

Action Script

Delphi

C

#

PHP

Angular

React

Node.js

Python

Java

JavaScript

jQuery

AJAX

Ruby

Perl

C

++

PHP

Python

Perl

Delphi

Visual Basic

Clipper

Basic

VBScript

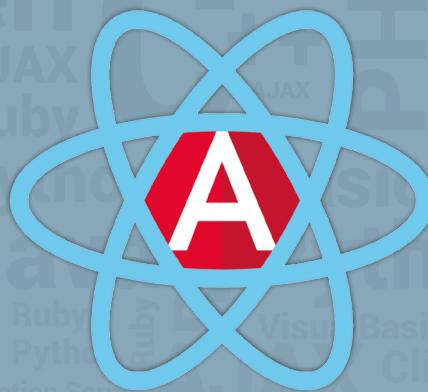
ASP

CGI

CGI

Создание веб-приложений  
с использованием

**Angular & React**



# Unit 3

Angular: основы

## Contents

Что такое Angular .....	3
Настройка окружения .....	7
Структура Angular приложения .....	12
Angular модули.....	15
Angular компоненты .....	18
Интерполяция строк.....	23
Создание нового компонента.....	25

# Что такое Angular

**Angular** — это открытый фреймворк, который позволяет разрабатывать приложения под несколько платформ: веб, мобильные устройства и десктоп.

**Фреймворк** — это своеобразный каркас, используемый для того, чтобы существенно облегчить разработку и объединение разных компонентов большого программного проекта.

Фреймворки предоставляют четкую структуру приложения и реализуются с использованием «паттернов проектирования». Наличие структуры подразумевает модульность приложения, а это дает возможность проще работать над приложением нескольким разработчикам одновременно. При использовании фреймворка кода становится заметно меньше, и он чище, что позитивно отражается на скорости разработки, а также поддержке и устранении ошибок в коде приложения.

## Цели и задачи Angular

Прежде всего Angular нацелен на разработку SPA-решений (*Single Page Application*), то есть одностраничных приложений.

Он упрощает создание пользовательских компонентов, которые могут быть добавлены в документы HTML, а также реализацию логики приложения. Активно использует привязку данных, содержит модуль внедрения зависимостей, поддерживает модульность и предоставляет механизм для настройки маршрутизации.

## Понятие одностраничного приложения (SPA)

Существует два основных подхода к созданию веб-приложений: традиционные многостраничные веб-приложения (*Multi Page Application* — MPA) и одностраничные приложения (*Single Page Application* — SPA).

В многостраничном веб-приложении каждый раз, когда приложение вызывает сервер, сервер отображает новую HTML-страницу. Это вызывает обновление страницы в браузере.

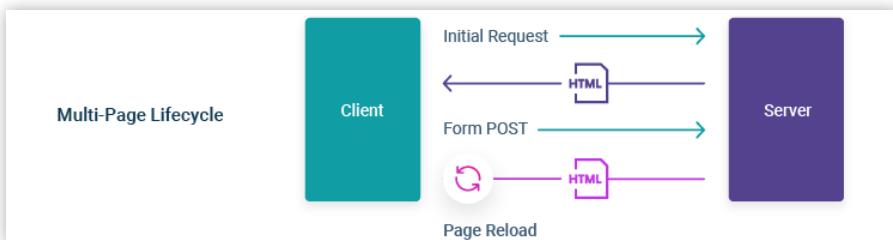


Рисунок 1

В одностраничном веб-приложении после загрузки первой страницы все взаимодействие с сервером происходит через вызовы AJAX. Эти AJAX вызовы возвращают данные, а не разметку, обычно в формате JSON. Приложение использует данные JSON для динамического обновления страницы без ее перезагрузки.



Рисунок 2

Скорее всего, вы уже использовали одностраничное приложение. Например, когда вы прокручивали свою новостную ленту в Facebook, динамически подгружались новые посты.

В МРА большая часть логики, включая формирование представления, выполняется на сервере, а в SPA, логика пользовательского интерфейса максимально перенесена на сторону клиента. Все элементы управления обрабатываются и отрисовываются с помощью JavaScript. То есть, сервер отдает только данные, как правило в формате JSON, а сторона клиента самостоятельно формирует страницу сайта, все шаблоны, списки, ссылки, таблицы и прочие обновляемые элементы.

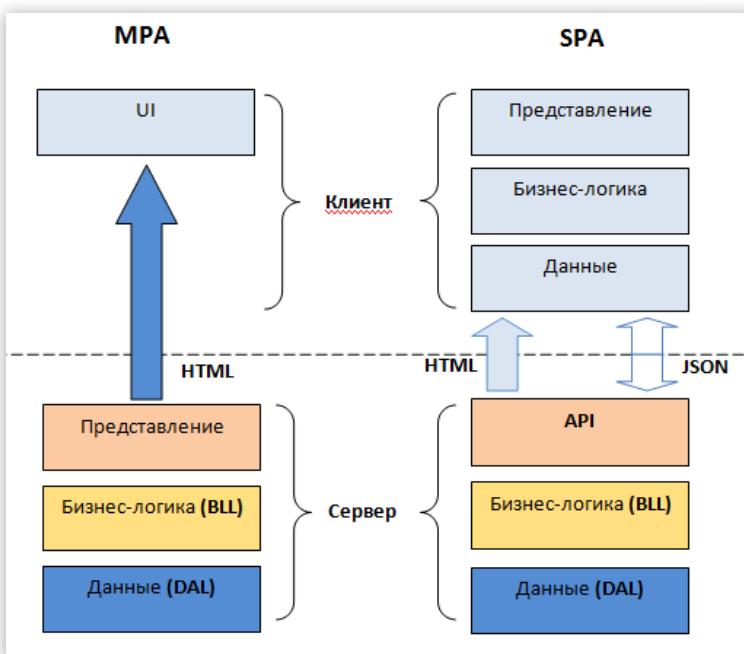


Рисунок 3

Одностраничные приложения быстрее загружаются, осуществляя выборку данных в фоновом режиме. Так как большинство ресурсов (например, HTML, CSS, скрипты) загружаются только один раз за время существования приложения, скорость реагирования на действия пользователя возрастает.

Также SPA повышают скорость разработки. Готовые библиотеки и фреймворки дают мощные инструменты для разработки веб приложений. Над проектом могут параллельно работать back-end и front-end разработчики. Благодаря четкому разделению они не будут мешать друг другу.

Однако, при всех своих достоинствах, Single Page Application имеет некоторые недостатки:

- **Плохая SEO оптимизация.** SPA работает на основе JavaScript и загружает информацию по запросу со стороны клиента. JavaScript не обрабатывается большинством поисковых систем, поэтому большинство страниц попросту недоступны для сканирования поисковыми ботами.
- **SPA-сайты не работают без включенного JS в браузере.** Некоторые пользователи отключают JavaScript в своих браузерах, а без него ваше приложение не будет работать.

# Настройка окружения

Разработка Angular-приложений требует определенной подготовки.

## Node.js

Многие инструменты, используемые для разработки приложений Angular, зависят от *Node.js*. Для того чтобы проверить установлен ли *Node.js* на вашей системе введите команду `node -v` в консоли или терминале. Эта команда выведет версию установленного *Node.js*.

```
C:\>node -v
v10.15.3
```

Если ничего не выведено, необходимо установить *Node.js*. Скачать установочный пакет можно с официального сайта <https://nodejs.org/en/download/>.

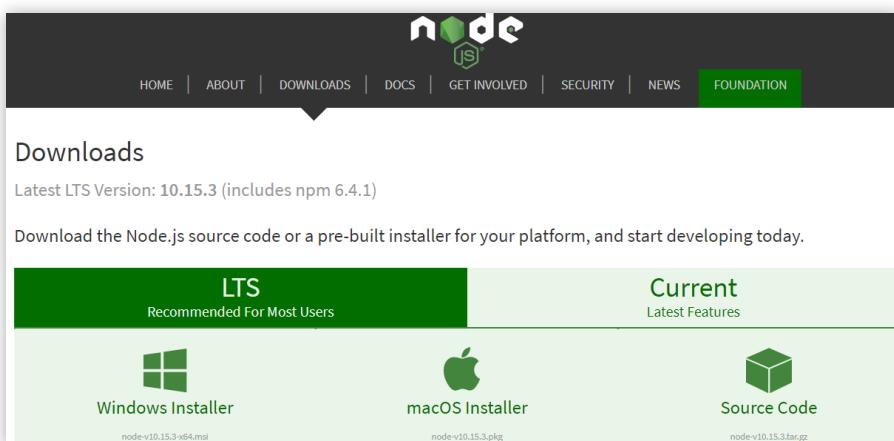


Рисунок 4

Вместе с *Node.js* устанавливается менеджер пакетов **npm**, для проверки установлен он или нет используйте в консоли или терминале команду **npm -v**:

```
C:\>npm -v
6.4.1
```

## Angular CLI

Далее необходимо установить Angular CLI — интерфейс командной строки, который позволяет быстро создавать проекты, добавлять файлы и выполнять множество определенных задач, такие как тестирование, сборка и развертывание.

```
C:\>npm install -g @angular/cli
[.....] | fetchMetadata: sill resolveWithNewModule util-deprecate@1.0.2
```

В системе Linux или macOS вам, возможно, придется использовать команду **sudo**.

## Редактор

В разработке приложений Angular можно использовать различные редакторы, их выбор огромен. В некоторых редакторах предусмотрена расширенная поддержка работы с Angular, включая выделение ключевых терминов и хорошую интеграцию с инструментарием.

Популярные редакторы с поддержкой Angular

Название	Описание
Sublime Text	Коммерческий кроссплатформенный редактор с пакетами для поддержки большинства языков программирования, фреймворков и платформ. За подробностями обращайтесь по адресу <a href="http://www.sublimetext.com/">http://www.sublimetext.com/</a>

Название	Описание
Atom	Бесплатный кроссплатформенный редактор с открытым кодом, уделяющий особое внимание возможностям настройки и расширения. За подробностями обращайтесь по адресу <a href="https://atom.io/">https://atom.io/</a>
Brackets	Бесплатный редактор с открытым кодом, разработанный компанией Adobe. За подробностями обращайтесь по адресу <a href="http://brackets.io/">http://brackets.io/</a>
WebStorm	Платный кроссплатформенный редактор с множеством интегрированных инструментов, чтобы вам не пришлось пользоваться командной строкой во время разработки. За подробностями обращайтесь по адресу <a href="https://www.jetbrains.com/webstorm/">https://www.jetbrains.com/webstorm/</a>
Visual Studio Code	Бесплатный кроссплатформенный редактор с открытым кодом, разработанный компанией Microsoft, с хорошими возможностями расширения. За подробностями обращайтесь по адресу <a href="https://code.visualstudio.com/">https://code.visualstudio.com/</a>

## Создание проекта

**Angular проект** — это набор файлов, который заключает в себе отдельное приложение, библиотеку или сквозные (e2e) тесты.

Одно Angular приложение может состоять из нескольких проектов, объединенных в одну рабочую область (*workspace*).

Чтобы создать новый **workspace** и начальный проект приложения используйте в консоли или терминале команду:

```
ng new <name>
ng n <name>
```

- `<name>` — это имя рабочей области и стартового проекта.
- `ng new` создаст одноименную директорию и поместит в нее исходные файлы «скелета» для Вашего приложения.

Для конфигурации Вашего приложения команда `ng new` запрашивает информацию о функциях, которые необходимо включить в первоначальный проект, например, добавление поддержки маршрутизации. Можно принять значения по умолчанию, нажав клавишу `enter` или `esc`.

Далее устанавливаются необходимые пакеты `npm` и другие зависимости. Это может занять несколько минут.

Создадим первое Angular приложение и назовем его `ToDoList`.

В консоли или терминале перейдите в каталог, в котором будет размещаться ваше Angular приложение и выполните команду:

```
C:\AngularWorkspaces>ng new ToDoList
```

После того, как `workspace` для приложения успешно установлен, перейдите в созданный каталог.

```
C:\AngularWorkspaces>cd ToDoList
```

```
C:\AngularWorkspaces\ToDoList>
```

Далее можно запустить стартовый проект. Для запуска Angular приложения необходим HTTP сервер.

При создании проекта с помощью Angular CLI команды `ng new`, так же был установлен `webpack-dev-server`. Это локальный веб-сервер `Node.js`, предназначенный специаль-

но для разработки, он поддерживает [LiveReload](#), перезагрузку страницы в браузере при обнаружении изменения в файле. Благодаря [webpack-dev-server](#) вы можете легко запускать свое приложение локально. Запустите сервер с помощью команды `CLI ng serve` с параметром `--open`.

```
C:\AngularWorkspaces\ToDoList>ng serve --open
```

Команда `ng serve` запускает сервер, просматривает ваши файлы и перестраивает приложение, когда вы вносите изменения в эти файлы.

Параметр `--open` указывает, что нужно открыть `url` в браузере по умолчанию.

Вы увидите следующую страницу:

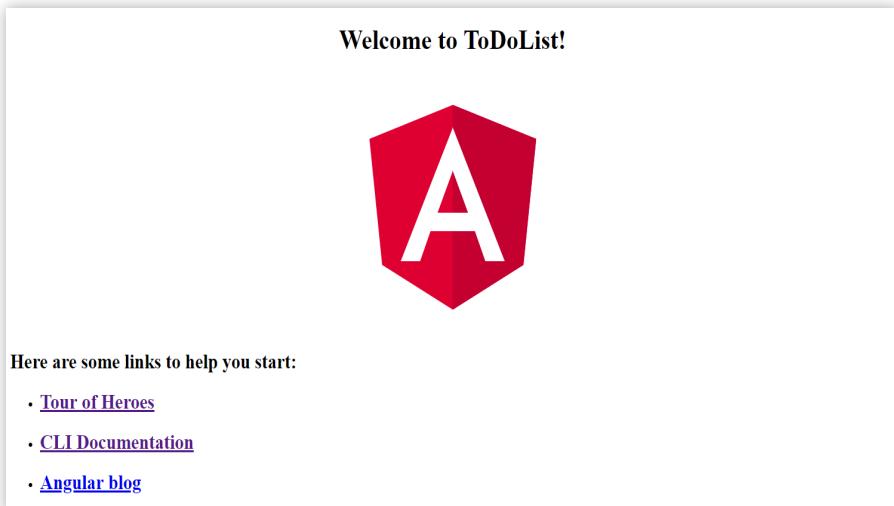


Рисунок 5

# Структура Angular приложения

Стартовое приложение уже имеет наполнение, в нем есть стартовый проект, проект для тестов, файлы конфигурации.

Зайдите в директорию только что сгенерированного проекта, и вы увидите следующую структуру:

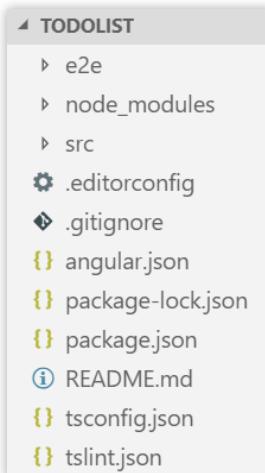


Рисунок 6

Каталоги:

- *e2e* — папка для end to end тестов. Обычно *e2e* используются для интеграционного тестирования и помогают убедиться, что приложение работает нормально.
- *node\_modules* — каталог для установленных **npm** пакетов.
- *src* — в этом каталоге размещаются проекты Angular.

Файлы:

- *.editorconfig* — конфигурационный файл для редактора.
- *.gitignore* — определяет намеренно не отслеживаемые файлы, которые Git должен игнорировать.
- *angular.json* — параметры конфигурации CLI для всех проектов в workspace, включая параметры конфигурации для инструментов сборки, обслуживания и тестирования, которые использует CLI.
- *package.json* — описывает зависимости от npm пакетов, доступных для всех проектов в workspace.
- *package-lock.json* — цель этого файла заключается в отслеживании точных версий установленных пакетов.
- *tsconfig.json* — конфигурация TypeScript для проектов в workspace.
- *tslint.json* — конфигурация TsLint, статического анализатора, который проверяет код TypeScript на наличие ошибок читаемости, удобства обслуживания и функциональности.

Каталог *src* является основным, в нем мы и будем работать. Посмотрим на его структуру:

- *app* — директория, в которой хранится весь исходный код приложения.
- *assets* — директория, в которой вы размещаете изображения и другие ресурсы, ее необходимо скопировать в конечную директорию сборки, когда вы создадите непосредственно само приложение.
- *index.html* — Это основная html-страница для нашего проекта. Тут можно добавить/изменить мета-теги **title**,

[description](#), изменить кодировку документа, [viewport](#) и т.д.

- *main.ts* — Отвечает за запуск всего нашего приложения.
- *polyfills.ts* — Подключаются различные библиотеки необходимые для работы.
- *tsconfig.app.json* — Файл отвечающий за компиляцию TypeScript.

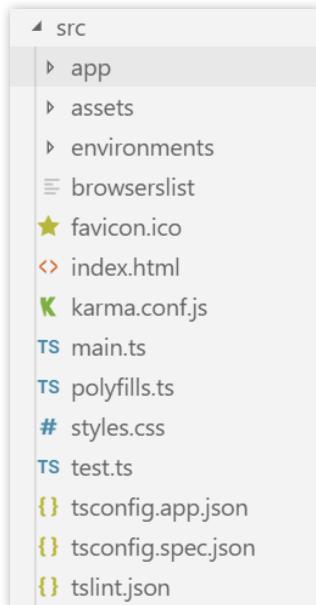


Рисунок 7

Вся основная работа над проектом будет происходить в папке *src/app*. Тут мы будем создавать компоненты, модули, сервисы и т.д. В общем все основные сущности фреймворка Angular.

# Angular модули

Angular приложение состоит из разных элементов: компоненты, шаблоны, директивы, сервисы и pipes. Большинство этих элементов мы будем изучать в нашем курсе.

Реальное приложение Angular будет состоять из множества таких элементов. По мере увеличения размера приложения управление ими становится сложным. Angular обеспечивает удобный способ упорядочивания этих элементов простым и эффективным образом с использованием Angular модулей.

Следует различать модули Angular и модули JavaScript.

Модули JavaScript, которые также называются JS-модулями, ES-модулями или ECMAScript-модулями, являются частью языка JavaScript. Модули JS хранятся в файле. Существует ровно один модуль на файл и один файл на модуль. Эти модули содержат небольшие блоки независимого, многократно используемого кода. Они экспортируют значение, которое можно импортировать и использовать в каком-то другом модуле.

Ниже приведен модуль Javascript, который импортирует `SomeClass` и экспортирует `SomeOtherClass`:

```
import { SomeClass } from './someModule';

export class SomeOtherClass {
  ...
}
```

Когда вы создаете Angular приложение, каждый скрипт, использующий **import** или **export** в исходном коде считается модулем JavaScript.

Модуль Angular представляет собой класс, к которому был применен декоратор **@NgModule**. Он помогает нам организовать части приложения в единые блоки. Каждый блок ориентирован на предоставление определенной функциональности. Компоненты, директивы, сервисы, которые реализуют эту функциональность, станут частью этого Модуля.

Модульная конструкция помогает разделить проблемы. Облегчает поддержку кода. Облегчает повторное использование кода.

Каждое приложение Angular содержит по крайней мере один модуль, называемый корневым модулем. Он описывает приложение для Angular и настраивает такие жизненно важные аспекты, как компоненты и службы.

Корневой модуль обычно определяется в файле с именем **app.module.ts** в папке **app**.

```
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 { }
```

Декоратор `NgModule()` — это функция, которая принимает один объект метаданных, свойства которого описывают модуль.

Свойство `declarations` передает Angular список директив, компонентов и каналов (`pipes`), которые принадлежат этому модулю.

Свойство `imports` используется для перечисления других модулей, чьи экспорттированные классы нужны в этом модуле.

Свойство `providers` определяет сервисы, которые вы хотите добавить в глобальную коллекцию сервисов. Эти сервисы будут доступны для внедрения через внедрение зависимостей (`dependency injection`).

Свойство `bootstrap` указывает основной компонент этого модуля, который должен быть загружен при загрузке модуля.

Корневой модуль нашего приложения импортирует модуль `BrowserModule`, который предоставляет функциональность, необходимую для запуска приложений Angular в браузерах.

Так же импортируется компонент `AppComponent` и объявляется как основной в свойстве `bootstrap`.

# Angular компоненты

Если вы сейчас посмотрите в папку `src/app`, то увидите тут 4 файла (`app.component.css`, `app.component.html`, `app.component.ts`, `app.component.spec.ts`) — это файлы корневого компонента нашего проекта.

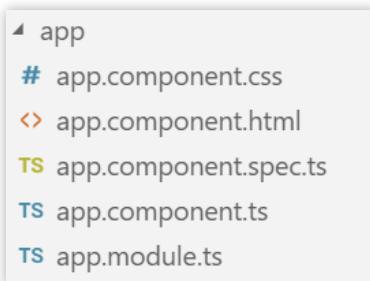


Рисунок 8

- `app.component.ts` — код класса компонента, написанный на TypeScript.
- `app.component.html` — шаблон компонента, написанный на HTML.
- `app.component.css` — собственные стили CSS компонента.
- `app.component.spec.ts` — unit тесты для компонента.

**Компонент** — это часть интерфейса приложения с собственной логикой. Весь интерфейс Angular приложения реализуется с помощью компонентов, поэтому они являются основными строительными блоками.

Каждый компонент состоит из трех частей: представления, класса и метаданных.

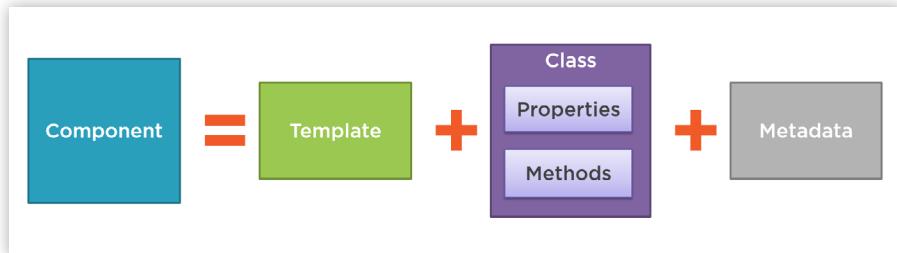


Рисунок 9

Представление (или Шаблон) определяет пользовательский интерфейс. Выглядит как обычный HTML, но также содержит директивы и привязки Angular (их мы разберем позже).

Класс, содержит данные и логику компонента. Как правило пишется на языке TypeScript, хотя можно его создавать с помощью таких языков как Dart или JavaScript.

**TypeScript** — это язык программирования, основанный на JavaScript и расширяющий его возможности. Был представлен Microsoft в 2012 году.

TypeScript является строго типизированным и компилируемым языком. Он реализует объектно-ориентированные концепции, такие как инкапсуляция, наследование и полиморфизм.

Компилятор TypeScript на выходе создает все тот же JavaScript, который затем исполняется браузером.

Метаданные предоставляют дополнительную информацию о компоненте Angular. Angular использует эту информацию для обработки класса. Метаданные определяются декоратором.

**Декоратор** — это функция, которая добавляет метаданные в класс, его методы и свойства.

Класс становится Компонентом, когда используется декоратор `@component`. Декоратор всегда имеет префикс `@`. Декоратор должен быть расположен непосредственно перед определением класса. Декораторы похожи на атрибуты в C#.

С помощью привязок данных в шаблон вставляются значения из свойств класса. При выполнении пользователем каких-либо действий в представлении, срабатывают привязки событий, они вызывают методы класса.

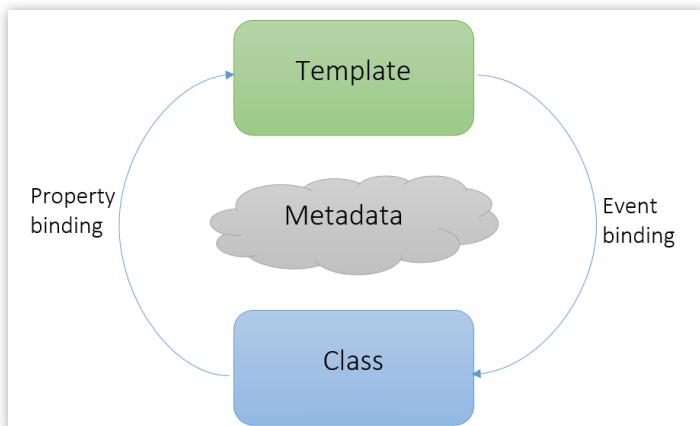


Рисунок 10

Давайте исследуем сгенерированный компонент.

Откройте проект в своем любимом редакторе или IDE и посмотрите код определенный в файле `./src/app/app.component.ts`.

```

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

@Component({
  selector: 'app-root',
  ...
)
  
```

```

    templateUrl: './app.component.html',
    styleUrls: ['./app.component.css']
})

export class AppComponent {
  title = 'ToDoList';
}

```

Компонент представляет собой класс `AppComponent`, перед которым указана функция декоратор `@Component`. Каждый декоратор `@Component` должен задавать свойства `selector` и `template` (или `templateUrl`), которые позволяют установить, как компонент должен определяться и отрисовываться на странице.

Свойство `selector` похоже на селекторы CSS. В элемент с этим селектором Angular будет добавлять представление компонента.

Свойства `template`, либо `templateUrl` определяют представление (шаблон), которое представляет собой блок разметки HTML с вкраплениями кода Angular.

Свойство `styles` или `styleUrls` позволяют определить набор стилей, которые будут использоваться компонентом.

Изменим значение свойства `title`:

```
title = 'To Do List';
```

Откроем файл шаблона компонента (`app.component.html`):

```

<!--The content below is only a placeholder and can
be replaced.-->
<div style="text-align:center">
```

```
<h1>
  Welcome to {{ title }}!
</h1>

</div>
<h2>Here are some links to help you start: </h2>

<ul>
  <li>
    <h2>
      <a target="_blank" rel="noopener"
          href="https://angular.io/tutorial">
        Tour of Heroes
      </a>
    </h2>
  </li>
  <li>
    <h2>
      <a target="_blank" rel="noopener"
          href="https://angular.io/cli">
        CLI Documentation
      </a>
    </h2>
  </li>
  <li>
    <h2>
      <a target="_blank" rel="noopener"
          href="https://blog.angular.io/">
        Angular blog
      </a>
    </h2>
  </li>
</ul>
```

Здесь мы видим разметку стартовой страницы, которая отобразилась в браузере, когда мы запустили HTTP сервер.

# Интерполяция строк

Удалим шаблон по умолчанию и заменим его следующей строкой HTML.

```
<h1>{{title}}</h1>
```

**Двойные фигурные скобки** — это синтаксис привязки элемента DOM к значению компонента. Так же называется интерполяцией.

Браузер обновляет страницу и отображает название нового приложения.

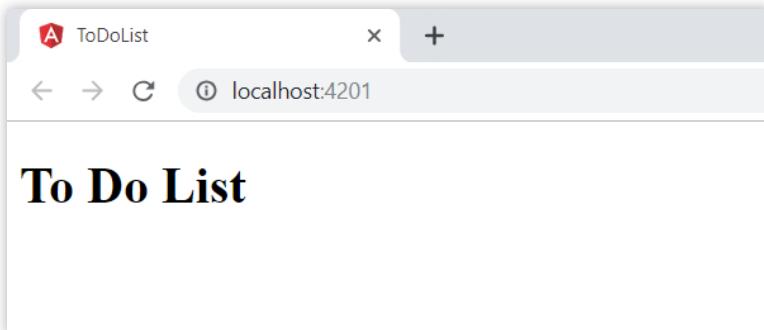


Рисунок 11

**Интерполяция** — это односторонняя привязка. Т.е. изменение данных происходят только в одном направлении. При изменении значения поля в классе обновится шаблон, но в шаблоне изменить значение нельзя.

Существует и двухсторонняя привязка, но ее мы рассмотрим позже.

В фигурные скобки можно вставлять не только свойство класса также допускается использование простых

арифметических операций, операций со строками (конкатенация), можно вызвать метод класса.

```
<p>{{ 'Hello & Welcome to ' +  
      ' Angular Data binding '} }</p>  
<p>{{ 100 * 80 }}</p>  
<p>{{ getTitle() }}</p>
```

С интерполяцией нельзя использовать:

- Присваивания (`=, +=, -=, ...`)
- Операторы, такие как `new, typeof, instanceof` и т.д.
- Несколько выражений разделенных; или же,
- Операторы инкремента и декремента `++` и `--`.

# Создание НОВОГО КОМПОНЕНТА

Теперь давайте создадим новый компонент для отображения информации об элементе списка дел и поместим этот компонент в оболочку приложения.

Для создания нового компонента используется Angular CLI команда:

```
ng generate component <name>
ng g component <name>
```

<name> — это имя компонента

Создадим компонент **ToDoItems**:

```
C:\AngularWorkspaces\ToDoList>ng generate component ToDoItems
```

CLI создает новую папку и четыре файла для описания компонента **ToDoItemsComponent**, а также объявляет его в главном модуле приложения:

```
CREATE src/app/to-do-items/to-do-items.component.html (30 bytes)
CREATE src/app/to-do-items/to-do-items.component.spec.ts (651 bytes)
CREATE src/app/to-do-items/to-do-items.component.ts (287 bytes)
CREATE src/app/to-do-items/to-do-items.component.css (0 bytes)
UPDATE src/app/app.module.ts (412 bytes)
```

CLI генерирует следующий код компонента:

```
import { Component, OnInit } from '@angular/core';
@Component({
  selector: 'app-to-do-items',
  templateUrl: './to-do-items.component.html',
  styleUrls: ['./to-do-items.component.css']
```

```

})
export class ToDoItemsComponent implements OnInit {
  constructor() { }
  ngOnInit() {
  }
}

```

Класс компонента реализует метод `ngOnInit()`, который объявлен в интерфейсе `OnInit`. Этот метод используется в жизненном цикле компонента и вызывается один раз после установки свойств компонента, которые участвуют в привязке. Он выполняет инициализацию компонента. Подробнее жизненный цикл мы будем разбирать позже.

Создадим класс, описывающий элемент списка дел и добавим в наш компонент объект этого класса. Для этого добавим новый файл `to-do-item.ts` в каталог `src/app`:

```

export class ToDoItem {
  id: number;
  name: string;
  isComplete: boolean;
}

```

А затем изменим класс компонента, добавив свойство `ToDoItem` и инициализировав его. Не забываем, что для описания класса используется язык TypeScript, поэтому при объявлении свойства мы можем указать его тип.

```

export class ToDoItemsComponent implements OnInit {
  ToDoItem: ToDoItem = {
    id: 1,
}

```

```

        name: "Call Joe",
        isComplete: false
    };

constructor() { }
ngOnInit() {
}

}

```

Для отображения нашего элемента обновим файл *to-do-items.component.html*, содержащий шаблон компонента. Добавим привязку для отображения свойств элемента.

```

<h2>{{ToDoItem.name}} Details</h2>

<div><span>id: </span>{{ToDoItem.id}}</div>
<div><span>name: </span>{{ToDoItem.name}}</div>

<div>
    <span>is complete: </span>{{ToDoItem.isComplete}}
</div>

```

Чтобы отобразить **ToDoItemsComponent**, необходимо добавить его в шаблон главного компонента **AppComponent**.

В **ToDoItemsComponent** указан селектор **app-to-do-items**, поэтому добавьте элемент **<app-to-do-items>** в файл шаблона **AppComponent**, чуть ниже заголовка.

```

<h1>{{title}}</h1>
<app-to-do-items></app-to-do-items>

```

Браузер перезагрузит страницу и отобразит информацию:

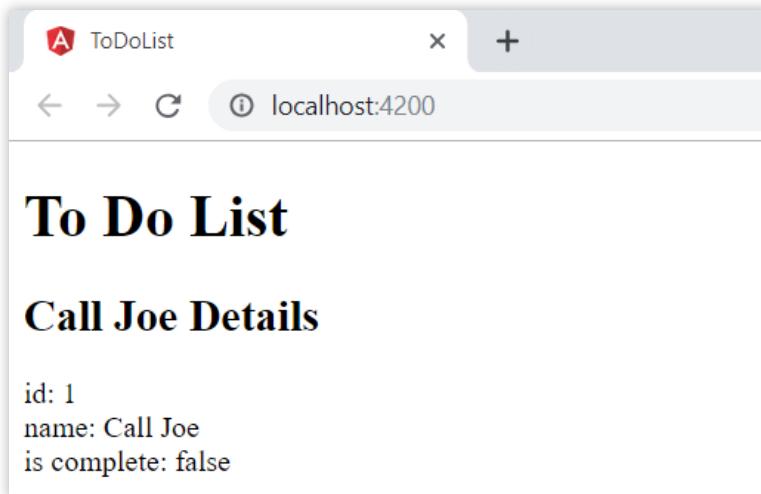


Рисунок 12

## Создание нового компонента



## Unit 3.

# Angular: директивы, data binding, сервисы, dependency injection, навигация

© Елена Сербова.

© Компьютерная Академия «Шаг», [www.itstep.org](http://www.itstep.org).

Все права на охраняемые авторским правом фото-, аудио- и видеопроизведения, фрагменты которых использованы в материале, принадлежат их законным владельцам. Фрагменты произведений используются в иллюстративных целях в объёме, оправданном поставленной задачей, в рамках учебного процесса и в учебных целях, в соответствии со ст. 1274 ч. 4 ГК РФ и ст. 21 и 23 Закона Украины «Про авторське право і суміжні права». Объём и способ цитируемых произведений соответствует принятым нормам, не наносит ущерба нормальному использованию объектов авторского права и не ущемляет законные интересы автора и правообладателей. Цитируемые фрагменты произведений на момент использования не могут быть заменены альтернативными, не охраняемыми авторским правом аналогами, и как таковые соответствуют критериям добросовестного использования и честного использования.

Все права защищены. Полное или частичное копирование материалов запрещено. Согласование использования произведений или их фрагментов производится с авторами и правообладателями. Согласованное использование материалов возможно только при указании источника.

Ответственность за несанкционированное копирование и коммерческое использование материалов определяется действующим законодательством Украины.