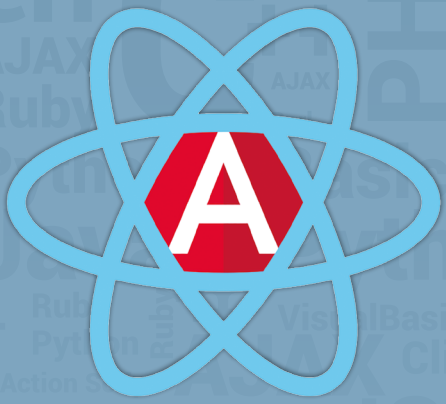
Building Web Application Using Angular and React

Angular & React



STEP computer ACADEMY

Lesson 3

Angular: Basics

Lesson materials are attached to this PDF file. In order to get access to the materials, open the lesson in Adobe Acrobat Reader.

Contents

What Is Angular?

Angular is an open-source framework that allows developing applications for multiple platforms: web, mobile, and desktop.

A framework is a kind of a skeleton used to ease the development and integration of different components of a large software project.

Frameworks provide a clear structure of an application and are implemented with design patterns. The structure implies the modularity of an app, which makes it easier for several developers to work on one app at the same time. A framework reduces the amount of code, makes it cleaner, which has a positive influence over the development speed, support, and bug fixes.

Tasks and Objectives of Angular

Angular aims at the development of Single-Page Application (or SPA) solutions.

It simplifies the creation of custom components that can be added to HTML documents, as well as the implementation of application logic. It makes extensive use of data binding, has a dependency injection module, supports modularity, and provides the mechanism for routing configuration.

The Concept of a Single-Page Application (SPA)

There are two main approaches to the creation of web apps: traditional Multi-Page Application (MPA) and Single-Page Application (SPA).

In a multi-page web app, every time the app calls the server, the server displays a new HTML page. This causes the page to refresh in the browser.

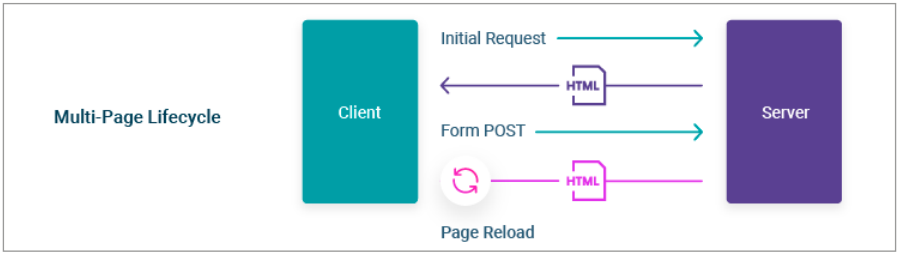


Figure 1

In a single-page web app, the whole interaction after the first page loads is done through AJAX calls. These AJAX calls return data instead of the markup, usually in the JSON format. The app uses JSON data to refresh the page dynamically, without reloading it.

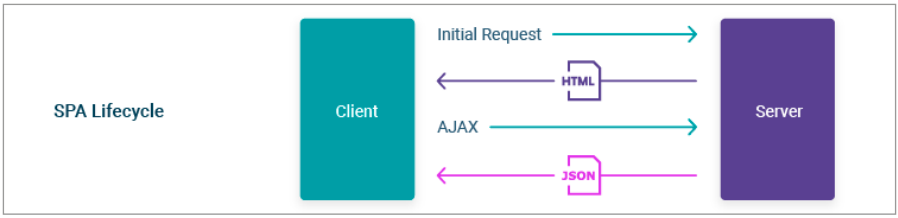


Figure 2

You have most probably used a single-page app. For example, when you scroll your Facebook feed, new posts load dynamically.

In MPA, most of the logic, including view, is done on the server side; and in SPA, the user interface logic is mostly done on the client side. All controls are processed and sampled

with JavaScript. That is, the server returns only data, usually in the JSON format, and the client side shapes the website page, all templates, lists, links, tables, and other updatable items.

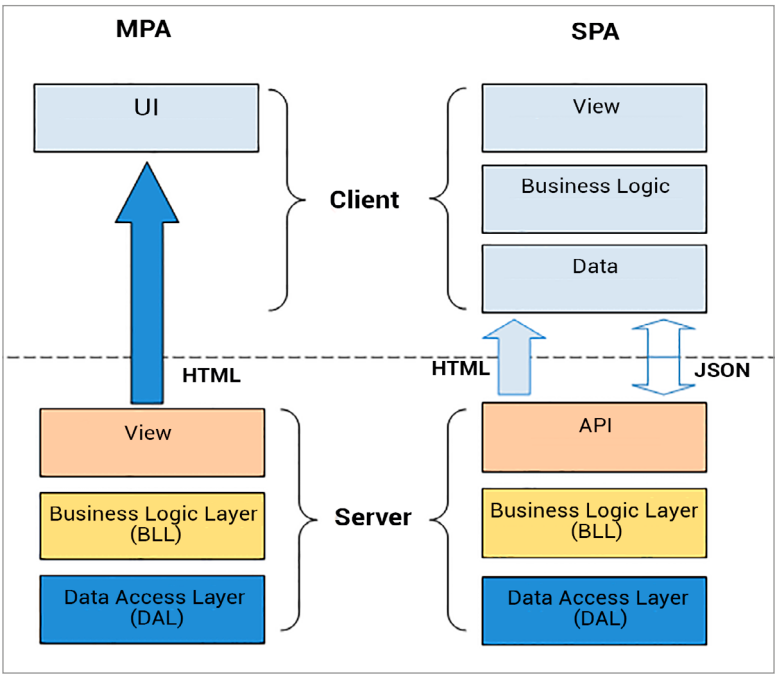


Figure 3

Single-page apps load faster by extracting data in the background. Because most resources (like HTML, CSS, scripts) load only once during the lifetime of the app, the time of response to user actions is shorter.

SPAs also increase development speed. Ready-made libraries and frameworks provide powerful tools for web app development. Both back-end and front-end developers can work on the project in parallel. They will not interfere with each other's work due to clear separation.

But despite all the benefits, Single-Page Application has some disadvantages:

- *Poor SEO.* SPA is based on JavaScript and loads information upon the client's request. Most search engines do not process JavaScript, this is why the majority of pages simply cannot be scanned by search engine bots.
- *SPA sites do not work if JS is disabled in the browser.* Some users turn off JavaScript in their browsers, and your app cannot work without it.

Environment Setup

The development of an Angular app requires certain skills.

Node.js

Many tools used for the development of Angular apps depend on *Node.js*. In order to check if you have *Node.js* installed, enter the command "node –v" in the console or terminal. This command outputs version of the installed *Node.js*.

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

If nothing outputs, install *Node.js*. You can download the installation package from the official website https://nodejs.org/en/download/.

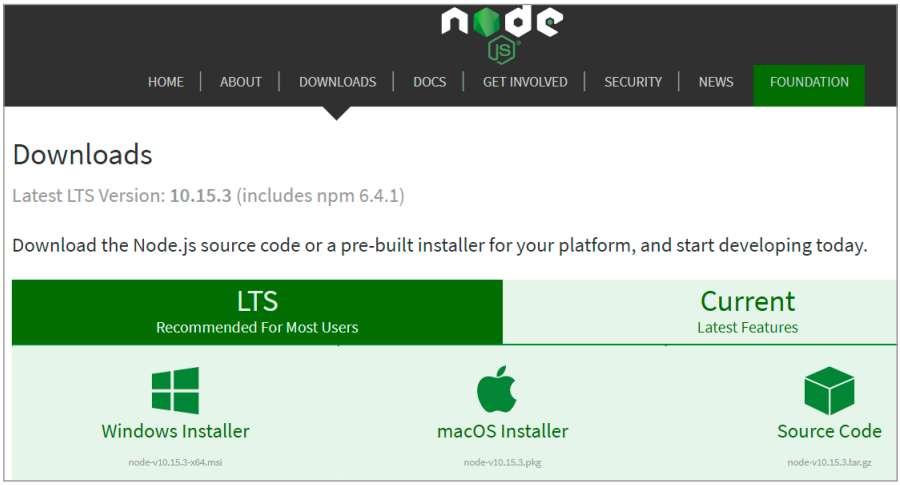


Figure 4

The npm, Node Package Manager, is installed along with *Node.js*. Check if you have it by writing the command "npm –v" in the console or terminal.

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

Angular CLI

Next, install the Angular CLI, a command-line interface that allows for quick project creation, file adding, and performing many tasks, such as testing, assembly, and deployment.

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

Linux or macOS might require the sudo command for this.

Editor

There are many code editors to choose from for the development of Angular apps. Some editors provide enhanced support for Angular, including highlighting key terms and good integration with the toolkit.

Popular editors that support Angular

Name	Description
Sublime Text	Sublime Text is a commercial cross-platform editor with packages to support most programming languages, frameworks, and platforms. For details, go to http://www.sublimetext.com/
Atom	Atom is an open-source, cross-platform editor that focuses on customization and extension options. For details, go to https://atom.io/
Brackets	Brackets is an open-source editor developed by Adobe. For details, go to http://brackets.io/

Name	Description
WebStorm	WebStorm is a paid cross-platform editor with many integrated tools, so you do not have to access the command-line all the time as you work. For details, go to https://www.jetbrains.com/webstorm/
Visual Studio Code	Visual Studio Code is an open-source, cross-platform editor developed by Microsoft, with good expansion options. For details, go to https://code.visualstudio.com/

Creating a Project

An Angular project is a set of files that contains a single app, a library, or end-to-end (e2e) tests.

One Angular project can consist of multiple projects united into one workspace.

To create a new workspace and startup project, use this command in the console or terminal:

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

- <name> is the name of the workspace and startup project.
- ng new creates a directory of the same name and puts there source files of the "skeleton" of your app.

To configure your app, the ng new command requests information about the functions to be included in the startup project, like adding routing support. You can accept the default values by pressing Enter or Esc.

Next, the required npm and other dependencies will be installed. This could take a while.

Create your first Angular app and name it ToDoList.

In the console or terminal, go to the directory that will contain your Angular app and enter the command:

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

After the workspace for your app is successfully installed, go to the created directory.

```
C:\AngularWorkspaces>cd ToDoList

C:\AngularWorkspaces\ToDoList>
```

Then you can run the startup project. Running an Angular app requires an HTTP server.

When you created the project with the Angular CLI command "ng new," the webpack-dev-server was also installed. This local *Node.js* web server is designed specifically for the development. It supports LiveReload, a page reload in the browser after changes made to the file are detected. Thanks to the webpack-dev-server you can easily run your app locally. Run the server with the CLI ng serve command and the --open parameter.

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

The ng serve command starts the server, views your files, and rebuilds the app when you make changes to these files.

The --open parameter indicates that you should open url in the default browser.

You will see this page:

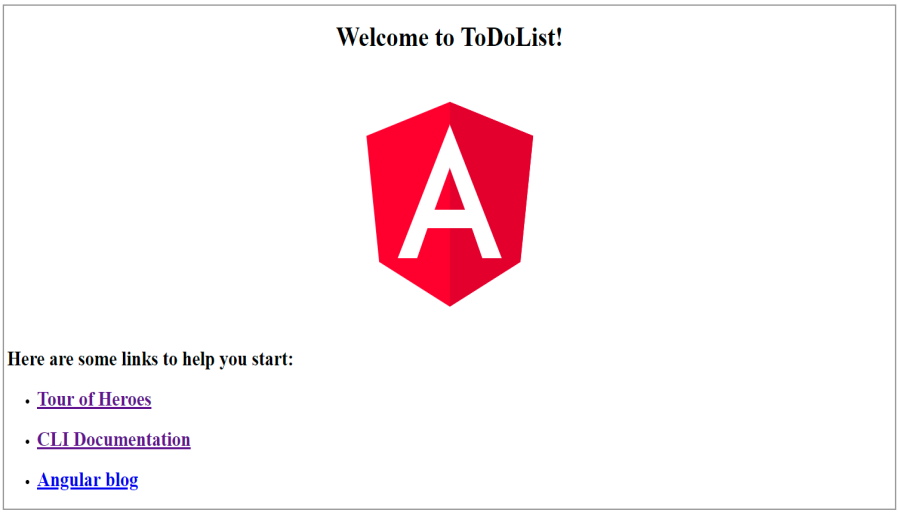


Figure 5

Structure of an Angular App

The startup app already has contents made up of the startup project, test project, configuration files.

Go to the directory of the newly generated project, and you will see this structure:

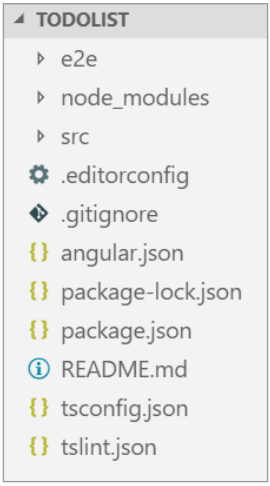


Figure 6

Directories:

- *e2e* is a folder for end-to-end tests. Usually, e2e are used for the integration test and help to ensure that the app is working properly.
- *node_modules* is a directory for the installed npm packages.
- src has Angular projects.

Files:

- *.editorconfig* is a configuration file for the editor.

- *.gitignore* determines the intentionally unmonitored files that Git must ignore.
- *angular.json* is CLI configuration parameters for all projects in the workspace, including configuration parameters for the build, maintenance, and testing used by CLI.
- *package.json* describes dependencies in npm packages available to all projects in the workspace.
- *package-lock.json* aims at tracking a precise version of the installed packages.
- *tsconfig.json* is the TypeScript configuration for projects in the workspace.
- *tslint.json* is the TsLint configuration, a static analyzer that checks the TypeScript code for readability errors, ease of maintenance, and features.

The src directory is the main, and we will work in it. Let's take a look at its structure:

- *app* is a directory that stores the whole source code of the app.
- *assets* is the directory where you put images and other assets. Make sure to copy it to the final assembly directory after you create the app.
- *index.html* is the main html page for our project. Here you can add/edit meta tags such as title and description, change the encoding, viewport, etc.
- *main.ts* is responsible for running our app.
- *polyfills.ts* adds various libraries necessary for work.
- *tsconfig.app.json* is a file responsible for the TypeScript compilation.

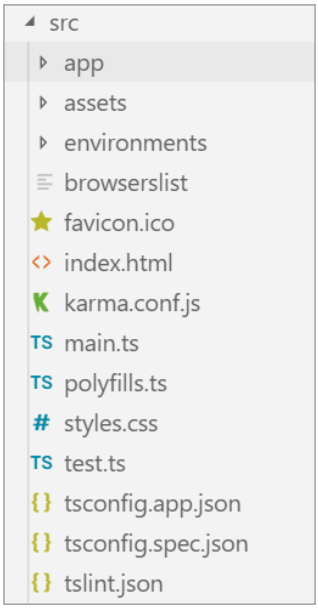


Figure 7

The major work on the project will take place in the *src/app* folder. Here we will create components, modules, services, etc., basically, all the main entities of the Angular framework.

Angular Modules

The Angular app consists of different elements: components, templates, directives, services, and pipes. We will learn most of them within this course.

A real Angular app consists of many such elements. They get harder to manage as the app becomes larger. Angular uses modules to provide a convenient way of easy and efficient ordering of these elements.

You should distinguish betweenAngular and JavaScript modules.

The JavaScript modules, also known as JS modules, ES modules, or ECMAScript modules, are a part of JavaScript. JS modules are stored in a file. There is exactly one module per file and one file per module. These modules contain small blocks of independent, reusable code. They export a value to be imported and used in some other module.

Below is the JavaScript module that imports SomeClass and exports SomeOtherClass:

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

export class SomeOtherClass {
  …
}
```

When you create an Angular app, every script that uses import or export in the source code is considered to be a JavaScript module.

The Angular module is a class which has the @NgModule decorator. It helps us to organize parts of the app into single blocks. Every block focuses on a certain feature. Components, directives, services that use this feature become part of this module.

The modular design helps share problems, it eases the code support and reuse.

Every Angular app has at least one module called the root module. It describes an Angular app and configures vital things such as components and services.

The root module is usually defined in the app.module.ts file, in the app folder.

```
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 { }
```

The NgModule () decorator is a function that takes a single metadata object whose properties describe the module.

The declarations property transmits to Angular the list of directives, components, and pipes that belong to this module.

The imports property lists other modules whose exported classes are required in this module.

The providers property determines the services that you want to add to the global collection of services. These services are available through the dependency injection.

The bootstrap property indicates the main component of this module that has to be loaded during the module load.

The root module of our app imports the BrowserModule that provides features required for running Angular apps in browsers.

Also, the AppComponent is imported and declared as the main in the bootstrap property.

Angular Components

If you go to the *src/app* now, you will see four files (*app.component.css*, *app.component.html*, *app.component.ts*, *app.component.spec.ts*). They are files of the root component of our project.

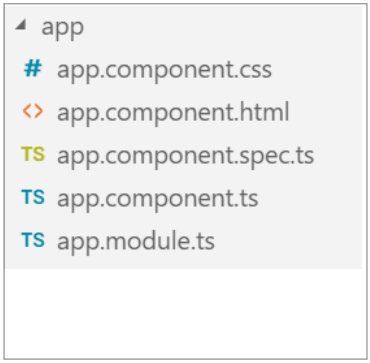


Figure 8

- *app.component.ts* is the component class code written with TypeScript.
- *app.component.html* is the component template written with HTML.
- *app.component.css* is a custom style of the CSS component.
- *app.component.spec.ts* is unit tests for the component.

A *component* is part of the app interface with its own logic. The whole interface of an Angular app is implemented with components, so they are the main building blocks.

Every component consists of three parts: view, class, and metadata.

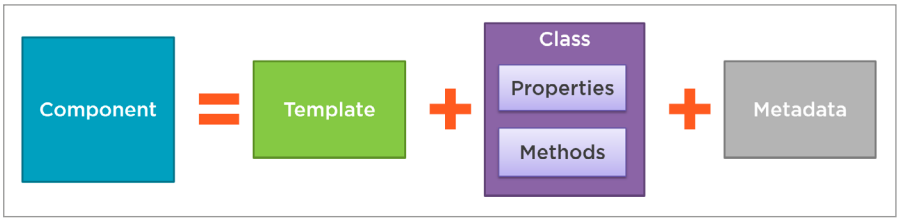


Figure 9

View (or Template) defines the user interface. It looks like a regular HTML but also contains Angular directives and binds (we will sort them out later).

A class has data and logic of the component. It is usually written in TypeScript, although it can be created with programming languages such as Dart or JavaScript.

TypeScript is a JavaScript-based programming language that expands the possibilities of JavaScript. It was presented by Microsoft in 2012.

TypeScript is a strongly typed and compiled language. It implements object-oriented concepts such as encapsulation, inheritance, and polymorphism.

The TypeScript compiler outputs same old JavaScript executed by the browser.

Metadata provide additional information about the Angular component. Angular uses this information for class processing. Metadata are determined by the decorator.

Decorator is a function that adds metadata to the class, along with its methods and properties.

The class becomes a Component when the @component decorator is used. The decorator always has the @ prefix. The decorator must go before the class definition. Decorators are similar to C# attributes.

They use data binding to insert values from the class properties in the template. When the user does something in the view, event bindings trigger and call class methods.

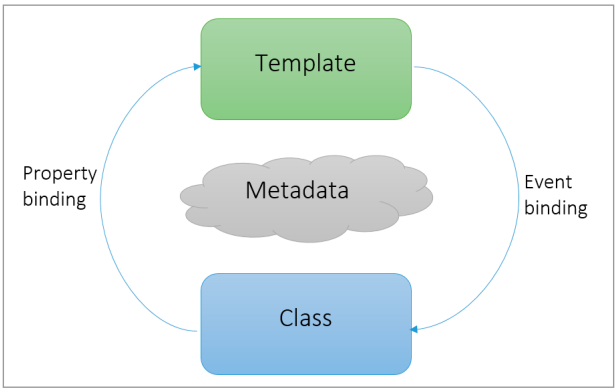


Figure 10

Let's examine the generated component.

Open the project in your favorite editor or IDE and look at the code defined in the file *./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';
}
```

The component is the AppComponent class preceded by the decorator function @Component. Every @Component

decorator must set the selector and template (or templateUrl) properties that help establish how the component must be defined and sampled on the page.

The selector property is similar to CSS selectors. Angular adds the component view to the element with this selector.

The template or templateUrl property defines the view (template) which is an HTML markup with some Angular code.

The styles or styleUrls property defines a set of styles to be used by the component.

Change the value of the title property:

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

Open the *app.component.html* file:

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

Here we see the markup of the start page that displayed in the browser when we started the HTTP server.

String Interpolation

Delete the default template and replace it with this HTML string.

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

Double curly braces are the syntax of binding the DOM element to the component value. It is also called interpolation.

The browser refreshes the page and displays the name of the new app.

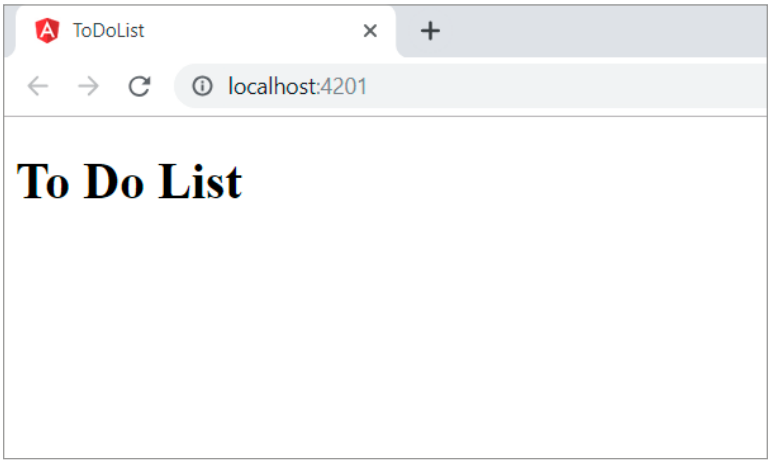


Figure 11

Interpolation is a one-way binding. That is, data are changed only in one direction. If you change the field value, the template in the class updates. But you cannot change the value in the template.

There is also a two-way binding, but we will discuss it later.

Curly braces can contain not only a class property but simple arithmetic operations, string operations (concatenation), or you can call a class method.

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

Things that you cannot use in interpolation:

- Assignments (=, + =, – =, ...)
- Operators such as new, typeof, instanceof, etc.
- Several expressions separated by ; or ,
- Increment and decrement operators: ++ and --.

Creating a New Component

Now let's add a new component to display information about a ToDo list item and place this component in the app shell.

A new component is added with this Angular CLI command:

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

<name> is the component name.

Create the ToDoItems component:

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

CLI creates a new folder and four files that describe the ToDoItemsComponent, it also declares the latter in the main module of the app:

```
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 generates the following code of the component:

```
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() {
  }
}
```

The component class is implemented by the ngOnInit() method declared in the OnInit interface. This method is used in the component's lifecycle and called once after you install the component's properties that participate in binding. It initializes the component. We will analyze lifecycle in more detail a bit later.

Create a class that describes a ToDo list item and add the object of this class to our component. For this, add a new file *to-do-item.ts* to the *src/app* directory:

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

Then change the component class by adding and initializing the toDoItem property. Do not forget that TypeScript is used for the description of the class, so we can specify its type as we declare properties.

```
export class ToDoItemsComponent implements OnInit {
  toDoItem: ToDoItem = {
      id: 1,
      name: "Call Joe",
```

```
      isComplete: false
  };

  constructor() { }
  ngOnInit() {

  }
}
```

To display our element, update the file *to-do-items.component.html* that contains the component template. Add a binding to display the element properties.

```
<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>
```

To display the ToDoItemsComponent, add it to the template of the main AppComponent.

The ToDoItemsComponent specifies the app-to-do-items selector, so add the <app-to-do-items> element to the template file AppComponent, below the title.

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

The browser will refresh the page and display the information:

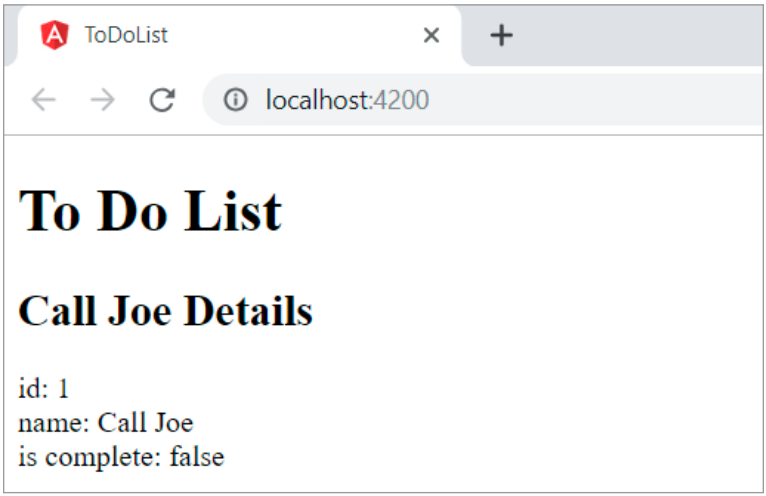

ToDoList
localhost:4200
To Do List
Call Joe Details
id: 1
name: Call Joe
is complete: false

Figure 12

Lesson 3

Angular: Basics