



Community Experience Distilled

What you need to know about Angular 2

Start building your next projects in Angular 2

Oliver Manickum

[PACKT]
PUBLISHING

What You Need to Know about Angular 2

The absolute essentials you need to get Angular 2 up and running

Oliver Manickum

[PACKT]
PUBLISHING

BIRMINGHAM - MUMBAI

What You Need to Know about Angular 2

Copyright © 2016 Packt Publishing

All rights reserved. No part of this book may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, without the prior written permission of the publisher, except in the case of brief quotations embedded in critical articles or reviews.

Every effort has been made in the preparation of this book to ensure the accuracy of the information presented. However, the information contained in this book is sold without warranty, either express or implied. Neither the author, nor Packt Publishing, and its dealers and distributors will be held liable for any damages caused or alleged to be caused directly or indirectly by this book.

Packt Publishing has endeavored to provide trademark information about all of the companies and products mentioned in this book by the appropriate use of capitals. However, Packt Publishing cannot guarantee the accuracy of this information.

First Published: July 2016

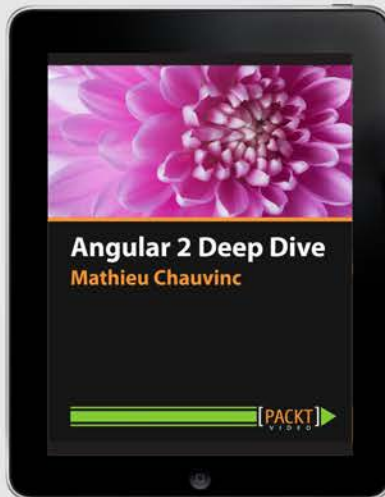
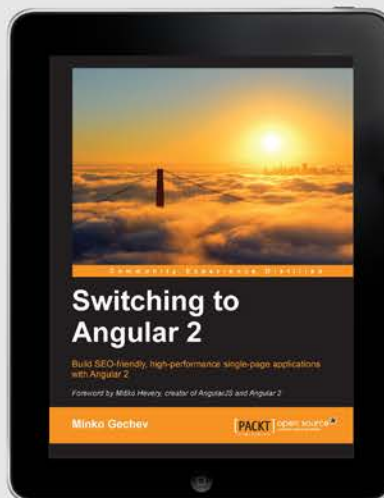
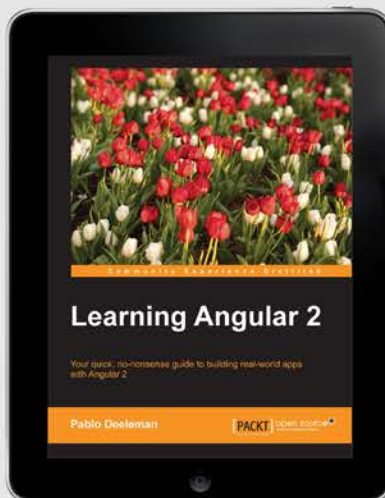
Production reference: 1300616

Published by Packt Publishing Ltd.
Livery Place
35 Livery Street
Birmingham B3 2PB, UK.

www.packtpub.com

Get
50%
Off

Your next eBook or Video



Use the following code to
apply your exclusive discount

ANGULAR50

About the Author

Oliver Manickum is a full stack developer and hails from South Africa. He is a passionate Arduino hacker and loves dabbling in all things electronic. When he is not having fun building stuff that will make MacGyver proud, he develops applications and games using Angular and HTML5. He has over 20 years of commercial experience and has developed and deployed hundreds of projects over his career.

I would like to thank my wife, Nazia Osman, for her constant support and understanding.

About the Reviewer

Kerry-Leigh holds 8 years of experience developing application solutions in industries such as finance, catering, cargo, security, and aviation. She has specialized skills in Oracle, Apex, and Java. Her experience in coding is not limited to the development of code; it extends to the entire development life cycle, from JAD sessions to user support. She has also managed and lead development teams and mentored junior developers.

The highlights of Kerry's career include the development of a world-class bridging finance system, a trade finance system, and a successful catering system, all of which provide user-friendly interfaces.

This, being my first book, has been an incredible journey, and I loved the experience and also have learned so much. This book gives so much insight, and the author really has outdone himself.

www.PacktPub.com

Support files, eBooks, discount offers, and more

At www.PacktPub.com, you can also read a collection of free technical articles, sign up for a range of free newsletters and receive exclusive discounts and offers on Packt books, eBooks, and videos.



<https://www2.packtpub.com/books/subscription/packtlib>

Do you need instant solutions to your IT questions? PacktLib is Packt's online digital book library. Here, you can access, read and search across Packt's entire library of books.

Why subscribe?

- Fully searchable across every book published by Packt
- Copy and paste, print and bookmark content
- On demand and accessible via web browser

Free access for Packt account holders

If you have an account with Packt at www.PacktPub.com, you can use this to access PacktLib today and view nine entirely free books. Simply use your login credentials for immediate access.

Table of Contents

Preparing for Angular 2	1
Options required to set up Angular 2	2
Installing Node.js and npm	2
Setting up the application folder	3
Components, Directives, and Templates	9
Using components to reuse Angular tasks	9
How do we use directives?	11
Templates and the power behind them	12
Metadata, Data Binding, and Services	14
How do we use data binding?	15
Creating services	16
Dependency Injection	20
Wrapping Up	23
Summary	26
What to do next?	28
Broaden your horizons with Packt	28

What you need to know about Angular 2

This eGuide is designed to act as a brief, practical introduction to Angular 2. It is full of practical examples which will get you up a running quickly with the core tasks of Angular 2.

We assume that you know a bit about what Angular 2 is, what it does, and why you want to use it, so this eGuide won't give you a history lesson in the background of Angular 2. What this eGuide will give you, however, is a greater understanding of the key basics of Angular 2 so that you have a good idea of how to advance after you've read the guide. We can then point you in the right direction of what to learn next after giving you the basic knowledge to do so.

What You Need to Know about Angular 2 will:

- Cover the fundamentals and the things you really need to know, rather than niche or specialized areas.
- Assume that you come from a fairly technical background and so understand what the technology is and what it broadly does.
- Focus on what things are and how they work.
- Include 3-5 practical examples to get you up, running, and productive quickly.

Overview

AngularJS was created by Google in 2010 to address the need of creating awesome single-page applications.

With the success and failures of Angular 1.x, Angular 2 has been created. It embraces ES6 and TypeScript to create beautiful JavaScript code that gets compiled for the browser.

This eGuide will provide a quick insight of Angular 2 as seen from Angular 1.x developers and help get you on the road to writing beautiful Angular 2 code.

1

Preparing for Angular 2

For those of you who have used Angular 1.x, you will find this book a valuable source of information while progressing into Angular 2. Angular 1.x is maintained mostly by Google and a group of like-minded individuals who are passionate about developing single-page applications. While knowledge of Angular 1.x is not a requirement to understand this book, a healthy understanding of JavaScript, Node.js, and npm is required.

Before we go into the heart of Angular 2, let's try and understand why Angular 2 is so different from Angular 1.x; they are both built on JavaScript, and they are both scripted languages. What makes Angular 2 so exciting? The answer lies in the different parts that make up Angular 2 and the inclusion of TypeScript and ECMA6.



TypeScript is a free and open source language that is built by Microsoft to help JavaScript scale. Typescript provides advanced autocompletion, navigation, and refactoring. The website <https://www.typescriptlang.org/> details TypeScript. ECMA6 is the standard of JavaScript that enables the use of classes and modules. For more information about ECMA6, head to <http://es6-features.org/>.

You will not need a special IDE to develop in TypeScript; any enhanced Notepad style editor will do. Visual Studio Code by Microsoft is a great open source editor, and you can find it at <http://code.visualstudio.com/>. Let's start setting up the development environment to run Angular 2.

Options required to set up Angular 2

For this book, we will set up Angular 2 by doing the following:

- Installing Node.js and npm
- Creating a folder for our application
- Creating a TypeScript configuration file
- Adding dependencies for the TypeScript compiler
- Creating a `package.json` file
- Installing the npm packages

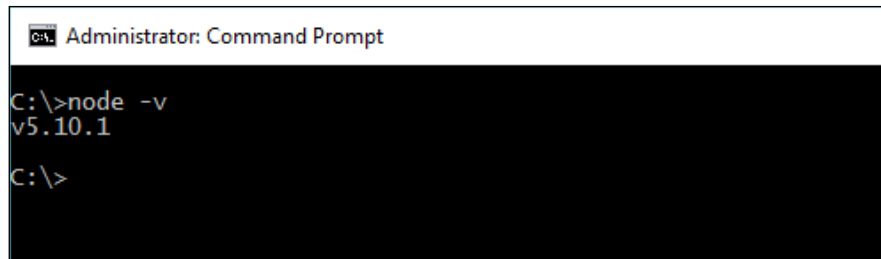
Installing Node.js and npm

To download and set up Node.js for your development environment, head to <https://nodejs.org/> and select the appropriate version (Mac, Windows, or Linux). For this book, we will use the **v5.10.1 Stable** version. Click on **Download** and install the application.

To verify that all is installed correctly, start your command prompt and type the following:

```
Node -v
```

The output should be similar to the following:

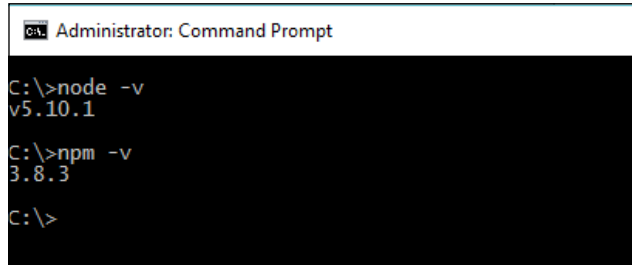


```
Administrator: Command Prompt
C:\>node -v
v5.10.1
C:\>
```

To verify that npm is working, type the following:

```
npm -v
```

The output will be as follows:



```
Administrator: Command Prompt
C:\>node -v
v5.10.1
C:\>npm -v
3.8.3
C:\>
```

If you see this output, with maybe different versions, both Node.js and npm are ready and available for use.

Setting up the application folder

Before we start our application, we must organize it into a location where Node.js and npm can find everything they need to compile and output to the system.

In the command prompt, let's create a folder called `angular2-helloworld`. Type the following:

```
mkdir angular2-helloworld
```

Then, type the following:

```
cd angular2-helloworld
```

We are now ready to put some content into this folder to get our Hello World application up and running.

Create a new file in this folder called `tsconfig.json` and add the following to it:

```
{
  "compilerOptions": {
    "target": "es5",
    "module": "system",
    "moduleResolution": "node",
    "sourceMap": true,
    "emitDecoratorMetadata": true,
    "experimentalDecorators": true,
    "removeComments": false,
    "noImplicitAny": false
  }
}
```

```
    },
    "exclude": [
      "node_modules",
      "typings/main",
      "typings/main.d.ts"
    ]
  }
}
```

This file will be the basic configuration for most of our TypeScript applications. This `tsconfig.json` file will tell the compiler how to process the TypeScript file. The important option is `noImplicitAny`, which tells the compiler how to handle variable declarations. By setting the variable to `false`, we can set the default type to `any`, which makes learning Angular 2 a little easier.



For more information about the `tsconfig.js` file, head to <https://www.typescriptlang.org/docs/handbook/tsconfig-json.html>.

Create a new file called `typings.json` and enter the following:

```
{
  "ambientDependencies": {
    "es6-shim": "github:DefinitelyTyped/DefinitelyTyped/es6-shim/es6-shim.d.ts#7de6c3dd94feaeb21f20054b9f30d5dabc5efabd",
    "jasmine": "github:DefinitelyTyped/DefinitelyTyped/jasmine/jasmine.d.ts#7de6c3dd94feaeb21f20054b9f30d5dabc5efabd"
  }
}
```

These two files will help TypeScript extend the language where it does not understand the code natively. `DefinitelyTyped` is a repo on GitHub that has most type definitions.

Next, create a file called `package.json` and add the following code:

```
{
  "name": "angular2-helloworld",
  "version": "1.0.0",
  "scripts": {
    "start": "tsc && concurrently \"npm run tsc:w\" \"npm run lite\"",
    "tsc": "tsc",
    "tsc:w": "tsc -w",
    "lite": "lite-server",
    "typings": "typings",
  }
}
```

```
    "postinstall": "typings install"
  },
  "license": "ISC",
  "dependencies": {
    "angular2": "2.0.0-beta.14",
    "systemjs": "0.19.25",
    "es6-shim": "^0.35.0",
    "reflect-metadata": "0.1.2",
    "rxjs": "5.0.0-beta.2",
    "zone.js": "0.6.6"
  },
  "devDependencies": {
    "concurrently": "^2.0.0",
    "lite-server": "^2.2.0",
    "typescript": "^1.8.9",
    "typings": "^0.7.12"
  }
}
```

Here, we will define the requirements for our `angular2-helloworld` application. The npm manager reads this file and downloads the required dependences, and this file also includes the startup parameters for npm to run our application.

The options in the `package.json` file are as follows:

- `tsc`: This means start the TypeScript compiler
- `lite`: This starts the file server to host the Angular app
- `typings`: This starts the typings tool
- `postinstall`: This calls the `typings install` method after the npm installation

At this point, it is safe to run the following command from the command line:

```
npm install
```

The npm manager will parse through the configuration file and download the required files for the application to run. If there are any errors, it will appear in red. Check the `package.json` file to make sure there are no typos and so on, or refer to the npm manual for more information. There may be warnings that appear, which it is usually okay to ignore.

Now that npm has completed the installation of modules, let's start by writing some code. Create a new subdirectory in the `angular2-helloworld` folder called `app` and run the following:

```
mkdir app  
cd app
```

Create a component file called `app.component.ts` and place the following code in it:

```
import {Component} from 'angular2/core';

@Component({
  selector: 'my-app',
  template: '<h1>Hello World !</h1>'
})
export class AppComponent { }
```

We will explain the app component file more in a later chapter. This file will be the root component of the application.

In this file, we have the `import` statement, which brings in the `angular2/core` module, and the `@Component` decorator, which brings in the metadata object. More about the component decorators will be discussed in a later chapter.

The `export class AppComponent { }` is an empty class that we will use for our main application to import; more on this will be discussed in a later chapter.

Now, we need to create a file that will let Angular load the component file we just created. Create a new file called `main.ts`, make sure that this file is in the `app` directory, and copy the following code into this file:

```
import {bootstrap} from 'angular2/platform/browser';
import {AppComponent} from './app.component';

bootstrap(AppComponent);
```

Bootstrap will handle the loading of the component based on the platform we decided to use. We will cover more on Bootstrap later.

Jump back to the `angular2-helloworld` directory by entering the following:

```
cd ..
```

Create a new file called `index.html` and load the following content into it

```
<html>
  <head>
    <title>Angular 2 Hello World</title>
    <meta name="viewport" content="width=device-width, initial-
      scale=1">

    <!-- 1. Load libraries -->
    <!-- IE required polyfills, in this exact order -->
```



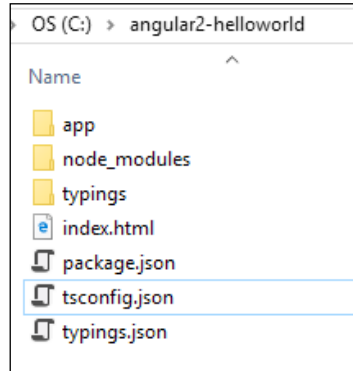
```
<script src="node_modules/es6-shim/es6-shim.min.js"></script>
<script src="node_modules/systemjs/dist/system-
polyfills.js"></script>
<script src="node_modules/angular2/es6/dev/src/testing/
shims_for_IE.js"></script>

<script src="node_modules/angular2/bundles/angular2-
polyfills.js"></script>
<script src="node_modules/systemjs/dist/system.src.js">
</script>
<script src="node_modules/rxjs/bundles/Rx.js"></script>
<script src="node_modules/angular2/bundles/angular2.dev.js">
</script>

<!-- 2. Configure SystemJS -->
<script>
  System.config({
    packages: {
      app: {
        format: 'register',
        defaultExtension: 'js'
      }
    }
  });
  System.import('app/main')
    .then(null, console.error.bind(console));
</script>
</head>

<!-- 3. Display the application -->
<body>
  <my-app>Loading...</my-app>
</body>
</html>
```

The folder structure should look similar to this:



Run this application by entering the following command in the command line:

```
npm start
```

If all went well, you will see the following pop up in your default browser. If your browser does not open automatically, open <http://localhost:3000/> to view **Hello World !**



More about the contents of the `index.html` file will be discussed in another chapter.

2

Components, Directives, and Templates

We have completed our Hello World! application. Now, let's expand this bit of code from the previous chapter and move into working with components, modules, and templates. In a new folder, let's create a new project and call it `angular2-todolist`.

Modify the `package.json` file so that the `name` property now says `angular2-todolist`, as follows:

```
"name": "angular2-todolist",
```

Modify the `index.html` file by changing the title from `Angular 2 Hello World to My Todo List`, as follows:

```
<title>Angular 2 - My Todo List</title>
```

Using components to reuse Angular tasks

Components are the way we build and specify different elements on a page. In Angular 1, we had to use directives, controllers, and scope to achieve the functionality we can easily find in a component.

Let's modify the component in `app/app.component.ts`. Change the code so that it looks similar to this:

```
import {Component} from 'angular2/core'  
  
@Component({  
  selector: 'my-component',  
  template: '<div><button (click)="sayMyName()">Do Something  
    Special</button></div>'
```

```
    })
    export class MyComponent {
      public name: String;
      constructor() {
        this.name = 'Angular 2 Rocks !';
      }
      sayMyName() {
        alert (this.name);
      }
    }
  }
}
```

We have modified the name of the component and the selector's name, so in the `index.html` file, change the name of the selector to `<my-component>`. It should look similar to this:

```
<my-component>Loading...</my-component>
```

In the `main.ts` file, change the code to match the following:

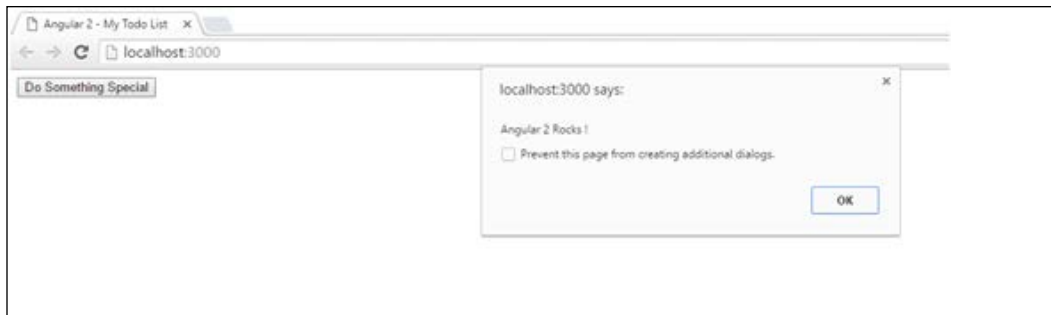
```
import {bootstrap}    from 'angular2/platform/browser';
import {MyComponent} from './app.component';

bootstrap(MyComponent);
```

At the command prompt, type `npm start`. The project should compile and open the browser displaying a button, as follows:

npm start

Our web page should open, and a button will be visible on the web page. Click on the button, and an alert dialog will show up with the text **Angular 2 Rocks !**



This is a basic component and its implementation. Let's move on to directives.

How do we use directives?

Create a new file in the `app` folder and call it `highlight.directive.ts`.

Copy the following code into this file:

```
import {Directive, ElementRef} from 'angular2/core';
@Directive({
  selector: '[myHighLight]'
})
export class HighlightDirective {
  constructor(el: ElementRef) {
    el.nativeElement.style.backgroundColor = 'orange';
  }
}
```

The first line of this file includes more symbols from the Angular library, as follows:

```
import {Directive, ElementRef} from 'angular2/core';
```

The `Directive` symbol is for the `@Directive` decorator, and `ElementRef` is to allow us to access the DOM element in the directive's constructor. If we want to do more with the directive—say, change the `onMouseOver` color of the selector—we will add attributes to the host tag.

Change the contents of `highlight.directive.ts` to the following:

```
import {Directive, ElementRef} from 'angular2/core';
@Directive({
  selector: '[myHighlight]',
  host: {
    '(mouseenter)': 'onMouseEnter()',
    '(mouseleave)': 'onMouseLeave()'
  }
})
export class HighlightDirective {
  private _el:HTMLElement;
  constructor(el: ElementRef) { this._el = el.nativeElement; }
  onMouseEnter() { this._highlight("yellow"); }
  onMouseLeave() { this._highlight(null); }
  private _highlight(color: string) {
    this._el.style.backgroundColor = color;
  }
}
```

The `host` property contains two methods that we will call to perform a CSS change on both `MouseEnter` and `MouseLeave`.

Templates and the power behind them

Angular uses templates to manage what the client sees in components. Templates contain the HTML code that is required for the view. Almost all HTML tags are allowed in templates, except the `<script>` tags that are basically ignored by the compiler. Other tags, such as `<html>`, `<body>`, and `<base>`, do not really apply to the template and are ignored. To add a template to the component, remove the `template` tag and add a new tag called `templateUrl`. The `templateUrl` tag will point to the path of your template.

Create a new file called `app.component.html` and copy the following content of the `template` tag into this file:

```
template: '<div><button (click)="sayMyName()">Do Something
Special</button></div>'
```

Rename the `template` tag `templateUrl` and add a property called `app/app.component.html`. Your `app.component.ts` file should look similar to this when it is complete:

```
import {Component} from 'angular2/core'
import {HighlightDirective} from './highlight.directive';

@Component({
  selector: 'my-component',
  templateUrl: 'app/app.component.html',
  directives: [HighlightDirective]
})
export class MyComponent {
  public name: String;
  constructor() {
    this.name = 'Angular 2 Rocks !';
  }
  sayMyName() {
    alert (this.name);
  }
}
```

With `npm` running in the background, all changes to the files should trigger an immediate change in your browser. Try changing the content of the HTML code to see the changes.

Data binding will be discussed more in the next chapter, but to introduce a simple concept, we will add a `H1` tag and put in the current date, which will be bound from within the component. Add a new property in the `app.component.ts` file called `today` and set the date to the current date.

The `export` part of the component will look similar this:

```
export class MyComponent {
  public name: String;
  public today: Date;

  constructor() {
    this.name = 'Angular 2 Rocks !';
    this.today = new Date();
  }
  sayMyName() {
    alert (this.name);
  }
}
```

We included a `date` variable and set its value in the constructor. Add a `H1` tag with a binding to this variable in the `app.component.html` file, as follows:

```
<h1>{{today | date:"fullDate"}}</h1>
```

The `|` sign is a pipe that replaces filters from Angular 1. Pipes, just like filters, help with formatting the outputs in our templates.

3

Metadata, Data Binding, and Services

Metadata tells Angular how to process a class. From our `app.component.ts` file, we just created a class. Decorators are a function, and they tell the class what to do, as follows:

```
@Component({
  selector: 'my-component',
  templateUrl: 'app/app.component.html',
  directives: [HighlightDirective]
})
export class MyComponent {
  public name: String;
  public today: Date;

  constructor() {
    this.name = 'Angular 2 Rocks !';
    this.today = new Date();
  }
  sayMyName() {
    alert (this.name);
  }
}
```

Here, `@Component` is the decorator. In this function, we will pass the parameters that form the configuration for the class. There are many parameters that can be passed to the decorator, as follows:

- `selector`: This is the CSS selector that Angular can bind to
- `templateUrl`: This is the location of the template file

- `directives`: This is an array of directives that can be bound to the template
- `providers`: This is an array of dependency injection providers for services

How do we use data binding?

Data binding allows data from the template to be bound to the component. There are four forms of data binding:

- To the DOM
- From the DOM
- To and from the DOM
- Two-way data binding combining properties and event binding in one notation using `ngModel`

Have a look at this code:

```
<h1>{{today | date:"fullDate"}}</h1>

<div>
  <button (click)="sayMyName()" myHighlight>Do Something
  Special</button>
</div>
```

In the `app.component.html` file, bind `{{today}}` to the `today` property in the `app.component.ts` file by one-way binding. The `sayMyName()` function is bound to the `click` event handler.

To demonstrate two-way data binding, change the `app.component.html` file to include `div` tags, as follows:

```
<div>
  <input [(ngModel)]="task">
</div>
<div>
  {{task}}
</div>
```

In the `app.component.ts` file, change the class to look similar to this:

```
export class MyComponent {
  public name: String;
  public today: Date;
  public task: string;
```

```
    constructor() {
        this.name = 'Angular 2 Rocks !';
        this.today = new Date();
        this.task = '';
    }
    sayMyName() {
        alert (this.name);
    }
}
```

In this example, we bound the input control to the `task` property. As the user types, the contents of the input will appear on the page.

Creating services

Services in Angular2 is a class with a defined function. Anything can be a service. Let's create a service that reads a list of tasks stored in a JS file. Create a file called `task.service.ts` in the app folder and add the following to it:

```
import {Injectable} from 'angular2/core';

@Injectable()

export class TaskService {
}
```

We included the Angular `Injectable` function and applied this function as an `@Injectable()` decorator. TypeScript will see the `Injectable` decorator and will emit metadata that Angular may need to inject other dependencies into this service.

Let's create a model for the tasks. Create a new file called `tasks.ts` in the app folder; in this file, we will create a model for our tasks, as follows:

```
export class Task {
    id: number;
    name: string;
}
```

Create a new file with a JavaScript array of tasks. Call this file `sample.tasks.ts`. Put the following content into the file:

```
import {Task} from './tasks';
export var TASKS: Task[] = [
    {"id": 11, "name": "Buy Bread"},
    {"id": 12, "name": "Buy Milk"},
]
```

```
        {"id": 13, "name": "Buy Soap"}
    ];
```

Note that we included our model in this file and created an array with this model. This array is exported in `TASKS`, as follows:

```
export var TASKS: Task[] = [
```

In our `tasks.service.ts` file, we will include this class and create a GET method. Replace the content of the service file with the following code block:

```
import {TASKS} from './sample.tasks';
import {Injectable} from 'angular2/core';

@Injectable()
export class TaskService {
    getTasks() {
        return TASKS;
    }
}
```

We created the service and the data with the model. We must now modify the template and component. Modify the `app.component.ts` file to match the following:

```
import {Component} from 'angular2/core';
import {HighlightDirective} from './highlight.directive';
import {TaskService} from './task.service';
import {Task} from './tasks';

@Component({
    selector: 'my-component',
    templateUrl: 'app/app.component.html',
    directives: [HighlightDirective],
    providers: [TaskService]
})
export class MyComponent {
    public name: String;
    public today: Date;
    public task: String;
    public taskList: Task[];

    constructor(private _taskService: TaskService) {
```

```
        this.name = 'Angular 2 Rocks !';
        this.today = new Date();
        this.task = '';
        this.taskList = this._taskService.getTasks();

    }
    sayMyName() {
        alert (this.name);
    }
}
```

We have included our service and task as an `import`. Here is a snippet of what we have changed:

```
import {TaskService} from './task.service';
import {Task} from './tasks';
```

To make the service available, we have to enable it as a provider. We will do this in the `@Component` decorator, as follows:

```
providers: [TaskService]
```

This exposes the service to the component. We will create an array that will contain `taskList` and make it of the `Task` type:

```
export class MyComponent {
    public name: String;
    public today: Date;
    public task: String;
    public taskList: Task[];

    constructor(private _taskService: TaskService) {
        this.name = 'Angular 2 Rocks !';
        this.today = new Date();
        this.task = '';
        this.taskList = this._taskService.getTasks();
    }
    sayMyName() {
        alert (this.name);
    }
}
```

The constructor includes the definition to connect to the service and assign it to a private variable. The underscore (`_`) is used so that we can quickly identify that the data is coming externally. The `this._taskService.getTasks()`; method allows us to get the tasks from the service file.

In the `app.component.html` file, add the following code:

```
<div>
  <ul>
    <li *ngFor="#item of taskList">{{item.name}}</li>
  </ul>
</div>
```

The `*ngFor` directive used in the component's template file allows us to iterate quickly through the `taskList` array.

We have now completed metadata, data binding, and services.

4

Dependency Injection

Dependency injection has been given its own chapter as it is one of the most powerful features in Angular.

The Angular 1.x implementation of dependency injection has some limitations:

- **Internal cache:** Dependencies are singletons; they are only created once per life cycle
- **Namespace collision:** We can only have one type of this
- **Tightly coupled:** This is built into the Angular 1.x framework

Let's use a new example to explain dependency injection. Let's create a class that creates a car. In order for us to do this, a normal way would be the following:

```
class Car {
  constructor() {
    this.engine = new Engine();
    this.tyres = Tyres.getInstance();
  }
}
```

This method is okay, but we are getting the engine from an `Engine` constructor and the tyres from a singleton. If we need to test this code by replacing `Engine` with a `MockEngine` constructor, then we will have to write new code. If we build code that we can test with, then we will inadvertently build reusable code. One way of making this code more reusable is to use the TypeScript constructor to pass in the type and value. Here is an example:

```
class Car {
  constructor(engine, tyres) {
    this.engine = engine;
    this.tyres = tyres;
  }
}
```

We moved the dependency creation out of the constructor and extended the constructor function to expect all the necessary dependencies. Now, we have not hardcoded any implementations in the code either; we moved it to the constructor. To create a new car now, all we need to execute is the following:

```
var car = new Car(  
  new Engine(),  
  new Tyres()  
);
```

Now, if we need to test our code, we can send in any mock dependencies. Take the following for example:

```
var car = new Car(  
  new MockEngine(),  
  new MockTyres()  
);
```

Now, this is dependency injection.

In Angular 2, dependency injection consists of the following:

- **Injector:** This is the object that exposes APIs
- **Provider:** This tells the injector how to create an instance of a dependency
- **Dependency:** This is the type of which an object should be created

In Angular 2, to create the same object, we would define it this way:

```
import { Injector } from 'angular2/core';  
  
var injector = Injector.resolveAndCreate([  
  Car,  
  Engine,  
  Tyres  
]);  
  
var car = injector.get(Car);
```

Injector is imported from Angular 2, which exposes APIs to help create Injector.resolveAndCreate(), which is a factory function that creates an injector and takes a list of providers.

In TypeScript, we will define the Car class as follows:

```
class Car {  
  constructor(engine: Engine, tires: Tires, doors: Doors) {
```

```
    ...  
  }  
}
```

By creating dependency injection in this way, we can remove the problems of name collisions. This structured format also removes all of the issues with Angular 1.x.

In the next chapter, we will use all the previous chapters to build an application (and make it look a little pretty, too).

5

Wrapping Up

Let's expand our to-do application into a working prototype. We have created everything we need from the first three chapters to create a functioning application. We will modify the services task to create the `add` and `remove` methods, then apply a Bootstrap theme to make the view pretty.

Start by modifying the `task.service.ts` file located in the `app` folder. Let's create a create method by changing the file to match the following:

```
import {TASKS} from './sample.tasks';
import {Injectable} from 'angular2/core';

@Injectable()
export class TaskService {
  getTasks() {
    return TASKS;
  };
  addTask(task) {
    TASKS.push(task);
  };
  deleteTask (task) {
    for (var i=0; i<TASKS.length; i++) {
      if (TASKS[i].id === task.id) {
        TASKS.splice(i,1);
        break;
      }
    }
  };
}
```

We have added two new methods: one called `addTask`, which accepts `task` as a parameter, and `deleteTask`, which also accepts `task` as a parameter. The `addTask` method pushes the new task into the array, and the `deleteTask` method removes the task from the array based on the ID of the `task` object sent.

The next file we will modify is the `index.html` file, which is located outside the app folder. We will add the Bootstrap CSS theme from the Bootstrap CDN.

Modify the contents of the `index.html` file to match the following:

```
<html>
  <head>
    <title>Angular 2 - My Todo List</title>
    <meta name="viewport" content="width=device-width
      , initial-scale=1">

    <!-- 1. Load libraries -->
    <!-- IE required polyfills, in this exact order -->
    <script src="node_modules/es6-shim/es6-shim.min.js"></script>
    <script src="node_modules/systemjs/dist/
      system-polyfills.js"></script>
    <script src="node_modules/angular2/es6/dev/src/
      testing/shims_for_IE.js"></script>

    <script src="node_modules/angular2/bundles/angular2-
      polyfills.js"></script>
    <script src="node_modules/systemjs/dist/
      system.src.js"></script>
    <script src="node_modules/rxjs/bundles/Rx.js"></script>
    <script src="node_modules/angular2/bundles/
      angular2.dev.js"></script>

    <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/
      bootstrap/3.3.6/css/bootstrap.min.css" integrity="sha384-
      1q8mTJOASx8j1Au+a5WDVnPi2lkFfwwEAa8hDDdjZlpL
      egxhjVME1fgjWPGmkzs7" crossorigin="anonymous">
    <!-- 2. Configure SystemJS -->
    <script>
      System.config({
        packages: {
          app: {
            format: 'register',
            defaultExtension: 'js'
          }
        }
      });
      System.import('app/main')
        .then(null, console.error.bind(console));
    </script>
  </head>
```

```
<!-- 3. Display the application -->
<body>
  <my-component>Loading...</my-component>
</body>
</html>
```

Note the addition of the Bootstrap CSS theme file located on the Bootstrap CDN.

We will next modify the `app.component.ts` file located in the `app` folder to match the following:

```
import {Component} from 'angular2/core';
import {HighlightDirective} from './highlight.directive';
import {TaskService} from './task.service';
import {Task} from './tasks';
import {OnInit} from 'angular2/core';

@Component({
  selector: 'my-component',
  templateUrl: 'app/app.component.html',
  directives: [HighlightDirective],
  providers: [TaskService]
})
export class MyComponent {
  public name: String;
  public today: Date;
  public task: String;
  public taskList: Task[];

  constructor(private _taskService: TaskService) {
    this.name = 'Angular 2 Rocks !';
    this.today = new Date();
    this.task = '';
    this.taskList = this._taskService.getTasks();
  }
  deleteTask(item) {
    this._taskService.deleteTask(item);
  }
  addNewTask() {
    this._taskService.addTask ( { id: Math.floor
      ((Math.random() * 10) + 10), name : this.task });
  }
}
```

```
        this.task = '';  
    }  
}
```

We removed the `sayMyName()` method and added `deleteTask` and `addNewTask` to match the new methods in the services. Note that this is a demo app, so there is no verification of IDs and so on. In a production environment, more data verification will be required to get these methods working better.

Next, modify the `app.component.html` file located in the `app` folder to match the following:

```
<h1>{{today | date:"fullDate"}}</h1>  
<div>  
  <div class="form-group">  
    <label for="usr">New Task:</label>  
    <input [(ngModel)]="task" class="form-control">  
  </div>  
  <button (click)="addNewTask()" class='btn-  
    default' myHighlight>Add Task</button>  
</div>  
<div>  
  <div class="list-group">  
    <a href="#" (click)="removeTask(item)" *ngFor="#item of  
      taskList" class="list-group-item">{{item.name}}</a>  
  </div>  
</div>
```

We removed the list and added a button and a `href` tag with Bootstrap themes attached to it.

When this is complete, save all the files and open your browser to view the new to-do app.

Summary

In the first chapter, you learned how to set up our environment for Angular 2. We loaded the Angular modules via `npm` and `Node.js` and got our Hello World! application running. The most important part of this chapter was showing off TypeScript with Angular 2.

The second and third chapters showed us the difference between Angular 1 and 2 using components, directives, and templates with metadata, data binding, and services. This formed the basic part of our to-do list application.

In the fourth chapter, we explained dependency injection in detail by creating a new example using a car. Dependency injection is one of the most important parts of Angular, so it had its own dedicated section.

In the fifth chapter, we wrapped up our to-do list application and added a bit of Bootstrap styling to it.

Angular 2 has changed everything about Angular 1.x, making JavaScript applications much more fun and structured. While this is a short overview of what Angular 2 offers, a more detailed specification can be found at <https://angular.io/docs/ts/latest/>.

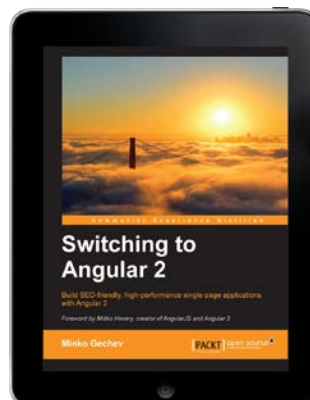
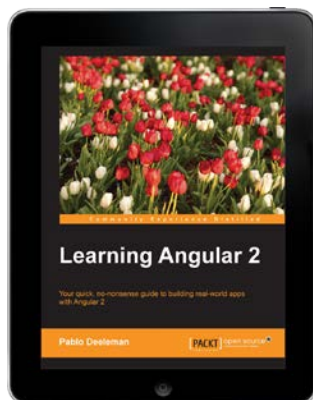
All code has been published to <https://github.com/Manickum/Angular2-PACKT>.

Feel free to contribute, commit, and send comments. Happy Angular 2!

What to do next?

Broaden your horizons with Packt

If you're interested in Angular 2, then you've come to the right place. We've got a diverse range of products that should appeal to budding as well as proficient specialists in the field of Angular 2.



To learn more about Angular and find out what you want to learn next, visit the AngularJS technology page at <https://www.packtpub.com/tech/AngularJS>.

If you have any feedback on this eBook, or are struggling with something we haven't covered, let us know at customer@packtpub.com.

Get a 50% discount on your next eBook or video from www.packtpub.com using the code:

