# Angular

Материал взят с сайтов <https://angular.io/> и <https://metanit.com/web/angular2/>

Angular представляет фреймворк от компании Google для создания клиентских приложений. Прежде всего он нацелен на разработку SPA-решений (Single Page Application), то есть одностраничных приложений.

Angular предоставляет такую функциональность, как двустороннее связывание, позволяющее динамически изменять данные в одном месте интерфейса при изменении данных модели в другом, шаблоны, маршрутизация и так далее.

Одной из ключевых особенностей Angular является то, что он использует в качестве языка программирования TypeScript.

Официальный репозиторий фреймворка на гитхабе: https://github.com/angular/angular. Там можно найти сами исходные файлы, а также некоторую дополнительную информацию.

# Первое приложение на Angular

Для работы с Angular необходимо установить сервер Node.js и пакетный менеджер npm, если они отсутствуют на рабочей машине. Для установки можно использовать программу установки node.js. Вместе с сервером она также установит и npm.

Для создания проектов, создания кода приложения и библиотеки, а также для выполнения различных текущих задач разработки, таких как тестирование, пакетирование и развертывание используется Angular CLI.

Для установки Angular CLI необходимо выполнить команду:

npm install -g @angular/cli

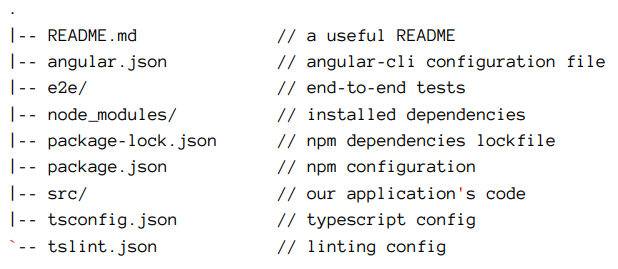
Для проверки установки можно воспользоваться командой:

ng --version

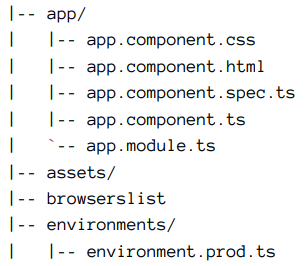
Для создания приложения воспользуемся командой:

ng new angular-hello-world

После создания приложения в папке проекта будет создана следующая структура



В папке src находятся все файлы с кодом нашего приложения





В файле index.html включен компонент Angular:

<!doctype html>

<html lang="en">

<head>

  <meta charset="utf-8">

  <title>AngularHelloWorld</title>

  <base href="/">

  <meta name="viewport" content="width=device-width, initial-scale=1">

  <link rel="icon" type="image/x-icon" href="favicon.ico">

</head>

<body>

  <!--Component Angular-->

  <app-root></app-root>

</body>

</html>

Для запуска приложения необходимо запустить web-server. Для этого можно воспользоваться командой:

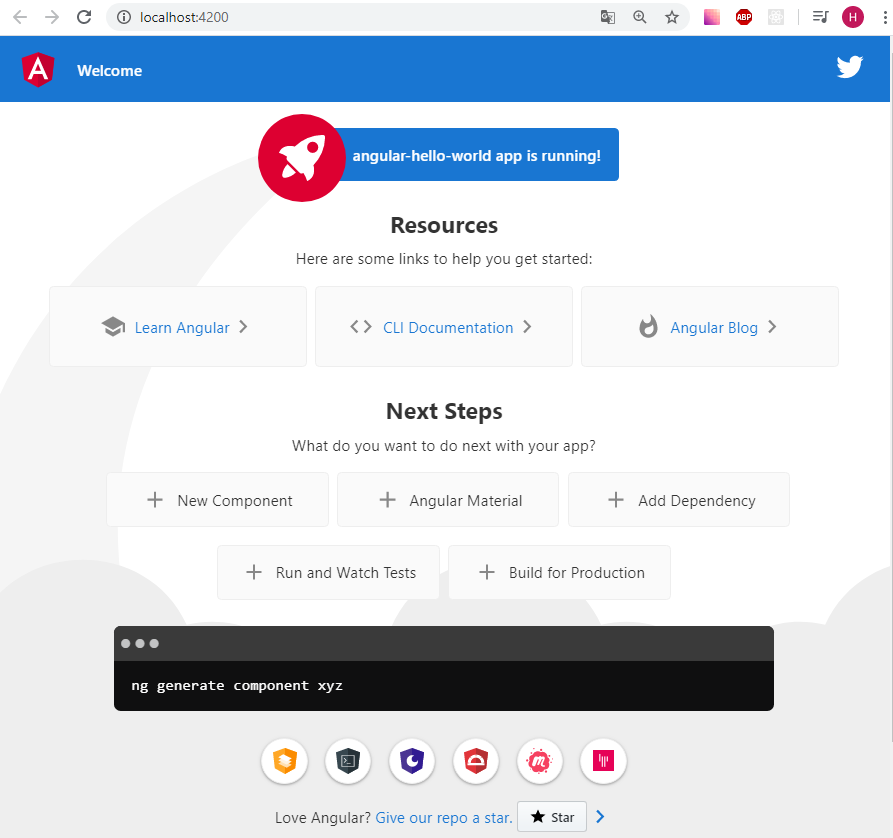
cd angular-hello-world

ng serve --open

Команда ng serve запускает сервер, просматривает файлы и перестраивает приложение, если вносятся изменения в эти файлы.

Опция --open (или просто -o) автоматически открывает браузер по адресу http: // localhost: 4200 /.

Стандартное приложение, созданное таким образом, выглядит примерно так:



# Основы Angular

# Компоненты

Одним из ключевых элементов приложения являются компоненты. Компонент управляет отображением представления на экране.

При создании приложения был определен следующий компонент:

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

@Component({

  selector: 'app-root',

  templateUrl: './app.component.html',

  styleUrls: ['./app.component.css']

})

export class AppComponent {

  title = 'angular-app';

}

Сам класс компонента здесь относительно небольшой. Чтобы класс мог использоваться в других модулях, он определяется с ключевым словом export. В самом же классе определена лишь одна переменная, которая в качестве значения хранит некоторую строку.

Для создания компонента необходимо импортировать функцию декоратора @Component из библиотеки @angular/core. Декоратор @Component позволяет идентифицировать класс как компонент.

Если бы мы не применили декоратор @Component к классу AppComponent, то класс AppComponent компонентом бы не считался.

Декоратор @Component в качестве параметра принимает объект с конфигурацией, которая указывает фреймворку, как работать с компонентом и его представлением с помощью свойства template. Template представляет HTML разметку со вставками кода Angular. Фактически template это и есть представление, которое увидит пользователь при работе с приложением.

Каждый компонент должен иметь один шаблон. Однако необязательно определять шаблон напрямую с помощью свойства template. Можно вынести шаблон во внешний файл с разметкой html, а для его подключения использовать свойство templateUrl как в нашем примере.

Шаблон может быть однострочным или многострочным. Если шаблон многострочный, то он заключается в косые кавычки (`), которые стоит отличать от стандартных ординарных кавычек (').

Также в примере выше устанавливается свойство selector, которое определяет селектор CSS. В элемент с этим селектором Angular будет добавлять представление компонента. Например, в примере выше селектор имеет значение app-root. Соответственно если html-страница содержит элемент <app-root></app-root>, то именно этот элемент будет использоваться для рендеринга представления компонента.

# Модули

Приложение Angular состоит из отдельных модулей. Как правило, приложения состоят из нескольких модулей. И каждое приложение Angular как минимум имеет один корневой модуль (root module), который, согласно условностям, называется AppModule. Например, в нашем приложении использовался следующий корневой модуль:

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 { }

Для работы модуля ему необходимы ряд библиотек, поэтому в начале файла идет их подключение. Имя каждой библиотеки Angular начинается с префикса @angular.

Библиотеки устанавливаются через пакетный менеджер npm и импортируются с помощью директивы import. Например, импорт функциональности декоратора NgModule из библиотеки @angular/core.

Для модуля AppModule необходимы:

* **NgModule**: функциональность декоратора NgModule, без которой мы не сможем создать модуль
* **BrowserModule**: модуль, необходимый для работы с браузером
* **AppComponent**: функциональность корневого компонента приложения

Непосредственно сам модуль представлен классом AppModule, который на первый взгляд ничего не делает и не содержит никакого функционала. Однако в Angular модуль это не просто класс. Каждый модуль должен определяться с декоратором @NgModule.

NgModule представляет функцию-декоратора, которая принимает объект, свойства которого описывают метаданные модуля. Наиболее важные свойства:

* **declarations**: классы представлений (view classes), которые принадлежат модулю. Angular имеет три типа классов представлений: компоненты (components), директивы (directives), каналы (pipes)
* **exports**: набор классов представлений, которые должны использоваться в шаблонах компонентов из других модулей
* **imports**: другие модули, классы которых необходимы для шаблонов компонентов из текущего модуля
* **providers**: классы, создающие сервисы, используемые модулем
* **bootstrap**: корневой компонент, который вызывается по умолчанию при загрузке приложения

В случае выше единственным классом представлений является компонент AppComponent. Поэтому он указывается для свойств declarations и bootstrap. И поскольку его действие зависит от модуля BrowserModule, то данный модули указывается для свойства imports.

Но при необходимости мы могли бы использовать и другие свойства.

# Загрузка приложения

При запуске приложения первым выполняется код, который определен в файле main.ts. Этот файл имеет следующее содержимое:

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

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

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

import { environment } from './environments/environment';

if (environment.production) {

  enableProdMode();

}

platformBrowserDynamic().bootstrapModule(AppModule)

  .catch(err => console.error(err));

Последняя строка импортирует функциональность модуля platformBrowserDynamic из пакета angular/platform-browser-dynamic. platformBrowserDynamic использует bootstrapModule для загрузки нужного модуля.

То есть фактически здесь platformBrowserDynamic запускает импортированный модуль AppModule. После этого начинает работать вся логика, которая заложена в модуле AppModule, который представляет главный модуль приложения. Больше ничего файл main.ts не делает.

В самом же AppModule нам надо указать, какой именно компонент будет использоваться в качестве основного при загрузке. Для этого используется параметр bootstrap декоратора NgModule.

# Стили и шаблоны компонента

Стилизация компонента может производиться как с помощью установки стилей в самом компоненте, так и с помощью подключения внешних css-файлов.

Для установки стилей в директиве @Component определено свойство styles:

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

@Component({

  selector: 'app-root',

  templateUrl: './app.component.html',

  //styleUrls: ['./app.component.css']

  styles: [`

      h1 {

        font-size: 3em;

        color: red;

        text-align: center;

    }

  `]

})

export class AppComponent {

  title = 'angular-app';

}

Параметр styles содержит набор стилей, которые будут использоваться компонентом.

При использовании стилей следует учитывать, что они применяются локально только к разметке, управляемой компонентом. Например, если на странице будут элементы вне области управления компонентом, то к ним уже не будут применяться стили.

**Селектор :host** ссылается на элемент, в котором хостится компонент. То есть в данном случае это элемент <app-root></app-root>. И селектор :host как раз позволяет применить стили к этому элементу:

 styles: [`

      h1 {

        font-size: 3em;

        color: red;

        text-align: center;

    }

    :host {

      font-family: Verdana;

      background-color: #555;

    }

  `]

})

Если стилей много, то код компонента становится слишком громоздким, и в этом случае предпочтительно вынести стили в отдельный файл css.

Для загрузки стилей используется параметр styleUrls, который позволяет указать набор файлов css, которые применяются для стилизации. В данном случае предполагается, что файл css располагается в проекте в папке app.

Похожим образом мы можем вынести шаблон в отдельный файл html. Также в папке app создадим новый файл app.component.html со следующим кодом:

@Component({

  selector: 'app-root',

  templateUrl: './app.component.html',

  styleUrls: ['./app.component.css']

})

За счет выноса кода css и html код самого компонента становится чище и проще. Однако если мы производим сборку приложения через webpack, то мы столкнемся с проблемами. Нам надо указать сборщику, как загружать файлы html и css. И для этого можно воспользоваться загрузчиками html-loader и raw-loader. В частности, в начале необходимо загрузить пакеты загрузчиков в проект через npm. И затем загрузчики raw-loader и html-loader надо применить в конфигурации webpack:

module: {

        rules: [   //загрузчик для ts

            {

                test: /\.ts$/, // определяем тип файлов

                use: [

                    {

                        loader: 'awesome-typescript-loader',

                        options: { configFileName: path.resolve(\_\_dirname, 'tsconfig.json') }

                    },

                    'angular2-template-loader'

                ]

            },

            {

                test: /\.html$/,

                loader: 'html-loader'

            },

            {

                test: /\.css$/,

                include: path.resolve(\_\_dirname, 'src/app'),

                loader: 'raw-loader'

            }

        ]

    },

# Привязка данных

Angular поддерживает механизм привязки, благодаря которому различные части шаблона могут быть привязаны к некоторым значениям, определенным в компоненте.

В Angular есть четыре формы привязки данных:

* Привязка элемента DOM к значению компонента (односторонняя). В двойных фигурных скобках указывается выражение, к которому идет привязка: {{выражение}}.

Например: <h1>Добро пожаловать {{name}}!</h1>

* Привязка свойства элемента DOM к значению компонента (односторонняя). Например:

<input type="text" [value]="name" />

* Привязка метода компонента к событию в DOM (генерация события в DOM вызывает метод на компоненте) (односторонняя). Например:

<button (click)="addItem(text, price)">Добавить</button>

* Двусторонняя привязка, когда элемент DOM привязан к значению на компоненте, при этом изменения на одном конце привязки сразу приводят к изменениям на другом конце. Например:

<input [(ngModel)]="name" placeholder="name">

* Привязка к атрибуту элемента html
* Привязка к классу CSS
* Привязка к атрибуту элемента html

Первый вид привязки заключается в использовании фигурных скобок, в которые передается значение из компонента.

Второй вид привязки. Мы можем привязать значение к свойству элемента html. В этом случае свойство указывается в квадратных скобках. Важно понимать, что здесь идет привязка не к атрибуту, а именно к свойству элемента в javascript, который представляет данный элемент html. То есть html-элемент <input> в javascript представлен интерфейсом HTMLInputElement, у которого есть свойство value.

Иногда возникает необходимость выполнить привязку не к свойству, а именно к атрибуту html-элемента. Хотя свойства и атрибуты html-элементов могут пересекаться, как это было показано выше в примере с свойством/атрибутом value, но такое соответствие бывает не всегда. В этом случае мы можем использовать выражение:

[attr.название\_атрибута]="значение"

Привязка стилей имеет следующий синтаксис:

[style.стилевое\_свойство]="выражение ? A : B"

После префикса style через точку идет название свойства стиля. В качестве значения передается некоторое выражение: если оно возвращает true, то стилевому свойству присваивается значение A; если оно возвращает false, то стилевому свойству присваивается значение B.

Для демонстрации привязки изменим структуру компонента:

export class AppComponent {

  title = 'angular-app';

  name = 'nick';

  age = 25;

  src = "http://gomel.itstep.by/wp-content/uploads/2015/01/-e1560346035318.jpg";

  isRed = false;

  changeTitle() {

    this.title = "Changed Title";

  }

}

А также содержимое app.component.html, используемого для представления компонента:

<h1>{{ title }} is running!</h1>

<p>2 + 2 = {{2 + 2}}</p>

<p [style.color]="isRed ? 'red': 'blue'">name - {{ name }}, age - {{ age }}</p>

<input type="text" [(ngModel)]="name">

<p>

  <button (click)="changeTitle()">Изменить заголовок</button>

</p>

<picture>

  <img src="" alt="" [attr.src]="src">

</picture>

В итоге должно получиться следующее:

