Angular Development with TypeScript

SECOND EDITION

Yakov Fain
Anton Moiseev

MEAP

MANNING

MEAP Edition
Manning Early Access Program
Angular Development with Typescript
Second Edition
Version 6

For more information on this and other Manning titles go to
www.manning.com

welcome

Thank you for purchasing the final MEAP for *Angular Development with TypeScript, Second Edition*. We wrote this book for developers who are open to learning new languages and frameworks to become more productive with developing Web applications. We expect the reader to know the basics of HTML, CSS, and JavaScript (based on ECMAScript 5 spec).

This MEAP (v6) adds Chapter 15 "Maintaining App State with ngrx".

Chapter 1 starts with a high level overview of the Angular architecture, and then we introduce you to Angular CLI – the tool that will generate a new Angular project in less than a minute so you can see the first app running right away. Then we discuss different ways of compiling Angular projects. Finally, we introduce a sample ngAuction application that you'll be developing with us starting from Chapter 2.

Chapter 2 will get you familiar with the main artifacts of Angular: components, services, directives, pipes and modules. At the end of Chapter 2 we provide detailed instructions on creating the first version of the sample ngAuction app. From this point on, most chapters will end with a hands-on section where you'll be working on this sample online auction, gradually adding new features or even doing a complete re-write.

Chapter 3 introduces Angular router used to arrange client-side navigation in a single-page app. You'll learn how to configure routes and how to pass parameters between the routes. You'll also see how to create component hierarchies where both parents and children have their own routes.

Chapter 4 covers more advanced router's API. You'll learn how to protect the routes and create components with multiple router outlets. We'll show you how to use the router for loading your app modules lazily (on demand). At the end of this chapter you'll continue working on the ngAuction under our guidance.

Chapter 5 is about Dependency Injection (DI). We'll start with an overview of DI as a design pattern, explaining the benefits of having a framework to create and inject object instances. You'll learn the roles of providers and injectors and how to easily swap the object being injected if need be. At the end of this chapter, you'll make a small facelift to ngAuction using the Angular Material library of UI components.

Chapter 6 is about working with observable streams of data, and prior to reading this chapter, you need to become familiar with the basics of RxJS library covered in Appendix D. We'll start with showing you how to treat events with observables. Then you'll see various Angular APIs that offer ready-to-use observable streams. You'll also learn how discard unwanted HTTP requests with the RxJS switchMap operator.

Chapter 7 introduces the Flex Layout library that will allow you to design UI layouts that will adapt to the width of the user devices. You'll also see how to use the ObservableMedia service that allows to apply different CSS depending on the screen size. At the end of this

chapter, we'll start re-writing ngAuction from scratch using the Angular Material and Flex Layout libraries.

Chapter 8 is about arranging inter-component communications in a loosely-coupled manner. You'll learn about the input and output properties of Angular components and will see how components A and B can communicate without knowing about each other. You'll see how components can communicate with each other via a common parent or via an injectable service.

Chapter 9 includes an overview of the component lifecycle and the change detection mechanism. At the end of the chapter, we'll continue working on ngAuction and will add a product view to this app.

Chapter 10 will get you familiar with Angular Forms API. You'll learn the difference between the template-driven and reactive forms. You'll see how to access the form data entered by the user as well as update the forms programmatically.

Chapter 11 continues coverage Forms API. Here' you'll learn how to validate the form's data. You'll learn how to use built-in and create custom validators for booth template-driven and reactive forms. Finally, we'll continue working on ngAuction. This time we'll add a search form so the user can search for products. You'll have a chance to apply the materials from chapters 10 and 11 in practice.

Chapter 12 is about communicating with web servers using HTTP. Angular offers a HttpClient service with rich API. You'll see how to issue GET and POST requests, intercept all HTTP requests and responses to implement cross-cutting concerns. As additional bonus, you'll learn how to write web servers using Node and Express frameworks. We'll use these servers so Angular clients have someone to talk to and will show you how to write scripts for deploying Angular apps under web servers.

Chapter 13 explains how to write Angular applications that communicate with server using the WebSocket protocol, which is an efficient and a low-overhead way or communication. One of the most valuable features of WebSocket communications is that the server can initiate the data push to the client when an important even happens without waiting for the client to request the data. You'll see a practical use of the WebSocket communication in our ngAuction, which implements the bidding notifications over the WebSockets. The new version of ngAuction comes as two separate projects – one with the client-side code and another with the server's code.

Chapter 14 is about testing. We'll introduce you to unit testing with Jasmine and end-to-end testing with Protractor. We'll also show you how to test the search workflow in ngAuction with Protractor.

Chapter 15 is about maintaining the app state in Redux style using the ngrx library. It starts with explaining the priinciples of Redux, and then you'll see how ngrx implements these principles in Angular apps. The chapter ends with the code review of the final version of ngAuction that uses ngrx for state management.

This book comes with four appendixes. They cover a new syntax of ECMAScript, TypeScript, basics of Node package manager (npm), and the library of reactive extensions RxJS.

Appendix A contains an overview of the syntax introduced in ECMAScript 6, 7, and 8. You'll learn how to use classes, fat arrow functions, spread and rest operators, what destructuring is, and how to write asynchronous code as if it's synchronous with the help of async-await keywords. At the time of this writing, ECMAScript 6 is supported by most of the major Web browsers.

Appendix B is an overview of the syntax of TypeScript, which is a superset of JavaScript. TypeScript will increase your productivity of creating JavaScript apps. Not only will you learn how to write classes, interfaces, generics, but also how compile the TypeScript code into JavaScript that can be deployed in all Web browsers today.

Appendix C is a brief overview of npm and Yarn package managers, which are used to install JavaScript packages, libraries and frameworks on developers' machines. You'll understand how project dependencies are configured in the file package.json and what semantic versioning is about.

Appendix D is an introduction to RxJS, a popular library of reactive extensions. You'll learn the roles of observables, observers, and subscribers and how to compose RxJS operators to handle data streams in a functional way. While the RxJS library can be used with any JavaScript app, it's a crucial part of the Angular framework so understanding the main concepts of RxJS is a must.

The code samples were developed using Angular 5, and you can find them at https://github.com/Farata/angulartypescript. After Angular 6 is released, we'll update the code samples.

Enjoy the reading and please provide your feedback at Author Forum. We like to fail fast.

—Yakov Fain and Anton Moiseev.

brief contents

1 Introducing Angular

This chapter covers

- A high-level overview of the Angular framework
- How to generate a new project with Angular CLI
- Getting started with Angular modules and components
- Introducing the sample application ngAuction

Angular is an open source JavaScript framework maintained by Google. It's a complete rewrite of its popular predecessor, AngularJS. The first version of Angular was released in September of 2016 under the name Angular 2. Shortly after, the digit 2 was removed from the name, and now it's just Angular. Twice a year, the Angular team makes major releases of this framework. Future releases will include new features, will perform better, and generate smaller code bundles, but the architecture of the framework most likely will remain the same.

Angular applications can be developed in JavaScript (using the syntax of ECMAScript 5 or later versions) or TypeScript. In this book we'll use TypeScript; our reasons for this are explained in Appendix B.

NOTE

Prerequisites

In this book, we expect you to know the syntax of JavaScript and HTML and to understand what web applications consist of. We also assume that you know what CSS is. If you're not familiar with the syntax of TypeScript and the latest versions of ECMAScript, we suggest you read Appendixes A and B first, and then continue reading from this chapter on. If you're new to developing using Node.js tooling, read the appendix C.

NOTE	**Code samples**
	All code samples in this book are based on Angular 5 released in November of 2017. Download the code samples from github.com/Farata/angulartypescript - we'll be providing instructions on how to run each code sample starting from chapter 2.

We'll start this chapter with a very brief overview of the Angular framework. Then we'll start coding. Actually, we'll generate our first project using the Angular CLI tool. Finally, we'll introduce the sample application ngAuction that we're going to build in this book.

1.1 Why select Angular for Web development

Web developers use different JavaScript frameworks and libraries, and the most popular are Angular, React, and Vue. You can find lots of articles and blog posts comparing them, but such comparison is not justified because React and Vue are just libraries that don't offer a full solution for developing and deploying a complete Web application, while Angular does.

If you pick React or Vue for your project, you'll also need to select other products that support routing, dependency injection, forms, bundling and deploying the app and more. In the end, your app will consist of multiple libraries and tools picked by a senior developer or an architect. If this developer decides to leave the project, finding replacement won't be easy as the new hire may not be familiar with all of the libraries and tools used in the project.

The Angular framework is a platform that includes all you need for developing and deploying a Web app, batteries included. Replacing one Angular developer with another is easy as long as the new person knows Angular.

From the technical perspective, we like Angular because it's a feature-complete framework that allows you to do the following right out of the box:

- Generate a new single-page web app in seconds using Angular CLI
- Create a web app that consists of a set of components that can communicate with each other in a loosely-coupled manner
- Arrange the client-side navigation using the powerful router
- Inject and easily replace services, which are classes where you implement data communication or other business logic
- Arrange state management via injectable singleton services
- Cleanly separate the UI and business logic
- Modularize your app so only the core functionality is loaded on app startup while other modules are loaded on demand
- Creating modern-looking UI using the Angular Material library
- Implement reactive programming where your app components do not pull the data that may not be ready yet, but simply subscribe to data source and get notifications where the data is available

Having said that, we need to admit that there is one advantage that React and Vue have over Angular. While Angular is a good fit for creating single-page apps, where the entire app is developed in this framework, the code written in React and Vue can be included into any Web app regardless of what other frameworks were used for development of any single-page or multi-page web app.

This advantage will disappear when the Angular team releases a new module currently known as Angular Elements (see github.com/robwormald/angular-elements). Then you'll be able to package your Angular components as custom elements (see developer.mozilla.org/en-US/docs/Web/Web_Components/Custom_Elements) that can be embedded into any existing web app written in JavaScript with or without any other libraries.

1.2 Why develop in TypeScript and not in JavaScript

You may be wondering, why not just develop in JavaScript? Why do we need to use another programming language if JavaScript is already a language? You wouldn't find articles about additional languages for developing Java or C# applications, would you?

The reason is that developing in JavaScript isn't overly productive. Say a function expects a `string` value as an argument, but the developer mistakenly invokes it by passing a numeric value. With JavaScript, this error can be caught only at runtime. Java or C# compilers won't even compile code that has mismatching types, but JavaScript is forgiving because it's a dynamically typed language.

Although JavaScript engines do a decent job of guessing the types of variables by their values, development tools have a limited ability to help you without knowing the types. In mid- and large-size applications, this JavaScript shortcoming lowers the productivity of software developers.

On larger projects, good IDE context-sensitive help and support for refactoring are very important. Renaming all occurrences of a variable or function name in statically typed languages is done by IDEs in a split second, but this isn't the case in JavaScript, which doesn't support types. If you make a mistake in a function or a variable name, it's displayed in red. If you pass the wrong number of parameters (or wrong types) to a function, the wrong ones show in red. IDEs also offer great context-sensitive help. TypeScript code can be refactored by IDEs. TypeScript follows the ECMAScript 6, 7, and 8 specifications and adds to them types, interfaces, decorators, class member variables (fields), generics, enums, the keywords `public`, `protected`, and `private` and more. Check the TypeScript "Roadmap" on GitHub at github.com/Microsoft/TypeScript/wiki/Roadmap to see what's coming in the future releases of TypeScript.

TypeScript interfaces allow you to declare custom types. Interfaces help prevent compile-time errors caused by using objects of the wrong types in your application.

The generated JavaScript code is easy to read, and it looks like hand-written code.

The Angular framework itself is written in TypeScript, and most of the code samples in the Angular documentation (see angular.io), articles, and blogs are given in

TypeScript.

In 2017, StackOverflow developers survey (bit.ly/2qPK9dw) shows TypeScript as the third most loved language and the sixth most wanted. If you prefer to see a more scientific proof that TypeScript is more productive compared to JavaScript, read the study "To Type or Not to Type: Quantifying Detectable Bugs in JavaScript" available at earlbarr.com/publications/typestudy.pdf.

SIDEBAR

From the authors real-world experience

We work for a company, Farata Systems, that over the years developed pretty complex software using the Adobe Flex (currently Apache Flex) framework. Flex is a very productive framework built on top of the strongly typed compiled ActionScript language, and the applications are deployed in the Flash Player browser plugin (a virtual machine).

When the web community started moving away from using browser plugins, we spent two years trying to find a replacement for the Flex framework. We experimented with different JavaScript-based frameworks, but the productivity of our developers seriously suffered. Finally we saw a light at the end of the tunnel with a combination of the TypeScript language, Angular framework, and a UI library Material Design.

1.3 A high-level overview of Angular

Angular is a component-based framework, and any Angular app is a tree of components (think views). Each view is represented by instances of component classes. An Angluar app has one root component, which may have child components. Each child component may have its own children and so on.

Say, you'd need to re-write the Twitter app in Angular. Take a prototype from your web designer and start by splitting it into components as shown in figure 1.1.

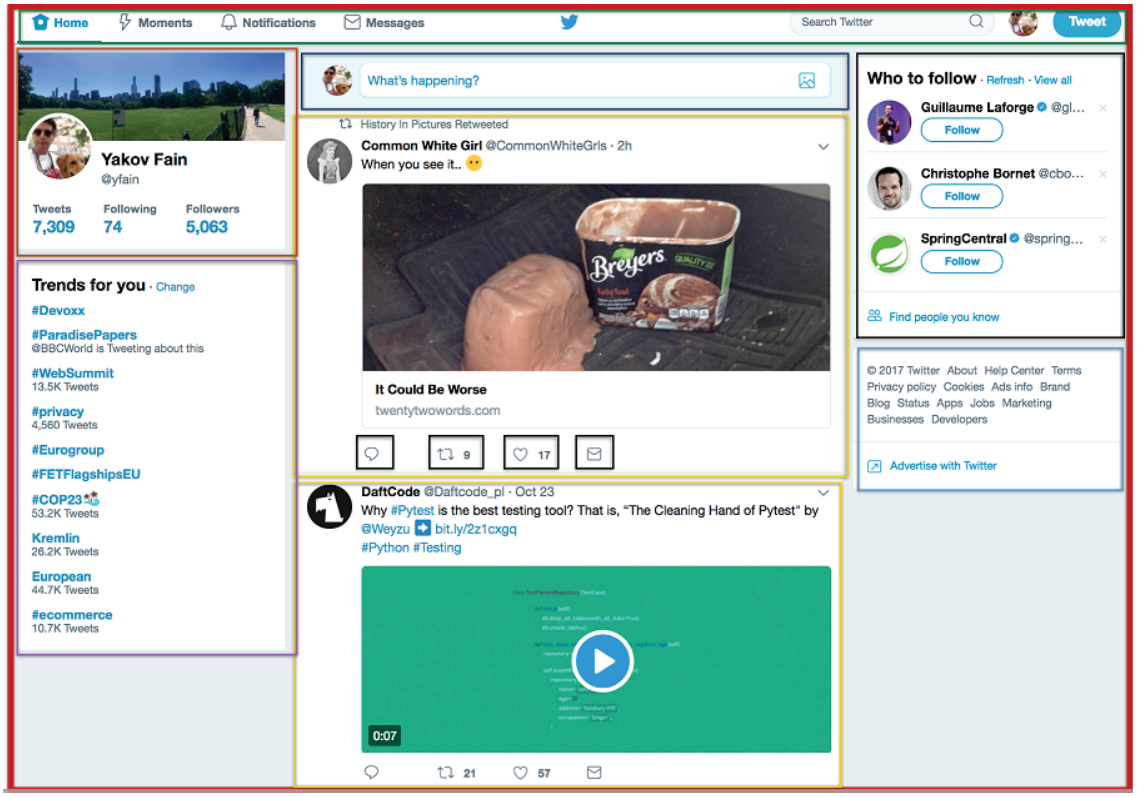

Figure 1.1 Splitting a prototype into components

The top level component with the thick border encompasses multiple child components. In the middle, you can see a New Tweet component on top and two instances of the Tweet component, which in turn has child components for reply, re-tweet, like, and direct messaging.

A parent component can pass the data to its child by binding the values to the child's component property. A child component has no knowledge of where the data came from. A child component can pass the data to its parent (without knowing who the parent is) by emitting events. This architecture makes components self-contained and reusable.

When writing in TypeScript, a component is a class annotated with a decorator `@Component()`, where you specify the component's UI (we explain decorators in the section B.10 "Decorators" in Appendix B).

```
@Component({
    ...
}
export class AppComponent {
    ...
})
```

Most of the business logic of your app is implemented in services, which are classes without UI. Angular will create instances of your service classes and will inject them into your components. Your component may depend on services, and your services may

depend on other services. A service is just a class that implements some business logic. Angular injects services into your components or other services using the dependency injection (DI) mechanism that we'll cover in chapter 5.

Components are grouped into Angular modules. A module is a class decorated with `@NgModule()`. A typical Angular module is a small class which usually has an empty body unless you want to write code that manually bootstraps the application, e.g. if an app includes a legacy AngularJS app. The `@NgModule()` decorator lists all components and other artifacts (e.g. services, directives, et al.) that should be included in this module, for example:

Listing 1.1 A module with one component

```
@NgModule({
  declarations: [
    AppComponent                    (1)
  ],
  imports: [
    BrowserModule
  ],
  bootstrap: [AppComponent]         (2)
})
export class AppModule { }
```

(1) Declare that AppComponent belongs to this module

(2) Declare that AppComponent is a root component

To write a minimalistic Angular app, you can create one `AppComponent`, and list it in the `declarations` and `bootstrap` properties of `@NgModule()`. A typical module lists several components, and the root component is specified in the `bootstrap` property of the module. The code sample above also lists `BrowserModule`, which is a must for apps that run in a browser.

A component is the centerpiece of the Angular architecture. Figure 1.2 shows a high-level diagram of a sample Angular application that consists of four components and two services, and all of them are packaged inside a module. Angular injects the Angular's `HttpClient` service into your app's `Service1`, which in turn is injected into the `GrandChild1` component.

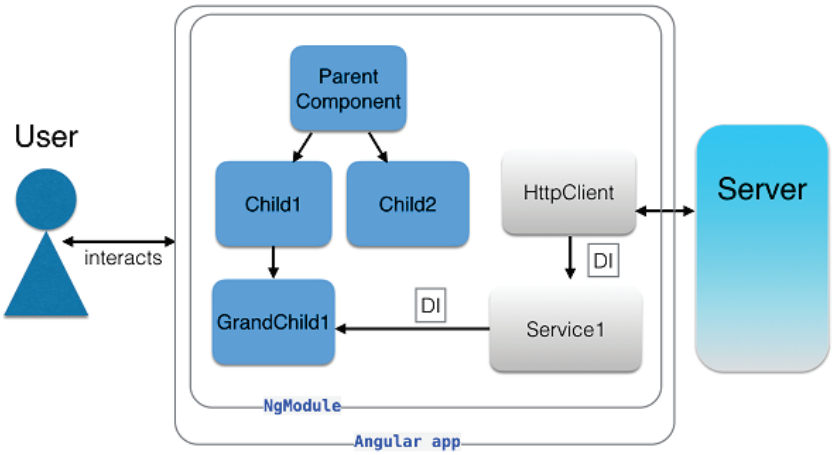

Figure 1.2 Sample architecture of an Angular app

The HTML template of each component is either inlined inside the component (the `template` property of `@Component()`) or in the file referenced from the component using the `templateUrl` property. The latter option offers a clean separation between the code and the UI. The same applies to styling components. You can either inline the styles using `styles` property, or provide the location of your CSS file(s) in `stylesURLs`. The following code snippet shows the structure of some search component:

Listing 1.2 A structure of a sample component

```
@Component({
  selector: 'app-search',                    (1)
  templateUrl: './search.component.html',    (2)
  styleUrls: ['./search.component.css']      (3)
})
export class SearchComponent {
  // Component's properties and methods go here
}
```

(1) Other components' templates can use the tag <app-search>

(2) The template's code is in this file

(3) Component's styles are in this file (could be more than one)

The value in the `selector` property defines the name of the tag that can be used in the other component's template. For example, the root app component can include a child search component as follows:

Listing 1.3 Using the search component in another one

```
@Component({
  selector: 'app-root',
  template: `<div>
              <app-search></app-search>      (1)
            </div>`,
  styleUrls: ['./app.component.css'],
})
```

```
export class AppComponent {
    ...
}
```

❶ The UI of the AppComponent includes the UI of the SearchComponent

In the above example, we use an inline template. Note that we use the back-tick symbols instead of quotes for a multi-line template (see the section A.3 "Template literals" in Appendix A).

The Angular framework is a great fit for developing single-page applications (SPA), where the entire browser's page is not getting refreshed and only a certain portion of the page (view) may be replacing another as the user navigates through your app. Such client site navigation is arranged with the help of the Angular router. If you want to allocate an area within a component's UI for rendering its child components, you'll use a special tag `<router-outlet>`. For example, on the app start, you may display the Home component in this outlet, and if the user clicks on the link Products, the outlet content will be replaced by the Product component.

To arrange navigation within a child component, you can allocate the `<router-outlet>` area in the child as well. We'll explain how the router works in Chapters 3 and 4.

SIDEBAR

UI components for Angular apps

The Angular team released a library of UI components called Angular Material (see material.angular.io). At the time of this writing, it has 30 well-designed UI components based on the Material Design guidelines (see material.io/guidelines). We recommend using Angular Material components in your projects, and if you'll need more components, in addition to Angular Material use one of the third-party libraries like PrimeNG, Kendo UI, DevExtreme, or others. You can also use a very popular Bootstrap library with Angular applications, and we'll show you how to do this in the ngAuction included in chapter 2. Staring from chapter 7, we'll re-write ngAuction replacing Bootstrap components with the Angular Material ones.

SIDEBAR

Angular for mobile devices

Angular's rendering engine is a separate module, which allows third-party vendors to create their own rendering engine that targets non-browser-based platforms. The TypeScript portion of the components remains the same, but the content of the `template` property of the `@Component` decorator may contain XML or another language for rendering native components.

As an example, you can write a component's template using XML tags from the NativeScript framework, which serves as a bridge between JavaScript and native iOS and Android UI components. Another custom UI renderer allows you to use Angular with React Native, which is an alternative way of creating native (not hybrid) UIs for iOS and Android.

We stated earlier that a new Angular app can be generated in seconds. Let's see how the tool called Angular CLI does it.

1.4 Introducing Angular CLI

Angular CLI is a tool for managing Angular projects throughout the entire life-cycle of an application. It serves as a code generator that greatly simplifies the process of new project creation as well as the process of generating new components, services, and routes in existing app. You can also use Angular CLI for building code bundles for dev and prod deployment. Angular CLI not only will generate a boilerplate project for you, but will also install Angular framework and all its dependencies.

Angular CLI became a de-facto standard way of starting new Angular projects. To install Angular CLI globally on your computer, run the following command in the Terminal window:

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

After the installation is complete, Angular CLI is ready to generate a new Angular project.

1.4.1 Generating a new Angular project

CLI stands for command line interface, and after installing Angular CLI, you can run the `ng` command from your Terminal window. Angular CLI understands many command line options and you can see all of them by running the `ng help` command in the Terminal window. We'll start with generating a new Angular project with the `ng new` command. Let's create a new project called hello-cli:

```
ng new hello-cli
```

This command will create a directory hello-cli and will generate a project with one module, one component, and all required configuration files including the file package.json, which includes all project dependencies (see Appendix C for details). After generating these files, Angular CLI will start npm to install all dependencies specified in package.json. When this command completes, you'll see a new directory hello-cli as shown in figure 1.3:

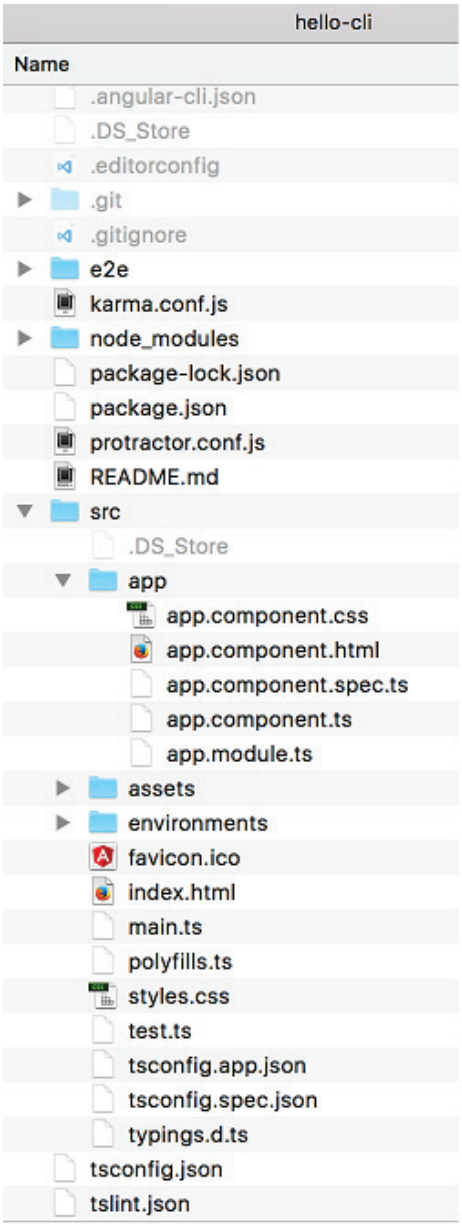

Figure 1.3 A newly generated Angular project

TIP	Say you have an Angular 4 project and want to switch to the latest version of Angular. You don't need to modify dependencies in the file package.json manually. Just run the `ng update` command, and all dependencies in package.json will be updated, assuming that you have the latest version of Angular CLI installed.

We'll review the content of the directory hello-cli in chapter 2 but let's build and run this project. In the Terminal window change to the hello-cli directory and run the following command:

```
ng serve
```

Angular CLI will spend about 10-15 seconds to compile TypeScript into JavaScript

and build the application bundles. Then Angular CLI will start its dev server ready to serve this app on port 4200. Your Terminal output may look as shown in figure 1.4.

```
/Users/yfain11/hello-cli
MacBook-Pro-8:hello-cli yfain11$ ng serve
** NG Live Development Server is listening on localhost:4200, open your browser
on http://localhost:4200/ **
Date: 2017-11-02T10:11:19.984Z                                              -
Hash: 6a0410d7576a15d5375e
Time: 5746ms
chunk {inline} inline.bundle.js (inline) 5.79 kB [entry] [rendered]
chunk {main} main.bundle.js (main) 20.2 kB [initial] [rendered]
chunk {polyfills} polyfills.bundle.js (polyfills) 548 kB [initial] [rendered]
chunk {styles} styles.bundle.js (styles) 33.5 kB [initial] [rendered]
chunk {vendor} vendor.bundle.js (vendor) 7.02 MB [initial] [rendered]

webpack: Compiled successfully.
```

Figure 1.4 Building the bundles with ng serve

Now open your Web browser at localhost:4200 and you'll see the landing page of your app as shown in figure 1.5.

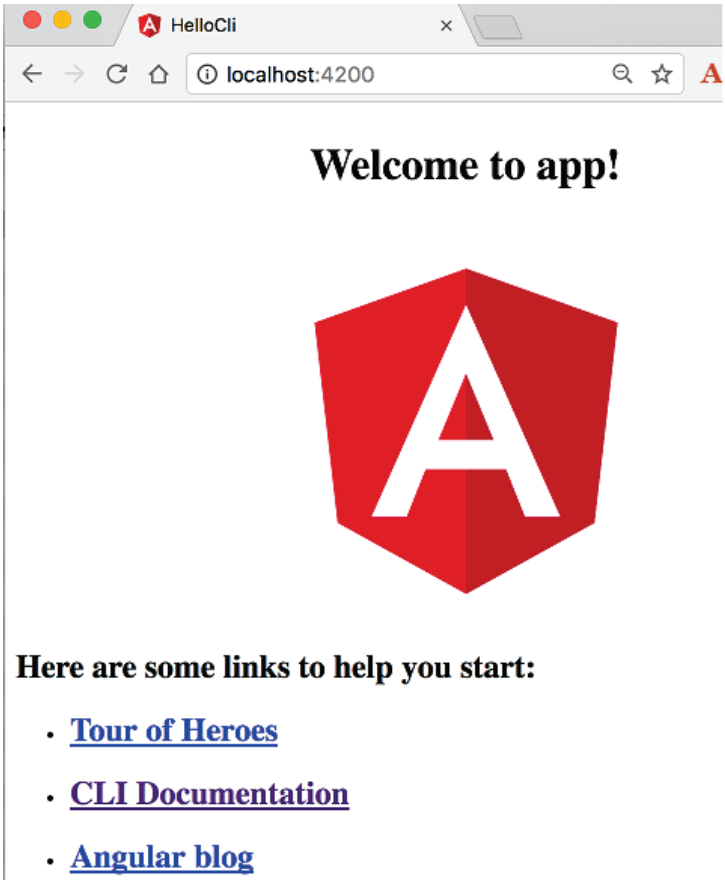

Figure 1.5 Running the app in the browser

Congratulations! You just created, configured, built, and ran your first Angular app without writing a single line of code!

SIDEBAR **Webpack and Angular CLI**

Currently, Angular CLI uses Webpack (see webpack.js.org) to build the bundles and Webpack Dev Server to serve the app. When you run `ng serve`, Angular CLI runs Webpack Dev Server. Starting from Angular 6, Angular CLI will offer an option to use Bazel and Closure Compiler for bundling.

The `ng serve` command builds the bundles in memory without generating actual files. While working on the project, you run `ng serve` once and then keep working on your code. Every time you modify and save a file, Angular CLI will rebuild the bundles in memory (it takes a couple of seconds) and you'll see the results of your code modifications right away. The following JavaScript bundles were generated:

- inline.bundle.js is a file used by the Webpack loader to load other files
- main.bundle.js includes your own code (components, services, etc)
- polyfills.bundle.js includes polyfills needed by Angular so it can run in older browsers
- styles.bundle.js includes CSS styles from your app
- vendor.bundle.js includes the code of Angular framework and its dependencies

For each bundle Angular CLI generates a source maps file to allow debugging the original TypeScript even though the browser will run the generated JavaScript. Don't be scared by the large size of the vendor.bundle.js - it's a dev build, and the size will be substantially decreased when we'll build the production bundles.

SIDEBAR

Some useful options of ng new

When you generate a new project with the `ng new` command, you can specify an option that can change what is being generated. If you don't want to generate a separate CSS file for the application component styles, specify the inline styles option:

```
ng new hello-cli -is
```

If you don't want to generate a separate HTML file for the application component template, use the inline template option:

```
ng new hello-cli -it
```

If you don't want to generate a file for unit tests, use the skip tests option:

```
ng new hello-cli -st
```

You can specify multiple options for the same `ng new` command.

To create a minimal project without the testing support and an app component with the inline template and styles use the option `--minimal`:

```
ng new hello-cli --minimal
```

If you're planning to implement navigation in your app, use the option `-routing` to generate an additional module where you'll be configuring routes:

```
ng new hello-cli --routing
```

For the complete list of available options, run the `ng help new` command.

1.4.2 Development and production builds

The `ng serve` command bundled the app in memory but didn't generate files and didn't optimize our Hello CLI application. We'll use the `ng build` command for file generation, but now let's start discussing the bundle size optimization and two modes of compilation.

Open the Network tab in the dev tools of your browser and you'll see that the browser had to load 7MB to render this simple app. In dev mode, the size of the app is not a concern because you run the server locally and it takes the browser less than a second to load this app as shown in figure 1.5.

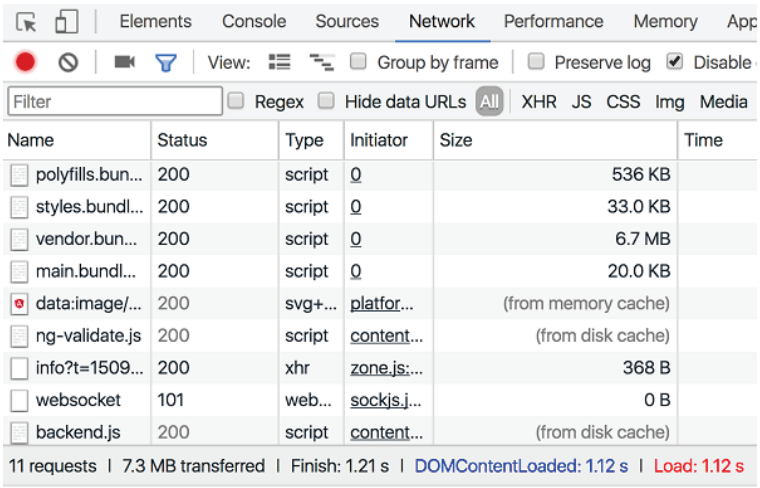

Figure 1.6 Running the non-optimized app

Now visualize a user with a mobile device browsing the Internet over a regular 3G connection. It'll take 20 seconds to load the same Hello CLI app. Many people can't tolerate waiting for 20 seconds for any app except Facebook (30% of the earth's population lives on Facebook). We need to reduce the size of the bundles before going live.

Applying the `-prod` option while building the bundles will produce much smaller bundles (see figure 1.6) by optimizing your code, i.e. it'll rename your variables into single-letter ones, will remove comments and empty lines, and will remove the majority of the unused code. There is another piece of code that can be removed from app bundles - the Angular compiler. Yes, the `ng serve` command included the compiler into the vendor.bundle.js. But how are you going to remove the Angular compiler from your deployed app when you build it for production?

1.5 JIT and AoT Compilations

Let's revisit the code of app.component.html. For the most part, it consists of standard HTML tags, but there is one line that browsers won't understand:

```
Welcome to {{title}}!
```

These double curly braces represent binding a value into a string in Angular, but this line has to be compiled by the Angular compiler (it's called ngc) to replace the binding with something that browsers would understand. A component template can include another Angular-specific syntax (e.g. structural directive `*ngIf` and `*ngFor`) that need to be compiled before asking the browser to render the template.

When you run the `ng serve` command, the template compilation is performed inside the browser. After the browser loads your app bundles, the Angular compiler (packaged inside vendor.bundle.js) performs the compilation of the templates from main.bundle.js. This is called just-in-time (JIT) compilation. This term means that the compilation happens at time of the arrival of the bundles to the browser.

The drawbacks of the JIT compilation are:

1. There is a time gap between the loading bundles and rendering the UI. This time is spent on JIT compilation. On a small app like Hello CLI, this time is minimal, but in real-world apps, the JIT compilation can take a couple of seconds, so the user needs to wait longer before just seeing your app.
2. The Angular compiler has to be included in the vendor.bundle.js, which adds to the size of your app.

Using the JIT compilation in production is discouraged, and we want the templates to be pre-compiled into JavaScript before the bundles are created. This is what ahead-of-time (AoT) compilation is about.

The advantages of the AoT compilation are:

1. The browser can render the UI as soon as your app is loaded. There is no need to wait for code compilation.
2. The ngc compiler is not included in the vendor.bundle.js and the resulting size of your app might be smaller.

Why use the word "might" and not "will"? The removing of the ngc compiler from the bundles should always result in smaller app size? Not always. The reason is that the compiled templates are larger than those that use a concise Angular syntax. The size of Hello CLI will definitely be smaller as there is only one line to compile. But in larger apps with lots of views, the compiled templates may increase the size your app so it's even larger than the JIT-compiled app with ngc included in the bundle. But you should use the AoT mode anyway because the user will see the initial landing page of your app sooner.

NOTE You may be surprised by seeing the ngc compiler errors in the app that was compiling fine with tsc. The reason being that AoT requires your code to be statically analyzable. For example, you can't use the keyword `private` with properties that are used in template and no default exports are allowed. Fix the errors reported by the ngc compiler and enjoy the benefits of the AoT compilation.

No matter if you choose JIT or AoT compilation, at some point you'll decide to do an optimized production build. How to do this?

1.5.1 Creating bundles with the -prod option

When you build the bundles with the `-prod` option, Angular CLI performs code optimization and AoT compilation. Let's see it in action by running the following command in our Hello CLI project:

```
ng serve -prod
```

Open the app in your browser and check the Network tab as shown in figure 1.6. Now

the size of the same app is only 108KB gzipped.

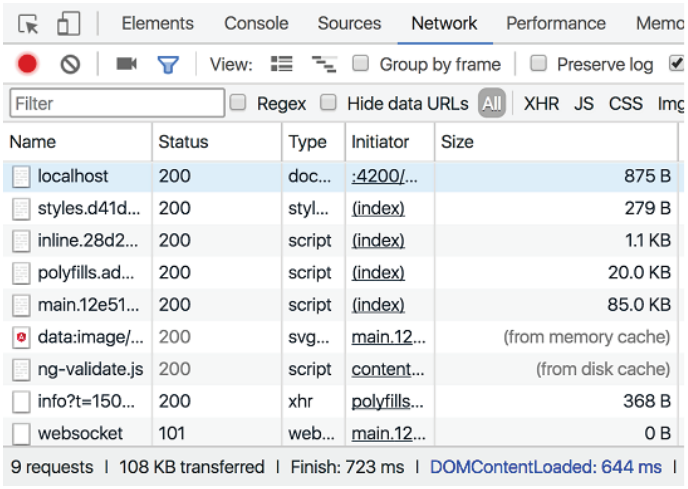

Figure 1.7 Running the optimized app with AoT

TIP When you do a production build, Angular CLI also applies additional code optimization option `--build-optimizer` by default. This option is available for both `ng serve` and `ng build` commands. The latter command actually generates files.

Take a look at the column with the bundle sizes - the dev server even did the gzip compression for us. Note that the file names of the bundles now include a hash code of each bundle. Angular CLI calculates a new hash code on each prod build to prevent browsers from using the cached version if a new app version is deployed in prod.

Shouldn't we always use AoT? Ideally, you should unless you use some third-party JavaScript libraries which produce errors during the AoT compilation. If you run into this problem, turn the AoT compilation off by building the bundles with the following command:

```
ng serve -prod -aot false
```

Figure 1.7 shows that both the size and the load time increased compared to the AoT compiled app seen in figure 1.6.

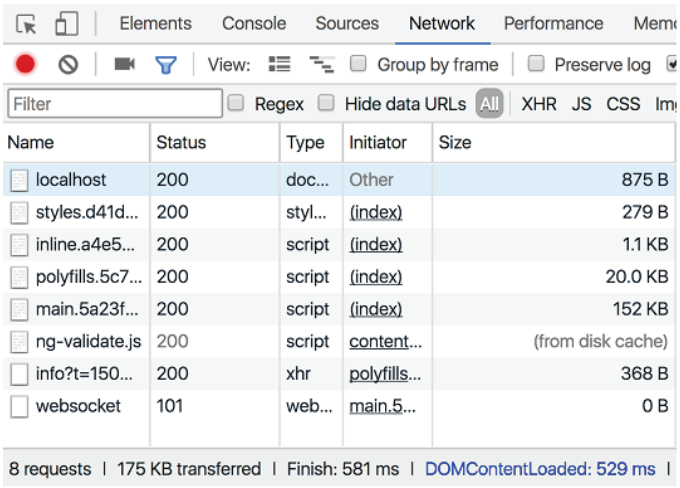

Figure 1.8 Running the optimized app without AoT

1.5.2 Generating bundles on the disk

We were using the `ng serve` command, which was building the bundles in memory. When you're ready to generate prod files, use the `ng build` command instead. The `ng build` command generates files in the dist directory (by default), but the bundle sizes won't be optimized.

With `ng build -prod` the generated files will be optimized but not compressed, so you'd need to apply the gzip compression to the bundles afterward. We'll go over the the process of building production bundles and deploying the app on the Node.js server in the section 12.5.3 "Building an app for prod deployment on the server" in chapter 12.

After the files are built in the dist directory, you can copy them to whatever web server you use. Read the product documentation for your web server, and if you know where to deploy an index.html file in your server, this would be the place for the Angular app bundles as well.

The goal of this section was to get you started with Angular CLI, and we'll continue its coverage in chapter 2. The first generated app is rather simple and it doesn't illustrate all the features of Angular, and the next section will give you some ideas of how things are done in Angular.

1.6 Introducing the sample ngAuction app

To make this book more practical, we'll start every chapter by showing you small applications that illustrate Angular syntax or techniques, and at the end of most of the chapters we'll use the new concepts in a working application. You'll see how components and services are combined into a working application.

Imagine an online auction (let's call it ngAuction) where people can browse and search the products. When the results are displayed, the user can select a product and bid on it. Each new bid will be validated on the server and will either be accepted or rejected. The information on the latest bids will be pushed by the server to all users subscribed for such notifications.

The functionality of browsing, searching, and placing bids will be implemented by

making requests to the server's RESTful endpoints, implemented in the server developed with Node.js. The server will use WebSockets to push notifications about the user's bid acceptance or rejection and about the bids placed by other users. Figure 1.8 depicts sample workflows for ngAuction.

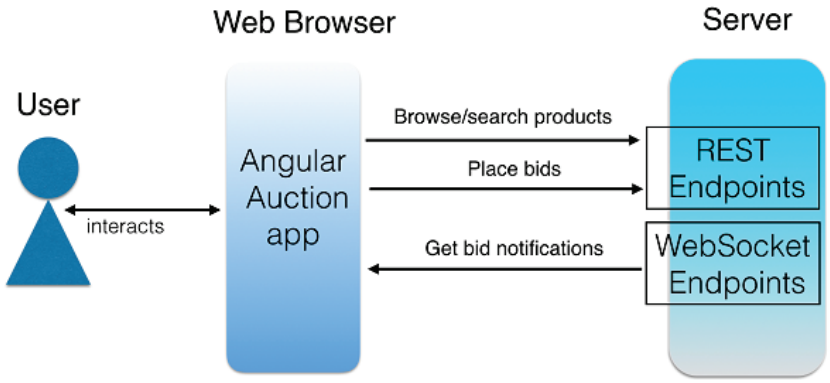

Figure 1.9 The ngAuction workflows

Figure 1.9 shows how the first version of the ngAuction home page will be rendered on desktop computers. Initially we'll use gray placeholders instead of product images.

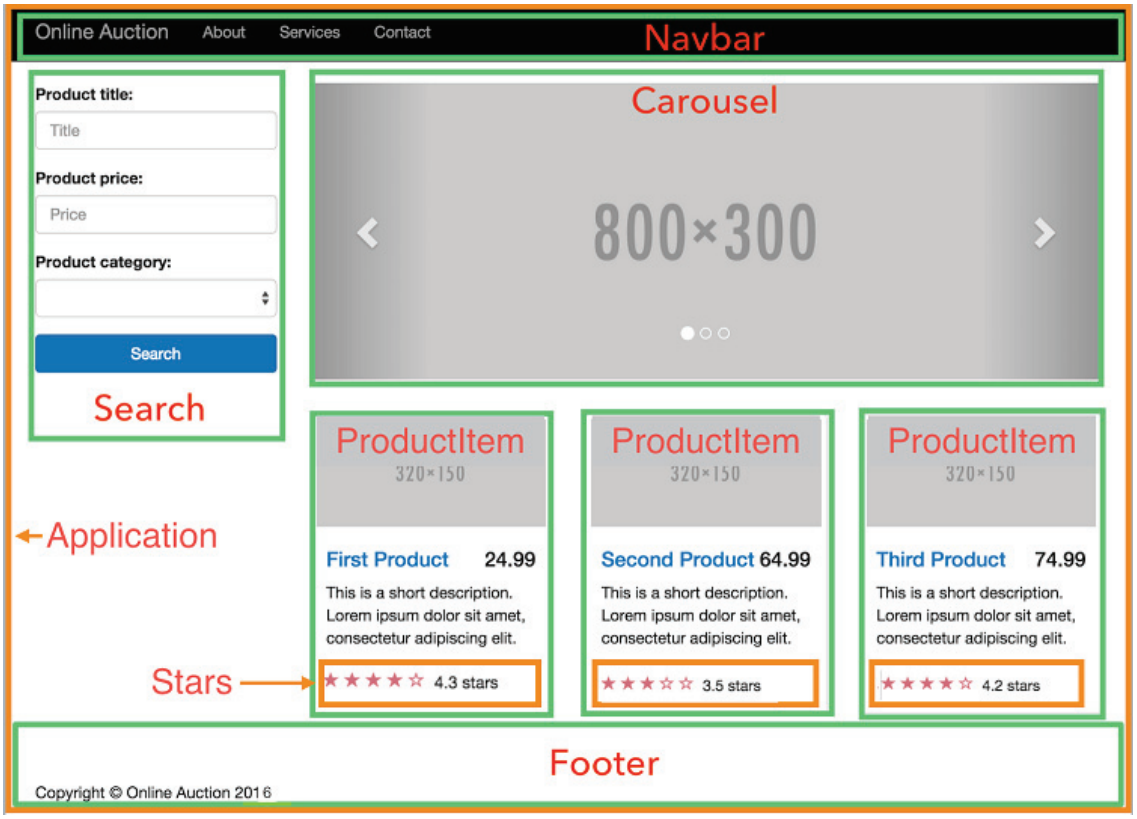

Figure 1.10 The ngAuction home page with highlighted components

We'll use responsive UI components offered by the Bootstrap library (see getbootstrap.com), so on mobile devices the home page may be rendered as in figure 1.10.

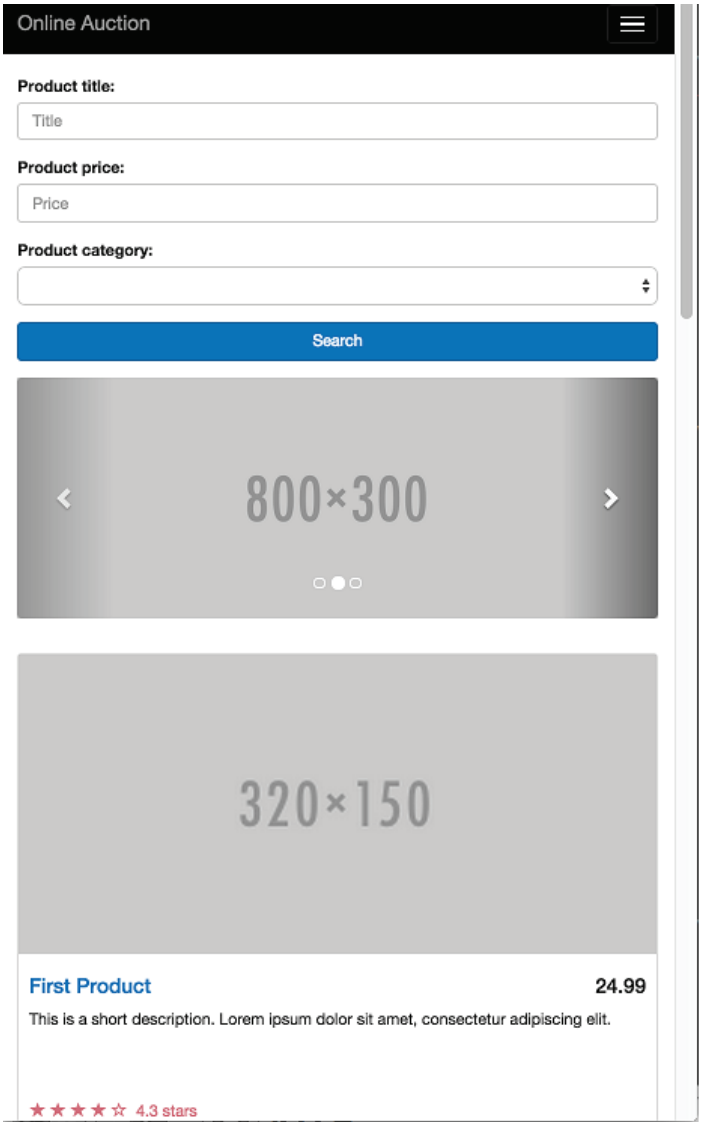

Figure 1.11 The online auction home page on smartphones

Starting from chapter 7, we'll redesign ngAuction - to completely remove the Bootstrap framework replacing it with the libraries Angluar Material and Flex Layout. The home page of the refactored version of ngAuction will look as in figure 1.11.

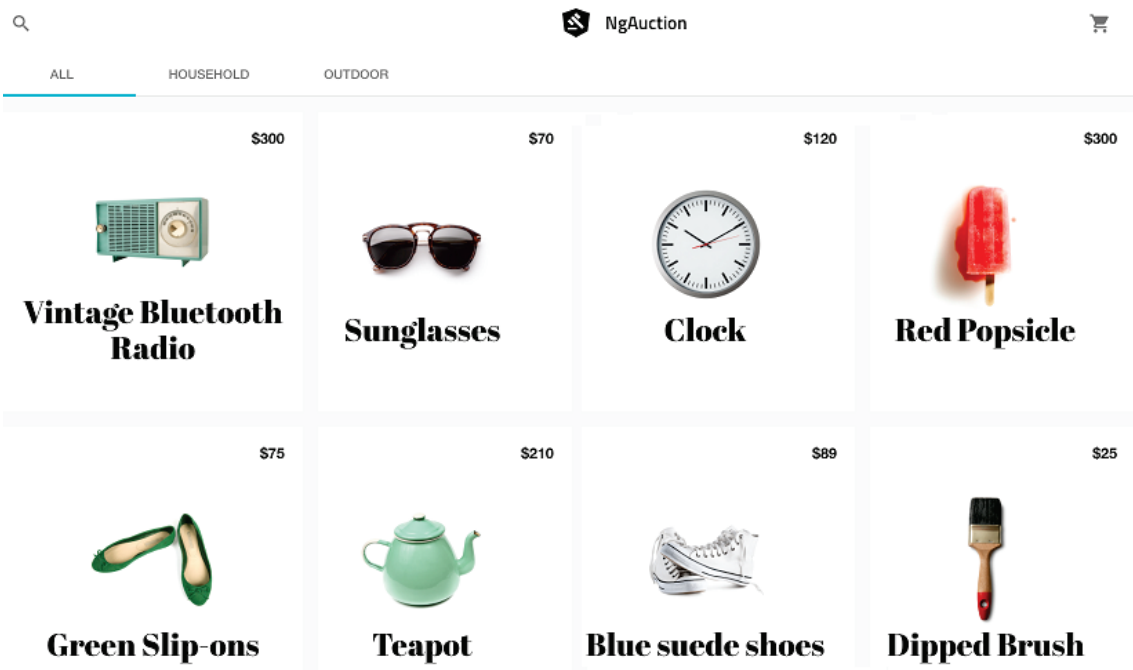

Figure 1.12 The redesigned ngAuction

The development of an Angular application comes down to creating and composing components. In chapter 2 we'll generate this project, its components, and services using Angular CLI, and in chapter 7 we'll refactor its code. Figure 1.12 shows the project structure for the ngAuction app.

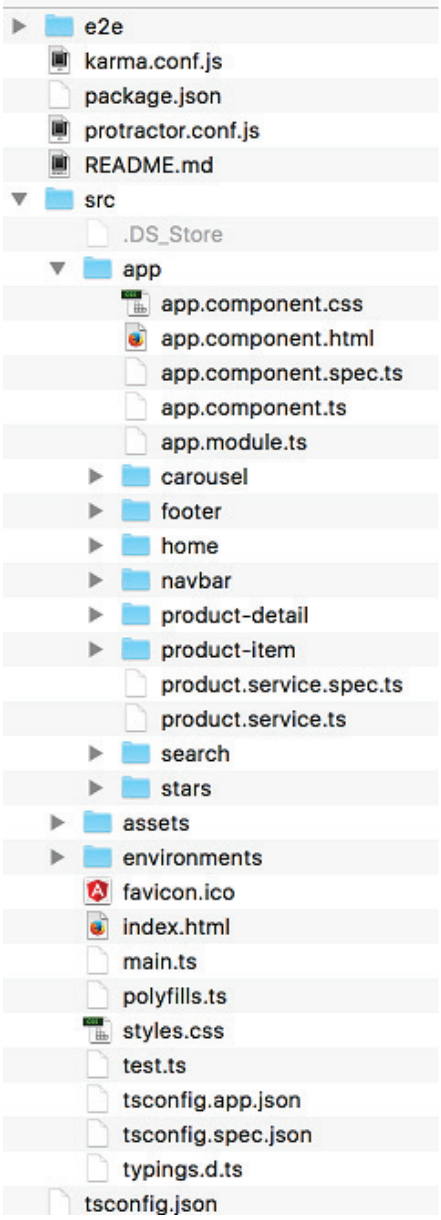

Figure 1.13 The project structure for the online auction app

In chapter 2 we'll start by creating an initial version of the landing page of ngAuction, and in subsequent chapters we'll keep adding functionality illustrating various Angular features and techniques.

NOTE **Choosing an IDE**

We recommend you to develop Angular applications using an IDE like WebStorm (it's inexpensive) or Visual Studio Code (it's free). They offer the auto-complete feature, provide convenient search, and have integrated Terminal windows so you can do all your work inside the IDE.

1.7 Summary

In this chapter, we took a high-level look at the Angular framework. We also introduced the Angular CLI tool and a sample ngAuction app that we'll be developing throughout this book.

- Angular applications can be developed in TypeScript or JavaScript
- Angular is a component-based framework
- The TypeScript source code has to be transpiled into JavaScript before deployment
- Angular CLI is a great tool that helps in jump-starting your project. It supports bundling and serving your apps in development, and preparing production builds