSegoe]>{{name}}.2</p>  
    `,  
  styles: [.segoeFont{font-family:"Segoe Print";}]  
})  
export class AppComponent {  
  name: string = "Sergey";  
  isSegoe: boolean = true  
}
```



Директива подключения стилей [ngStyle]

```
import { Component } from '@angular/core';
@Component({
  selector: 'app-root',
  template: // Статическое и динамическое подключение класса
    `

### 

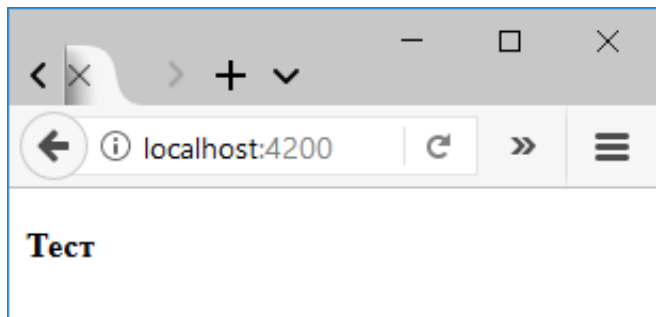
A screenshot of a web browser window. The browser has a single tab titled "AngularApp". The address bar shows "localhost:4200". The page content displays three lines of text: "Sergey" in a decorative font, "Sergey.2" in a similar font, and "Sergey.3" in a larger, bold, decorative font.
```

Атрибутные директивы @Directive

60

app.component.ts

```
import { Component } from '@angular/core';
@Component({
  selector: 'app-root', // Имя в index.html
  template: `<p bold> Тест </p>`
})
export class AppComponent {
}
```



app.module.ts

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

highlight.directive.ts

```
import { Directive, ElementRef } from '@angular/core';
@Directive({
  selector: '[bold]'
})
export class BoldDirective {
  constructor(private elementRef: ElementRef) {

    this.elementRef.nativeElement.style.fontWeight
    = "bold";
  }
}
```

@Directive позволяет задать
свой вариант поведения
элемента HTML

Взаимодействие с пользователем

@HostListener

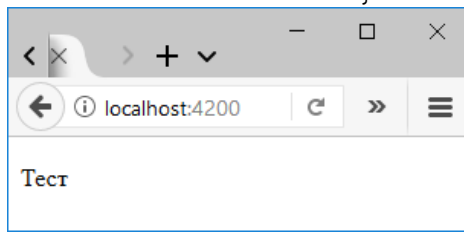
app.component.ts

```
import { Component } from '@angular/core';
@Component({
  selector: 'app-root',
  template: `<p appHighlight> Тест </p>`
})
export class AppComponent {
}
```

app.module.ts

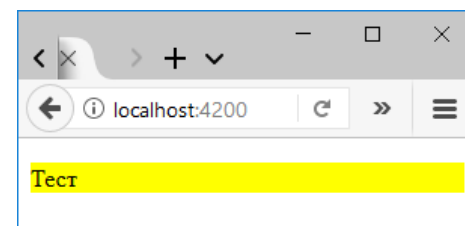
```
import { NgModule } from '@angular/core';
import { BrowserModule }
from '@angular/platform-browser';
import { AppComponent }
from './app.component';
import { HighlightDirective }
from './highlight.directive'
```

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



highlight.directive.ts

```
import { Directive,
  ElementRef, HostListener }
from '@angular/core';
@Directive({
  selector: '[appHighlight]'
})
export class HighlightDirective {
  constructor(private el: ElementRef) {
  }
  @HostListener('mouseenter')
  onMouseEnter() {
    this.highlight('yellow');
  }
  @HostListener('mouseleave')
  onMouseLeave() {
    this.highlight(null);
  }
  private highlight(color: string) {
    this.el.nativeElement.style
      .backgroundColor = color;
  }
}
```

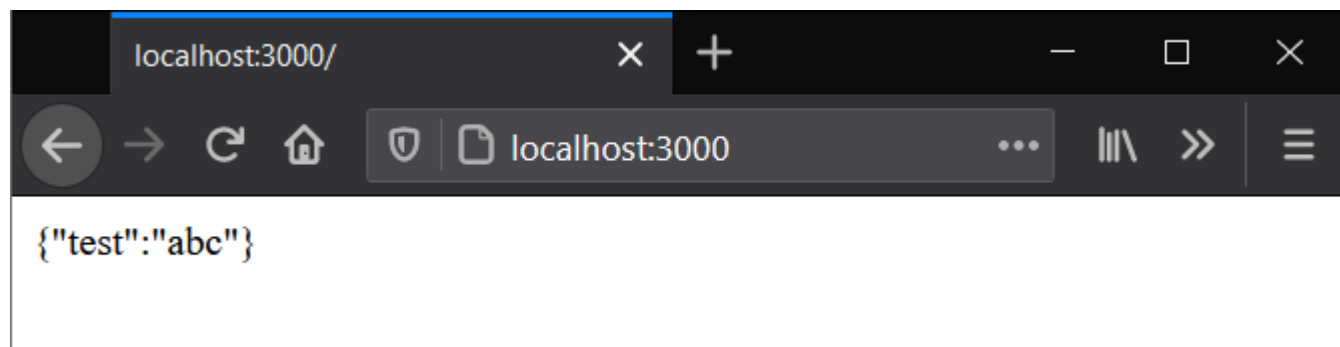


Взаимодействие с сервером (1), localhost:3000, CORS

app.js

```
let express = require("express");
let server = express();
server.get("/", (req, res)=>{
  res.header("Access-Control-Allow-Origin", "*");
  res.header("Access-Control-Allow-Headers", "Content-Type");
  res.header('Access-Control-Allow-Methods', 'GET');
  res.send(JSON.stringify({test:"abc"}));
});
server.listen(3000);
```

Можно передавать сразу
json, тогда получатель
получит объект



Взаимодействие с сервером (2), HttpClientModule

app.module.ts

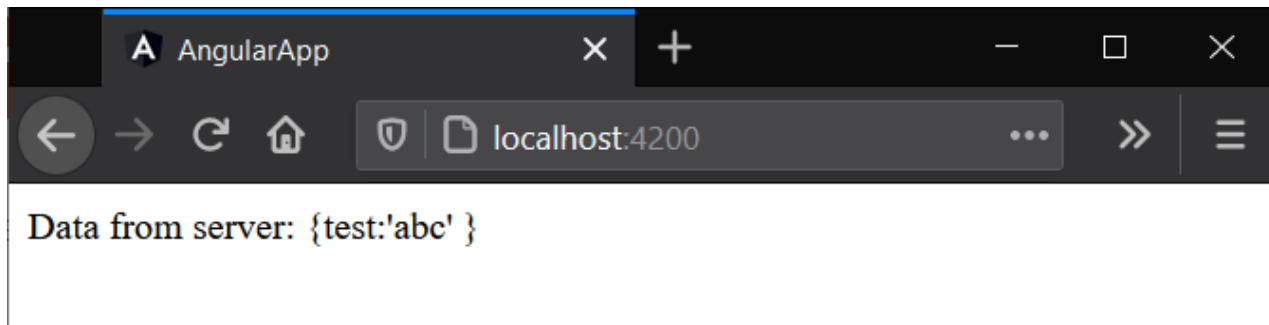
```
import { NgModule } from '@angular/core';  
import { BrowserModule } from '@angular/platform-browser';  
import { FormsModule } from '@angular/forms';  
import { HttpClientModule } from '@angular/common/http';  
import { AppComponent } from './app.component';
```

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

Взаимодействие с сервером (3), CORS, get, subscribe, <any>

app.component.ts

```
import { Component } from '@angular/core';
import { HttpClient, HttpHeaders } from '@angular/common/http';
@Component({
  selector: 'app-root',
  template: `Data from server: {{{"test":'{{appValue}}' }}}`,
  providers: []
})
export class AppComponent {
  appValue:any = 'Edit it'
  constructor(private http: HttpClient) {}
  ngOnInit(): void {
    const headers = new HttpHeaders();
    this.http.get<any>("http://localhost:3000", {headers: headers})
      .subscribe(value=>{
        this.appValue = value.test;
      },
      error=> {
        console.log(error)
      })
  }
}
```

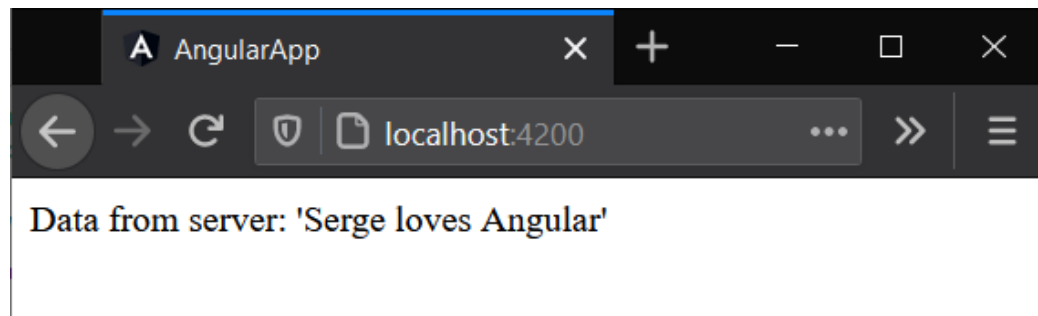


Передача параметров на сервер / GET

65

фрагмент app.component.ts

```
export class AppComponent {  
  appValue:any = 'Edit it'  
  constructor(private http: HttpClient) {}  
  ngOnInit(): void {  
    const params = new HttpParams()  
      .set('p1', "Serge")  
      .set('p2', "Angular");  
    this.http.get<any>("http://localhost:3000", {params})  
      .subscribe(value=>{  
        this.appValue = value.myresult;  
      },  
      error=> {  
        console.log(error)  
      })  
  }  
}
```



app.js

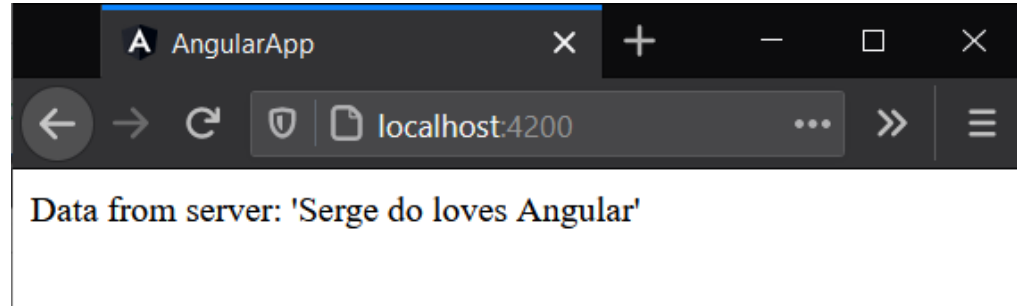
```
let express = require("express");  
let server = express();  
server.get("/", (req, res)=>{  
  const p1 = req.query.p1; // Получение параметра запроса p1  
  const p2 = req.query["p2"]; // Получение параметра запроса p2  
  res.header("Access-Control-Allow-Origin", "*");  
  res.header("Access-Control-Allow-Headers", "Content-Type");  
  res.header('Access-Control-Allow-Methods', 'GET');  
  res.send({"myresult": `${p1} loves ${p2}`});  
});  
server.listen(3000);
```

Работа с POST

66

фрагмент app.component.ts

```
export class AppComponent {  
  appValue:any = 'Edit it'  
  constructor(private http: HttpClient) {}  
  ngOnInit(): void {  
    const body = {p1: "Serge do", p2: "Angular"}  
    this.http.post<any>("http://localhost:3000", body)  
      .subscribe(value=>{  
        this.appValue = value.myresult;  
      },  
      error=> {  
        console.log(error)  
      })  
  }  
}
```



app.js

```
let express = require("express");  
let server = express();  
// создаем парсер для данных в формате json  
const bodyParser = express.json();  
server.use((req, res, next)=>{  
  res.header("Access-Control-Allow-Origin", "*");  
  res.header("Access-Control-Allow-Headers", "Content-Type");  
  res.header('Access-Control-Allow-Methods', 'GET, POST');  
  next();  
});  
server.post("/", bodyParser, (req, res)=>{  
  const p1 = req.body.p1; // Получение параметра запроса p1  
  const p2 = req.body["p2"]; // Получение параметра запроса p2  
  res.send({myresult: `${p1} loves ${p2}`});  
});  
server.listen(3000);
```

Замена реального сервиса на поддельный, useFactory, providers, MyMockService

app.component.ts

```
import { Component } from '@angular/core';
import { MyMockService } from "./mymock.service";
import { MyHttpService } from "./http.service";
import { HttpClient } from "@angular/common/http";

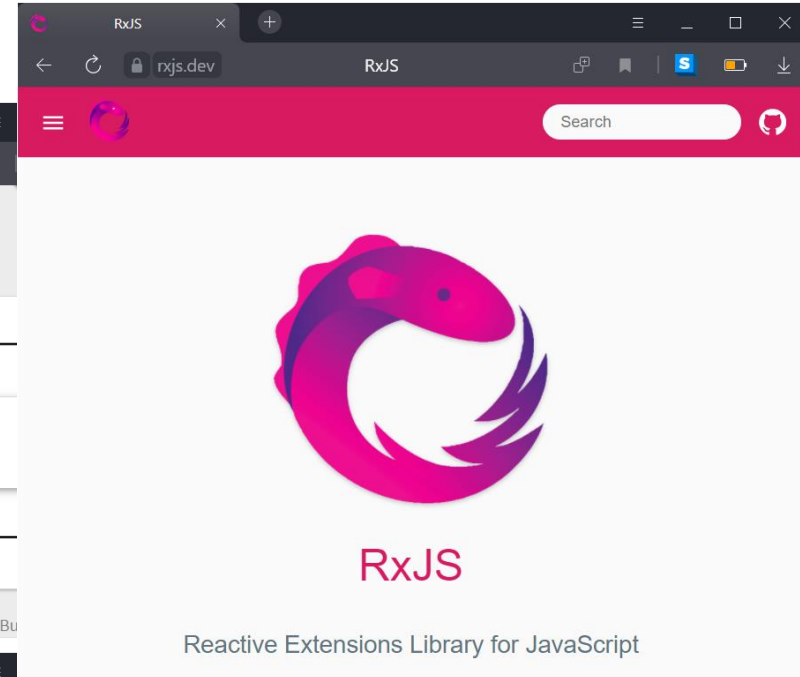
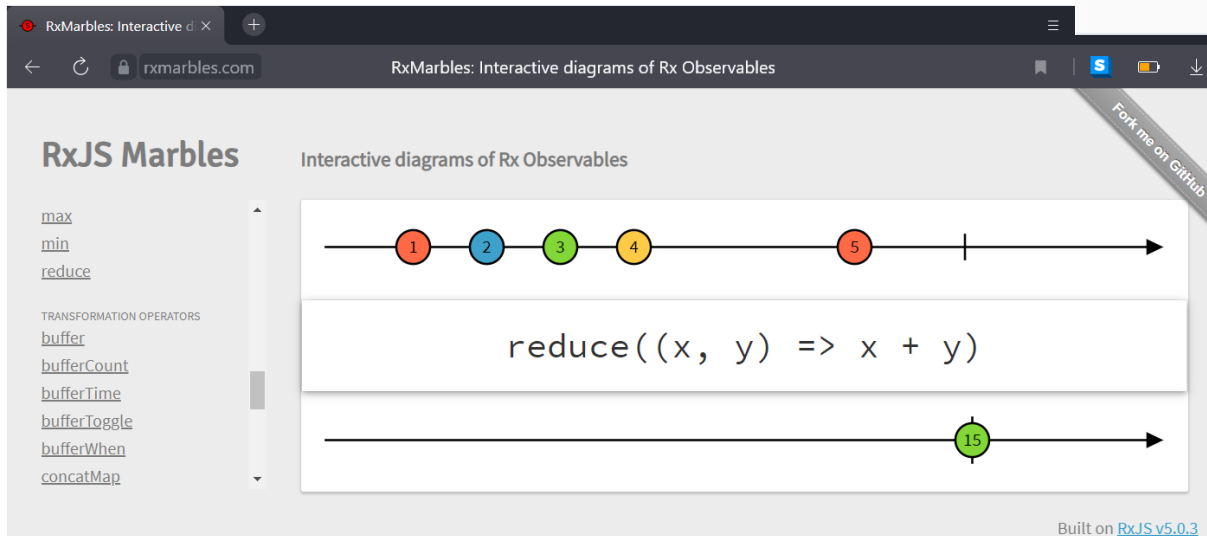
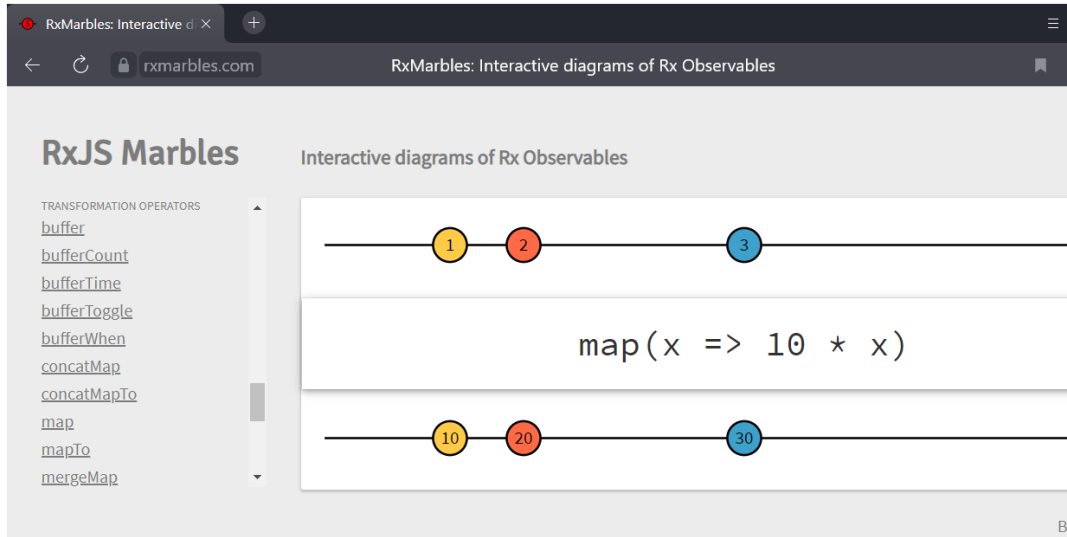
@Component({
  selector: 'app-root',
  template: `Data from server: {{appValue}}`,
  providers: [ // Провайдеры
    {provide: HttpClient, useClass: HttpClient},
    { provide: 'IS_DEV', useValue: true }, // Подделка
    {
      provide: MyHttpService, // Реальный сервис
      useFactory: (IS_DEV:boolean, httpClient:HttpClient) => {
        if(IS_DEV) // Если подделка - вернуть подделку
          return new MyMockService()
        return new MyHttpService(httpClient)
      },
      deps: ["IS_DEV", HttpClient] // Нужны два параметра
    }
  ])
export class AppComponent {
  appValue: any = 'Edit it';
  constructor(private http: HttpClient, private service: MyHttpService) {
    this.appValue = service.get() // Получение значения
    // Важно! Что в этом примере нет ожидания на ответ сервера!
  }
}
```

mymock.service.ts

```
import { Injectable } from "@angular/core";
@Injectable()
export class MyMockService {
  data: any = "No";
  constructor() {}
  public get(): any {
    return this.data;
  }
}
```

В MyHttpService при этом перенесено соединение с сервером и получение данных **data** (как в предыдущем примере)

Observable<any> из библиотеки RxJS 68



- **Наблюдаемый RxJS** – два метода:
 - подписаться и**
 - отказаться от подписки**
- **Наблюдатель** – объект с `next()`, `error()` и `complete()`
- **Подписка**
- **Оператор** – функция
- **Предмет** – `EventEmitter`
- **Планировщик**

<https://rxmarbles.com/>

<https://rxjs.dev/>

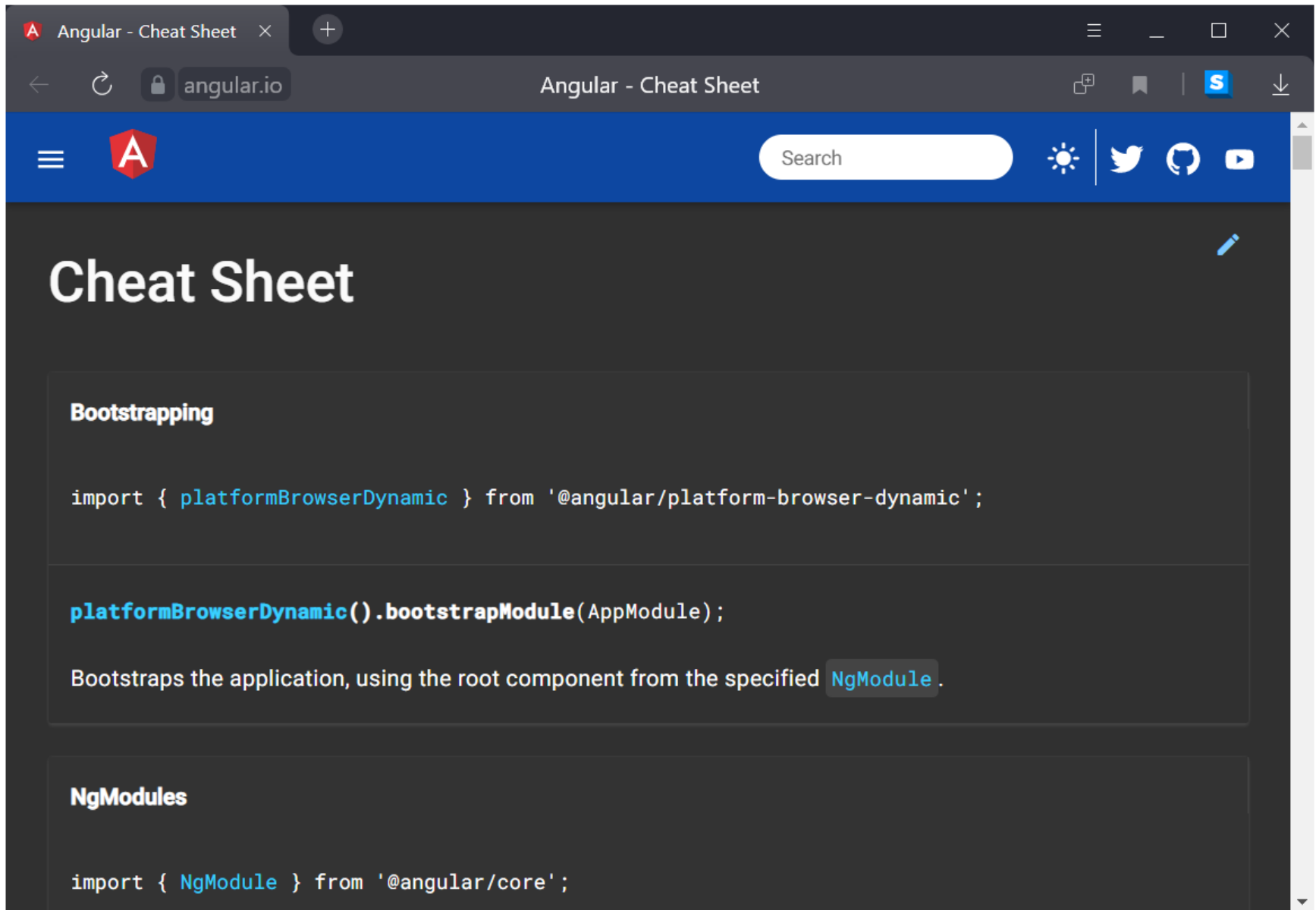
<https://angdev.ru/doc/angular-and-rxjs/>

Обработка данных с использованием Observable ⁶⁹

```
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';
import { Data } from './data';
import { Observable } from 'rxjs';
import { map } from 'rxjs/operators';
@Injectable()
export class HttpService{
  constructor(private http: HttpClient){ }
  getUsers() : Observable<Data[]> {
    return this.http.get('assets/users.json').pipe(map((data: any)=>{
      let dataObjList = data["someList"];
      return dataObjList.map(function(dataObject: any): Data {
        return new Data(dataObject.prop1, dataObject.prop2);
      });
    }));
  }
}
```

Справка по Angular

70



The screenshot shows a web browser window with the URL `angular.io` and the page title "Angular - Cheat Sheet". The page features a dark blue header with the Angular logo, a search bar, and social media icons. The main content area is dark gray and titled "Cheat Sheet". It contains two sections: "Bootstrapping" and "NgModules".

Bootstrapping

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

`platformBrowserDynamic().bootstrapModule(AppModule);`

Bootstraps the application, using the root component from the specified `NgModule`.

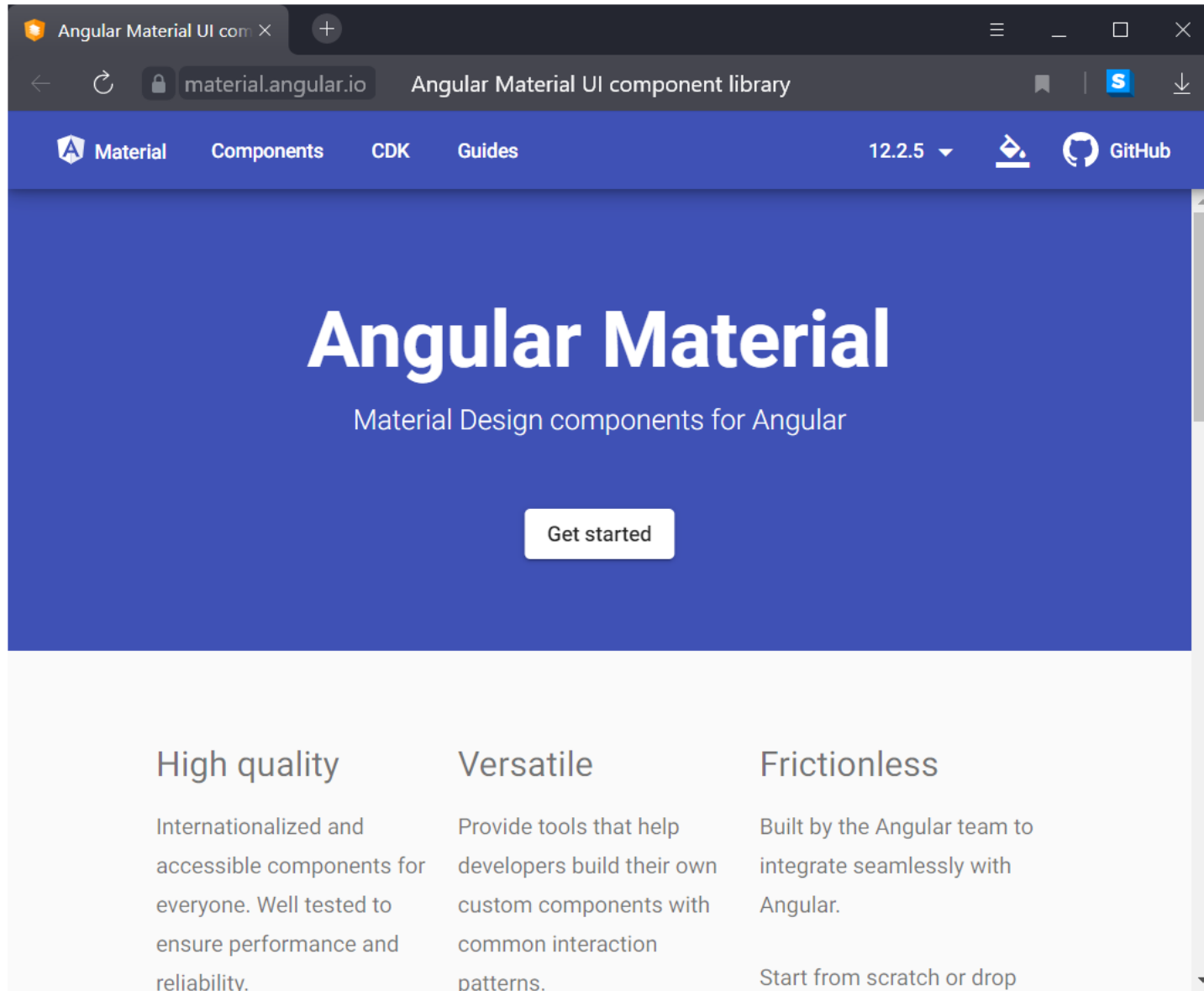
NgModules

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

<https://angular.io/guide/cheatsheet>

Angular Material

71



<https://material.angular.io/>

Компоненты Angular Material

72

Autocomplete

Badge

Bottom Sheet

Button

Button toggle

Card

Checkbox

Chips

Core

Datepicker

Dialog

Divider

Expansion
Panel

Form field

Grid list

Icon

Input

List

Menu

Paginator

Progress bar

Progress
spinner

Radio button

Ripples

Select

Sidenav

Slide toggle

Slider

Snackbar

Sort header

Stepper

Table

Tabs

Toolbar

Tooltip

Tree

<https://material.angular.io/components/>

E2E фреймворк тестирования Angular

73



<https://protractor.angular.io/>

Вопросы для самопроверки

74

- Из чего состоит Angular? Какими директивами это описывается?
- Какой минимальный вариант приложения мы рассмотрели (по составу)?
- Какие свойства @NgModule/@Component Вы знаете и для чего они нужны?
- Что такое шаблон компонента? Как его можно описать? Подключить?
- Какие встроенные директивы Вы знаете? Как их использовать?
- Что такое привязка (binding)? Какие бывают? Как их использовать?
- Как организовывается маршрутизация (routing)?
- Как передать параметры при маршрутизации?
- Что такое шаблонная переменная? Мы можем обратиться к дочернему компоненту?
- Как реализуется валидация данных?
- Что такое сервисы? Что такое DI? Что такое иерархия сервисов?
- Что такое параметризация вывода?
- Какой жизненный цикл у приложения Angular?
- Как осуществляется взаимодействие между компонентами?
- Какие директивы Вы знаете? Можно ли создать новые?
- Как взаимодействовать с сервером? Как передать/получить данные?
- Что такое CORS?
- Для чего нужна библиотека RxJS?
- Для чего нужна Angular Material?
- Где научиться делать e2e-тесты для Angular?