**Chapter 3. Router basics**

*This chapter covers*

* Configuring parent and child routes
* Passing data while navigating from one route to another
* Configuring and using child routes

In a single-page application (SPA), the web page won’t be reloaded, but its parts may change. You’ll want to add navigation to this application, so it’ll change the content area of the page (known as the *router outlet*) based on the user’s actions. The Angular router allows you to configure and implement such navigation without performing a full page reload.

In general, you can think of a *router* as an object responsible for the view state of the application. Every application has one router object, and you need to configure the routes of your app.

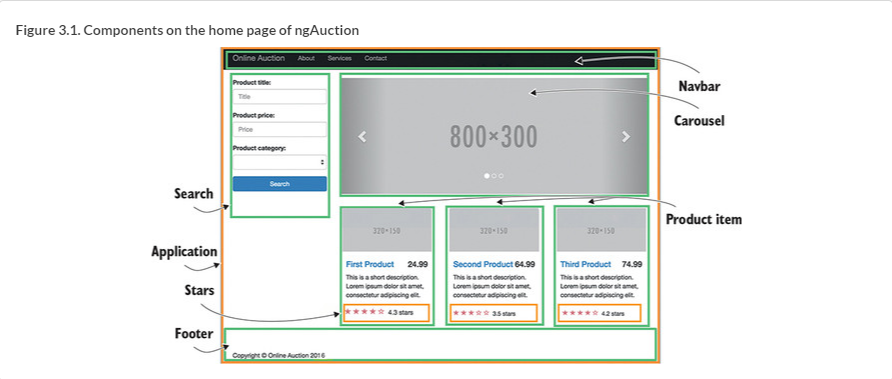
In this chapter, we’ll discuss the major features of the Angular router, including configuring routes in parent and child components, passing data to routes, and adding router support to the HTML anchor elements.

The ngAuction app now has a home view; you’ll add a second view so that if the user clicks the title of a product on the home page, the page’s content will change to display the details of the selected product. At any given time, the user will see either the HomeComponent or the ProductDetailComponent in the <router-outlet> area.

### 3.1. Routing basics

You can think of an SPA as a collection of states, such as home, product detail, and shipping. Each state represents a different view of the same SPA.

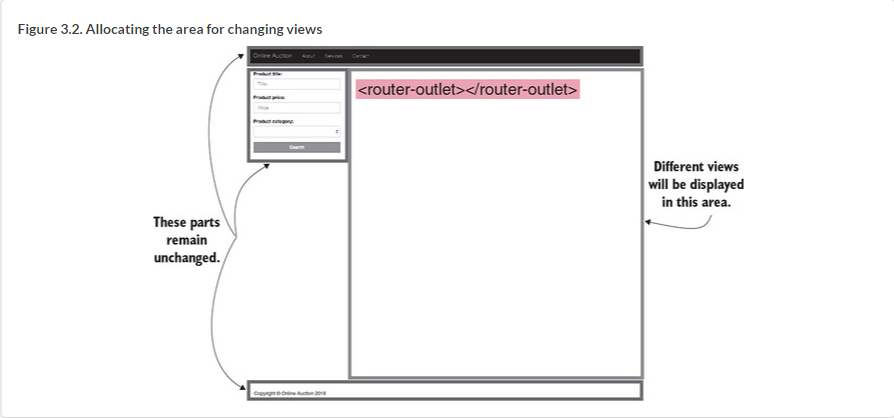
[Figure 3.1](https://livebook.manning.com/book/angular-development-with-typescript-second-edition/chapter-3/10#!/book/angular-development-with-typescript-second-edition/chapter-3/ch03fig01) shows the landing page of the ngAuction app, which has a navigation bar (a component) at the top, a search form (another component) on the left, and a footer (yet another component) at the bottom, and you want these components to remain visible all the time.



Besides the parts that are always visible, there’s a content area (see [figure 3.2](https://livebook.manning.com/book/angular-development-with-typescript-second-edition/chapter-3/12#!/book/angular-development-with-typescript-second-edition/chapter-3/ch03fig02)) that initially will display the <nga-home> component and its children but can show other views as well, based on the user’s actions. To show other views, you’ll need to configure the router, so it can display different views in the outlet, replacing one view with another. You’ll be assigning a component for each view that you want to display in this area. This content area is represented by the tag <router-outlet>.

##### TIP

There can be more than one outlet on the page. We’ll cover this in [section 4.2](https://livebook.manning.com/book/angular-development-with-typescript-second-edition/chapter-3/13#!/book/angular-development-with-typescript-second-edition/chapter-4/ch04lev1sec2) in [chapter 4](https://livebook.manning.com/book/angular-development-with-typescript-second-edition/chapter-3/13#!/book/angular-development-with-typescript-second-edition/chapter-4/ch04).



The router is responsible for managing client-side navigation, and later in this chapter we provide a high-level overview of the router. In the non-SPA world, site navigation is implemented by a series of requests to a server, which refreshes the entire page by sending the appropriate HTML documents to the browser. With SPAs, the code for rendering components is already on the client (except for the lazy-loading scenarios covered in section 4.3 of [chapter 4](https://livebook.manning.com/book/angular-development-with-typescript-second-edition/chapter-3/16#!/book/angular-development-with-typescript-second-edition/chapter-4/ch04)), and you need to replace one view with another.

As the user navigates the application, the app can still make requests to the server to retrieve or send data. Sometimes a view (the combination of the UI code and data) has everything it needs already downloaded to the browser. Other times a view will communicate with the server by issuing AJAX requests or via WebSockets. Each view will have a unique URL shown in the location bar of the browser. We’ll discuss that next.

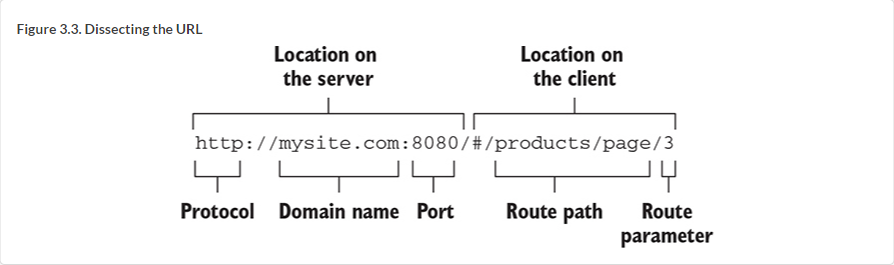
### 3.2. Location strategies

At any given time, the browser’s location bar displays the URL of the current view. A URL can contain different parts, or segments. It starts with a protocol followed by a domain name, and it may include a port number. Parameters that need to be passed to the server may follow a question mark (this is true for HTTP GET requests), like this: <http://mysite.com:8080/auction?someParam=1234>.

In a non-SPA, changing any character in the preceding URL results in a new request to the server. In SPAs, you need the ability to modify the URL without forcing the browser to make a server-side request, so the application can locate the proper view on the client. Angular offers two location strategies for implementing client-side navigation:

* **HashLocationStrategy—**A hash sign (#) is added to the URL, and the URL segment after the hash uniquely identifies the view to be used as a web page fragment. This strategy works with all browsers, including the old ones.
* **PathLocationStrategy—**This History API–based strategy works only in browsers that support HTML5. This is the default location strategy in Angular.

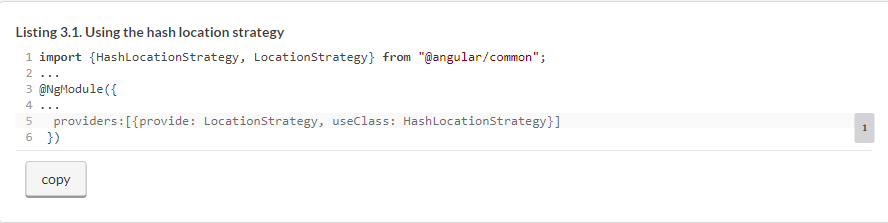
#### **3.2.1. Hash-based navigation**

A sample URL that uses hash-based navigation is shown in [figure 3.3](https://livebook.manning.com/book/angular-development-with-typescript-second-edition/chapter-3/22#!/book/angular-development-with-typescript-second-edition/chapter-3/ch03fig03). Changing any character to the right of the hash sign doesn’t cause a direct server-side request but navigates to the view represented by the path (with or without parameters) after the hash. The hash sign serves as a separator between the base URL and the client-side location of the required content.

Try to navigate an SPA like Gmail and watch the URL. For the Inbox, it looks like this: <https://mail.google.com/mail/u/0/#inbox>. Now go to the Sent folder, and the hash portion of the URL will change from *inbox* to *sent*. The client-side JavaScript code invokes the necessary functions to display the Sent view. But why does the Gmail app still show you the “Loading . . .” message when you switch to the Sent box? The JavaScript code of the Sent view can still make AJAX requests to the server to get the new data, but it doesn’t have to load any additional code, markup, or CSS from the server.

To use hash-based navigation, @NgModule() has to include the providersvalue (we discuss providers in the [section 5.2](https://livebook.manning.com/book/angular-development-with-typescript-second-edition/chapter-3/26#!/book/angular-development-with-typescript-second-edition/chapter-5/ch05lev1sec2) in [chapter 5](https://livebook.manning.com/book/angular-development-with-typescript-second-edition/chapter-3/26#!/book/angular-development-with-typescript-second-edition/chapter-5/ch05)), as shown in the following listing.

##### Listing 3.1. Using the hash location strategy



#### **3.2.2. History API-based navigation**

The browser’s History API allows you to move back and forth through the user’s navigation history as well as programmatically manipulate the history stack (see “Manipulating the Browser History” in the Mozilla Developer Network, <http://mng.bz/i64G>). In particular, the pushState()method is used to attach a segment to the base URL as the user navigates your SPA.

Consider the following URL: <http://mysite.com:8080/products/page/3>(note the absence of the hash sign). The URL segment *products/page/3* can be pushed (attached) to the base URL programmatically without using the hash tag. If the user navigates from page 3 to 4, the application’s code will push the URL segment *products/page/4*, saving the previously visited *products/page/3* in the browser history.

Angular spares you from invoking pushState() explicitly—you just need to configure the URL segments and map them to the corresponding components. With the History API–based location strategy, you need to tell Angular what to use as a base URL in your application so it can properly append the client-side URL segments. If you want to serve an Angular app on a non-root path, you have to do the following:

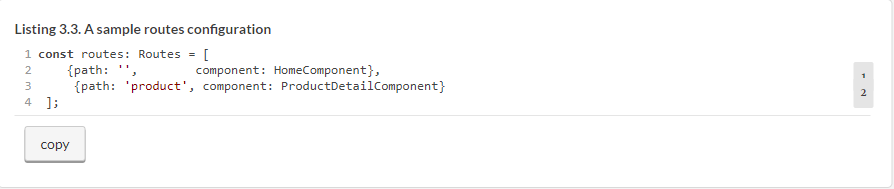
* Add the <base> tag to the header of index.html, such as <base href="/mypath">, or use the --base-href option while running ng build. Angular CLI–generated projects include <base href="/">in index.html.
* Assign a value for the APP\_BASE\_HREF constant in the root module and use it as the providers value. The following listing uses /as a base URL, but it can be any URL segment that denotes the end of the base URL.



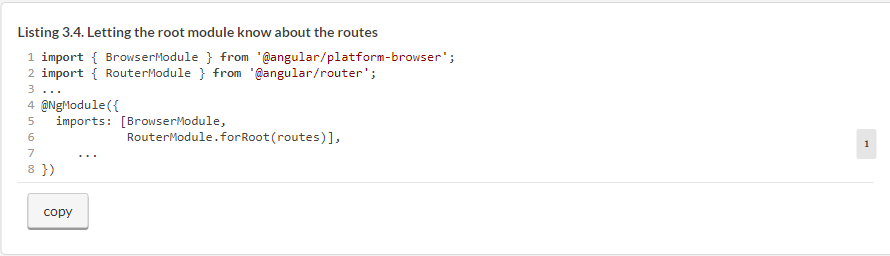
APP\_BASE\_HREF affects how the router resolves routerLink properties and the router.navigate() calls within the app, whereas the <base href=". . ."> tag affects how the browser resolves URLs when loading static resources like <link>, <script>, and <img> tags.

### 3.3. The building blocks of client-side navigation

Let’s get familiar with the main concepts of implementing client-side navigation using the Angular router. Routes are configured using the RouterModule. If your application needs routing, make sure your package.json file includes the dependency @angular/router. Angular includes many classes supporting navigation—for example, Router, Route, Routes, ActivatedRoute, and others. You configure routes in an array of objects of type Route, as in the following listing. Each of the elements in this array is an object of type Route.



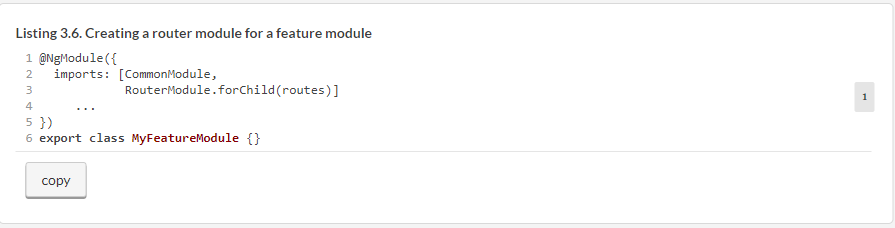
Because route configuration is done on the module level, you need to let the app module know about the routes in the @NgModule() decorator. If you declare routes for the root module, use the forRoot() method, for example, as shown in the following listing.



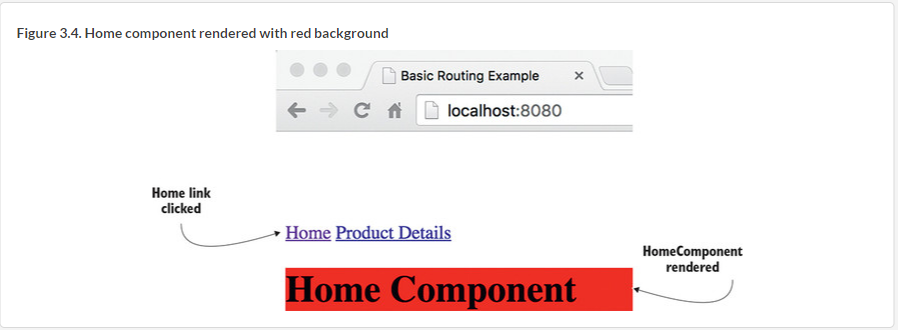
If you generated your app using the Angular CLI command **ng new** with the **--routing** option (as you did in the hands-on section in [chapter 2](https://livebook.manning.com/book/angular-development-with-typescript-second-edition/chapter-3/42#!/book/angular-development-with-typescript-second-edition/chapter-2/ch02)), you’ll get a separate file, app-routing.module.ts, where you can configure routes, as illustrated in the next listing.



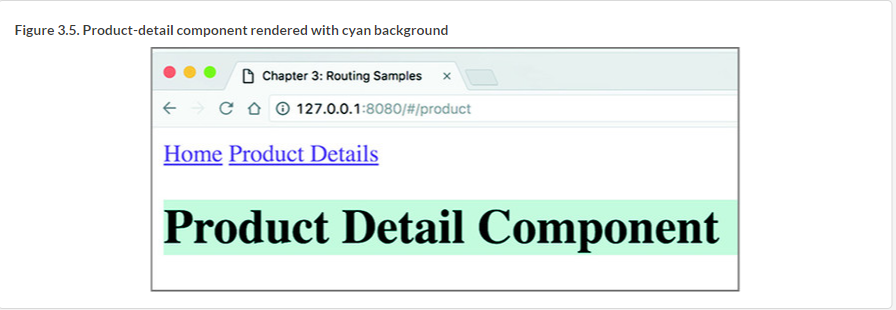
If you’re configuring routes for a feature module (not for the root one), use the **forChild()** method, which also creates a router module but doesn’t create the router service (**forRoot**() should have created the service by now), as you can see in the following listing.



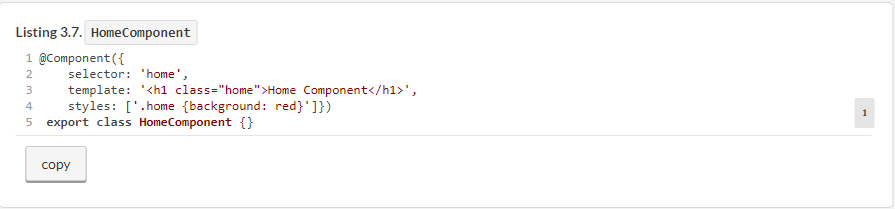
Let’s start with a simple app that illustrates routing. Say you want to create a root component that has two links, Home and Product Details, at the top of the page. The application should render either **HomeComponent** or **ProductDetailComponent**, depending on which link the user clicks. **HomeComponent** will render the text “Home Component,” and **ProductDetailComponent** will render “Product Detail Component.” Initially the web page should display **HomeComponent**, as shown in [figure 3.4](https://livebook.manning.com/book/angular-development-with-typescript-second-edition/chapter-3/46#!/book/angular-development-with-typescript-second-edition/chapter-3/ch03fig04).



After the user clicks the Product Details link, the router should display the **ProductDetailComponent**, as shown in [figure 3.5](https://livebook.manning.com/book/angular-development-with-typescript-second-edition/chapter-3/48#!/book/angular-development-with-typescript-second-edition/chapter-3/ch03fig05).



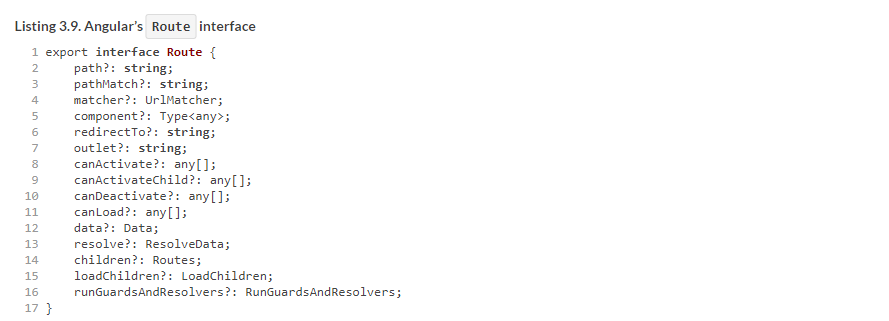
You can see how the URLs for these routes look in [figures 3.4](https://livebook.manning.com/book/angular-development-with-typescript-second-edition/chapter-3/50#!/book/angular-development-with-typescript-second-edition/chapter-3/ch03fig04) and [3.5](https://livebook.manning.com/book/angular-development-with-typescript-second-edition/chapter-3/50#!/book/angular-development-with-typescript-second-edition/chapter-3/ch03fig05). The main goal of this basic app is to become familiar with the router, so the components will be very simple, as in the following listing.



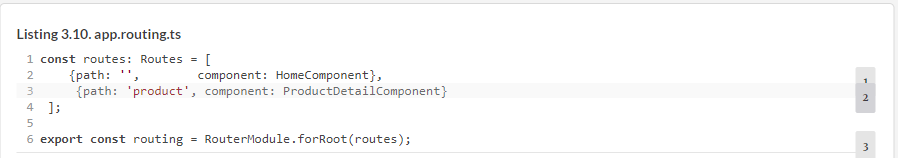
The code of the **ProductDetailComponent** looks similar, as you can see in the following listing, but instead of red it uses a cyan background.



The **Routes** type is just a collection of the objects of the type defined in the Route interface, as shown in the next listing.



You can pass to the **forRoot**() or **forChild**() functions a config object that only has a couple of properties filled in. In the basic app, you use just two properties defined in the **Route** interface: **path** and **component**. We’ll do it in a file called app.routing.ts, as in the following listing.



The next step is to create a root component that will contain the links for navigating between the home and product-detail views. The following listing shows the root AppComponent located in the app.component.ts file.



The square brackets around routerLink denote property binding, while the brackets on the right represent an array with one element (for example, ['/']). The second anchor tag has the routerLink property bound to the component configured for the /product path. The matched components will be rendered in the area marked with <router-outlet>, which in this app is located below the anchor tags.

None of the components are aware of the router configuration, because it’s the module’s business, as shown in the following listing.



The module’s providers property is an array of registered providers (there’s just one in this example) for dependency injection, which is covered in [chapter 5](https://livebook.manning.com/book/angular-development-with-typescript-second-edition/chapter-3/63#!/book/angular-development-with-typescript-second-edition/chapter-5/ch05). At this point, you just need to know that although the default location strategy is PathLocationStrategy, you want Angular to use the HashLocationStrategy class for routing (note the hash sign in the URL in [figure 3.5](https://livebook.manning.com/book/angular-development-with-typescript-second-edition/chapter-3/63#!/book/angular-development-with-typescript-second-edition/chapter-3/ch03fig05)).

In the project router-samples that comes with this chapter, we’ve configured multiple applications in the .angular-cli.json file. The app described in this section has the name *basic*, and you can run it by entering the following command in your Terminal window:



##### NOTE

In Angular 6, the .angular-cli.json file is renamed angular.json. Also, if you decide to run the Angular 6 version of this app (it comes with the book code samples), the --app option isn’t needed: ng serve basic -o.

##### TIP

Don’t forget to run npm install in the project router-samples.

In the basic routing code sample, we arranged the navigation using routerLink in HTML anchor tags. But what if you need to arrange navigation programmatically without asking the user to click a link?

### 3.4. Navigating to routes with navigate()

Let’s modify the basic code sample to navigate by using the **navigate**() method. You’ll add a button that will also navigate to the **ProductDetailComponent**, but this time no HTML anchors will be used.

The following listing reuses the router configuration from the previous section but invokes the navigate()method on the Router instance that will be injected into the AppComponent via its constructor.



In [listing 3.13](https://livebook.manning.com/book/angular-development-with-typescript-second-edition/chapter-3/75#!/book/angular-development-with-typescript-second-edition/chapter-3/ch03ex13), the user needs to click a button to go to the product route. But the navigation could be implemented without requiring user actions—just invoke the navigate() method from your application code when necessary. For example, you can force the app to navigate to the login route if the user isn’t logged in.

By default, the address bar of the browser changes as the user navigates with the router. If you don’t want to show the URL of the current route, use the **skipLocationChange** directive:



In this case, the URL remains http://localhost:4200/#/ even when the user navigates to the product route. To achieve the same effect with programmatic navigation, use the following syntax:



##### ***HANDLING 404 ERRORS***

If the user enters a nonexistent URL in your application, the router won’t be able to find a matching route and will print an error message on the browser console, leaving the user to wonder why no navigation is happening. Consider creating an application component that will be displayed whenever the application can’t find the matching component.

For example, you could create a component named \_404Component and configure it with the wildcard path \*\*:



The wildcard route configuration has to be the last element in the array of routes. The router always treats the wildcard route as a match, so any routes listed after the wildcard route won’t be considered.

### 3.5. Passing data to routes

The basic routing application showed how you can display different components in the router outlet area, but you often need to also pass some data to the component. For example, if the app component shows a list of products and you want to navigate to the product-detail route, you need to pass the product ID to the component that represents the destination route. In this case, you need to add a parameter to the path property in the route configuration. In the following listing, you change the configuration of the product route to indicate that the **ProductDetailComponent** has to be rendered when the URL segment includes the value after 'product' (the colon denotes the variable part of the path - :id).



Accordingly, your app component needs to include the value of the product ID in the **routerLink** to ensure that the value of the product ID will be passed to the **ProductDetailComponent** if the user chooses to go this route. The new version of the app may look like the following listing.



The second **routerLink** is bound to the two-element array providing the static part of the path /product and the value /:id that represents the product ID. The elements of the array build up the path specified in the routes configuration given to the RouterModule.forRoot() method. For the product-detail route, Angular will construct the URL segment /product/1234.

To see this app in action, run the following command in your Terminal window:

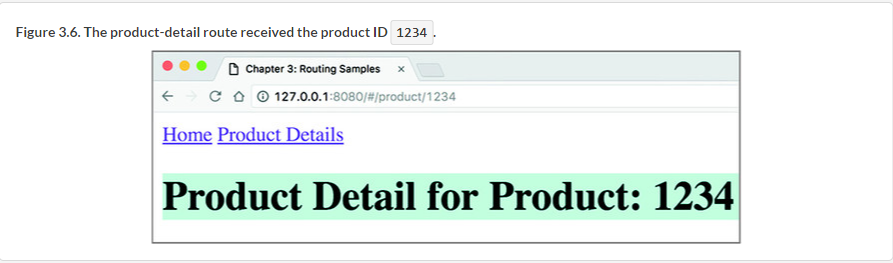


#### **3.5.1. Extracting parameters from ActivatedRoute**

If a parent component can pass a parameter to the route, the component that represents the destination route should be able to receive it. Instruct Angular to inject the instance of **ActivatedRoute** to the constructor of the component that represents the destination route. The instance of the **ActivatedRoute** will include the passed parameters, as well as the route’s URL segment and other properties. The new version of the component, which renders product detail and is capable of receiving parameters, will be called ProductDetailComponent, which will get an object of type **ActivatedRoute** injected into it, as shown in the following listing.



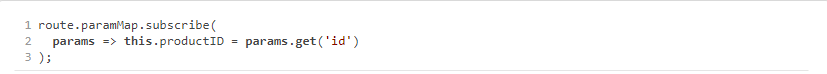
[Figure 3.6](https://livebook.manning.com/book/angular-development-with-typescript-second-edition/chapter-3/95#!/book/angular-development-with-typescript-second-edition/chapter-3/ch03fig06) shows how the product-detail view will be rendered in the browser. Note the URL: the router replaced the product/:id path with /product/1234.



##### **PASSING CHANGING PARAMETERS TO THE ROUTE**

In this app, you use the property snapshot of type ActivatedRouteSnapshot to retrieve the parameter’s value. The snapshot means “one-time deal,” and this property is used in scenarios when the parameters passed to the route never change. In your app, it works because the product ID in the parent route never changes and is always 1234. But if you try to change the URL shown in [figure 3.6](https://livebook.manning.com/book/angular-development-with-typescript-second-edition/chapter-3/98#!/book/angular-development-with-typescript-second-edition/chapter-3/ch03fig06) manually (for example, make it /product/12345), the ProductDetailComponent won’t reflect the change in the parameter.

There are scenarios when the parameter value keeps changing after navigating to a route. For example, the AppComponent renders a list of products, and the user can select different products. Both AppComponentand ProductDetailComponent are rendered in the same window. In this case, instead of using the snapshotproperty, you need to subscribe to the **ActivatedRoute.paramMap** property, which will emit a new value each time the user clicks on a different product, for example:



You’ll see this example in [chapter 6](https://livebook.manning.com/book/angular-development-with-typescript-second-edition/chapter-6/ch06) in [section 6.6](https://livebook.manning.com/book/angular-development-with-typescript-second-edition/chapter-6/ch06lev1sec6).

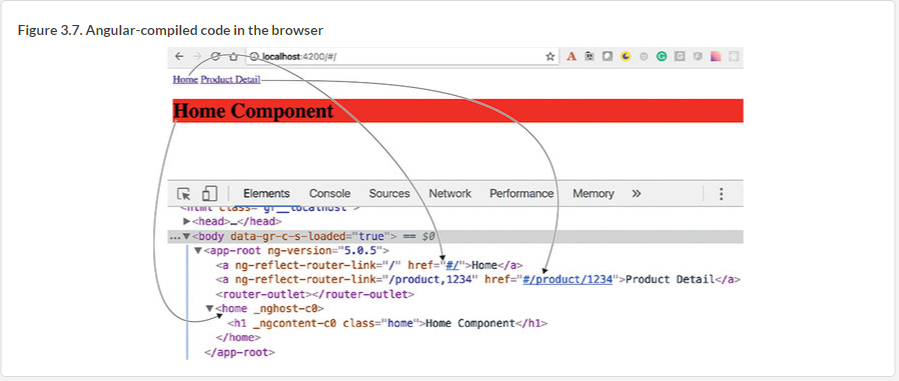
Let’s review the steps that Angular performed under the hood to render the main page of the application:

**1**.  Check the content of each routerLink.

**2**.  Concatenate the values specified in the array. If an array item is an expression, evaluate this expression (like productId). Finally, append the value of APP\_BASE\_HREF in the beginning of the resulting string.

**3**.  The RouterLink directive adds the href attribute if this directive is attached to an <a> element; otherwise, it just listens to the click events.

[Figure 3.7](https://livebook.manning.com/book/angular-development-with-typescript-second-edition/chapter-3/102#!/book/angular-development-with-typescript-second-edition/chapter-3/ch03fig07) shows a snapshot of the home page of the application with the Chrome Developer Tools panel open. Because the path property of the configured home route had an empty string, Angular didn’t add anything to the base URL of the page. But the anchor under the Product Details link has already been converted into a regular HTML tag. When the user clicks the Product Details link, the router will attach a hash sign and add /product/1234 to the base URL so that the absolute URL of the product-detail view will become http://localhost:4200/#/product/1234.



##### **NOTE**

In this section, you learned how to pass dynamic data to routes—the data that may change during the runtime. Sometimes, you need to pass static data to routes (data that doesn’t change). You can pass any arbitrary data to routes by using the **data** property in the **routes** configuration. You’ll see such an example in section 4.3.1 in [chapter 4](https://livebook.manning.com/book/angular-development-with-typescript-second-edition/chapter-3/109#!/book/angular-development-with-typescript-second-edition/chapter-4/ch04).

Sometimes, you need to pass to a route optional query parameters that are not part of the route configuration. Let’s take a look at how to pass query parameters.

#### **3.5.2. Passing query parameters to a route**

You can use query parameters (the URL segment after the question mark), as in the following URL: http://localhost:4200/products?category=sports. Query parameters aren’t scoped to a particular route, and if you want to pass them while navigating with the routerLink, you can do as follows:



Because query parameters aren’t scoped to a particular route and can be accessed by any active route, the route configuration doesn’t need to include them:

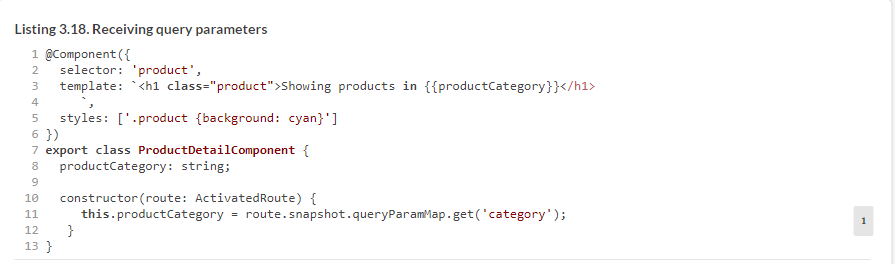


To pass query parameters using programmatic navigation, you need to have access to the Router object. The code could look like the following listing.



In this example, you pass an object with one query parameter; but you can specify multiple parameters as well.

To receive query parameters in the destination component, you’ll use the ActivatedRoute object again.

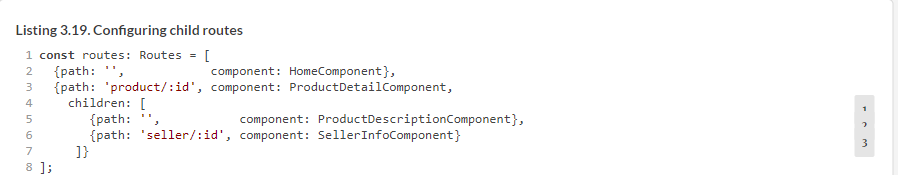


To see this code sample in action, run the following command:



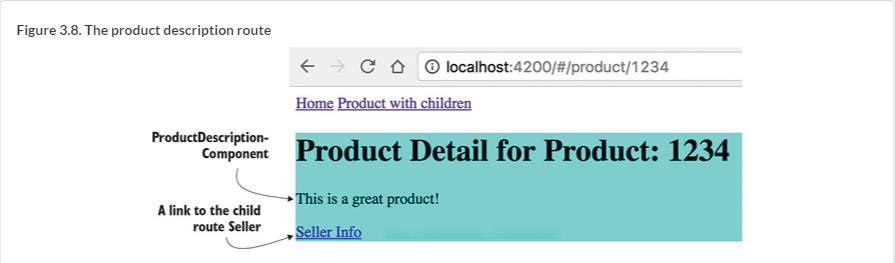
### 3.6. Child routes

An Angular application is a tree of components that have parent-child relations. A child component can have its own routes, but all routes are configured outside of any component. Imagine that you want to enable ProductDetailComponent (the child of the AppComponent) to show either the product description or the seller’s info. Moreover, there could be more than one seller of the same product, so you’ll need to pass the seller ID to show the details of the seller. The following listing configures routes for the child, ProductDetailComponent, by using the **children** property of the **Route**.



Here, the children property is a part of the configuration of the route with the path product/:id. You pass the product ID while navigating to the product route, and then, if the user decides to navigate to the seller, you pass the seller ID to the SellerInfoComponent.

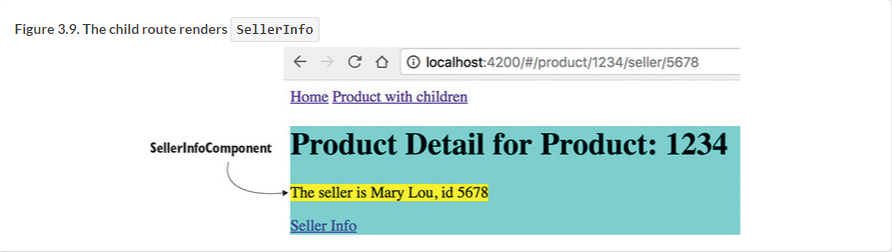
[Figure 3.8](https://livebook.manning.com/book/angular-development-with-typescript-second-edition/chapter-3/127#!/book/angular-development-with-typescript-second-edition/chapter-3/ch03fig08) shows how the application will look once the user clicks the Product Details link on the root component, which renders ProductDetailComponent (the child), showing ProductDescriptionComponent by default, because the latter component was configured for the empty path property.



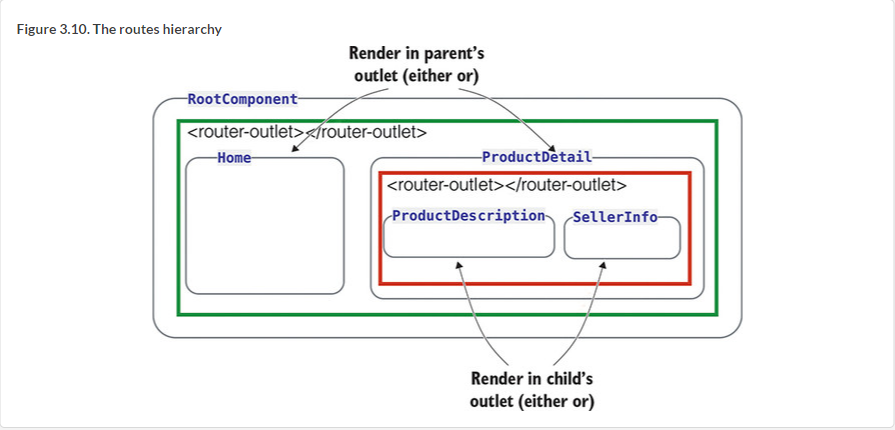
[Figure 3.9](https://livebook.manning.com/book/angular-development-with-typescript-second-edition/chapter-3/ch03fig09) oshsw ory taaoplipcin farte rkb ayvt ikclcs drk Ltrcuod Ksiatle nejf znb qrvn likcsc Sellre Jxnl.

##### **NOTE**

If you’re reading the electronic version of this book, you’ll see that the seller’s info is shown on a yellow background. We did this intentionally to discuss the styling of components a bit later in this chapter.



To implement the views shown in [figures 3.8](https://livebook.manning.com/book/angular-development-with-typescript-second-edition/chapter-3/132#!/book/angular-development-with-typescript-second-edition/chapter-3/ch03fig08) and [3.9](https://livebook.manning.com/book/angular-development-with-typescript-second-edition/chapter-3/132#!/book/angular-development-with-typescript-second-edition/chapter-3/ch03fig09), you’ll modify ProductDetailComponent so it also has two children, SellerInfoComponent and ProductDescriptionComponent, and its own <router-outlet>. [Figure 3.10](https://livebook.manning.com/book/angular-development-with-typescript-second-edition/chapter-3/132#!/book/angular-development-with-typescript-second-edition/chapter-3/ch03fig10) shows the hierarchy of components that you’re going to implement.

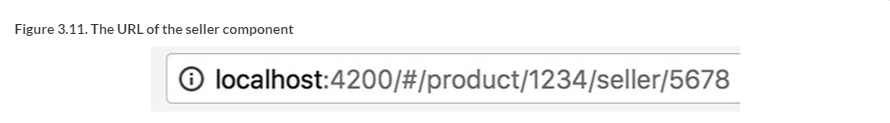


The following three listings show the code of the ProductDetailComponent, ProductDescriptionComponent, and SellerInfoComponent. The new version of ProductDetailComponent has its own outlet, where it can display either ProductDescriptionComponent (the default) or SellerInfoComponent.



When the user clicks the Product with Children link, and it has children, the product/1234 segment is added to the URL. The router finds a match to this path in the configuration object and renders the ProductDetailComponentin the outlet.

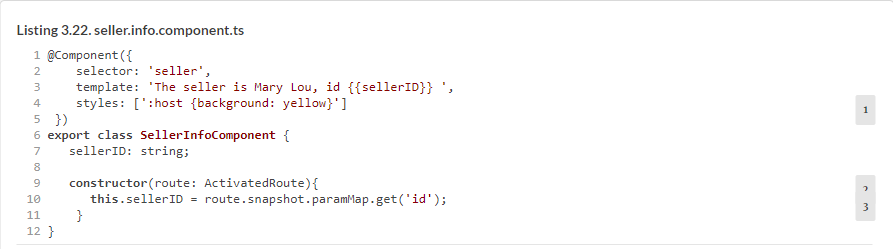
The user navigates to the ProductDetailComponent, which by default renders the ProductDescriptionComponent as per route configuration. Then, the user clicks the Seller Info link, and the URL will include the product/1234/seller/5678 segment after the hash sign, as shown in [figure 3.11](https://livebook.manning.com/book/angular-development-with-typescript-second-edition/chapter-3/138#!/book/angular-development-with-typescript-second-edition/chapter-3/ch03fig11).



The router will find a match in the configuration object and will render SellerInfoComponent in the child’s <router-outlet>. The code of the ProductDescriptionComponent is trivial, as you can see in the following listing.



Because SellerInfoComponent expects to receive the seller ID, its constructor needs an argument of type ActivatedRoute to get the seller ID, as the following listing shows, and as you did in ProductDetailComponent.



The :host pseudo class selector can be used with elements that are created using Shadow DOM (discussed in [section 8.5.1](https://livebook.manning.com/book/angular-development-with-typescript-second-edition/chapter-3/145#!/book/angular-development-with-typescript-second-edition/chapter-8/ch08lev2sec5) in [chapter 8](https://livebook.manning.com/book/angular-development-with-typescript-second-edition/chapter-3/145#!/book/angular-development-with-typescript-second-edition/chapter-8/ch08)), which provides better encapsulation for components. Although not all web browsers support Shadow DOM yet, Angular emulates Shadow DOM by default. Here, you use :host to apply the yellow background color to SellerInfoComponent. In the emulated mode, the :host selector is transformed into a randomly generated, attribute-based selector, like this:



The attribute (here, ng-host-f23ed) is attached to the element that represents the component. Shadow DOM styles of the components aren’t merged with the styles of the global DOM, and the IDs of the components’ HTML tags won’t overlap with the IDs of the DOM.

To run this code sample, enter the following command in the Terminal window of the router-samples project:



##### **DEEP LINKING**

*Deep linking* is the ability to create a link to specific content inside a web page rather than to the entire page. In the basic routing applications, you’ve seen examples of deep linking:

* The URL http://localhost:4200/#/product/1234 links not just to the product-detail page but to a specific view representing the product with an ID of 1234.
* The URL http://localhost:4200/#/product/1234/seller/5678 links even deeper. It shows the information about the seller with an ID of 5678 that sells the product whose ID is 1234.

You can easily see deep linking in action by copying the link http://localhost:4200/#/product/1234/seller/5678 from the application running in Chrome and pasting it into Firefox or Safari. There is a caveat, though. With PathLocationStrategy, when you enter the direct URL of the route in the browser’s address bar, it still makes a request to the server, which won’t find the resource (and rightly so) named as your route. This will cause a 404 error. Configure your web server to do a redirect to index.html of your app in cases when a requested resource isn’t found. This will put your Angular app back in control, and the route will be properly resolved. The Angular CLI development server is already configured for redirects.

##### **ROUTER EVENTS**

As the user navigates your app, Angular dispatches events, such as NavigationStart, NavigationEnd, and so on. There are about a dozen router events, and you can intercept any of them if need be. In [chapter 6](https://livebook.manning.com/book/angular-development-with-typescript-second-edition/chapter-3/151#!/book/angular-development-with-typescript-second-edition/chapter-6/ch06), [section 6.6](https://livebook.manning.com/book/angular-development-with-typescript-second-edition/chapter-3/151#!/book/angular-development-with-typescript-second-edition/chapter-6/ch06lev1sec6), you’ll see an example of using router events to decide when to show and hide the progress bar if the navigation is slow. For debugging purposes, you can log router events in the browser’s console by using the enableTracing option (it works only in the root module):



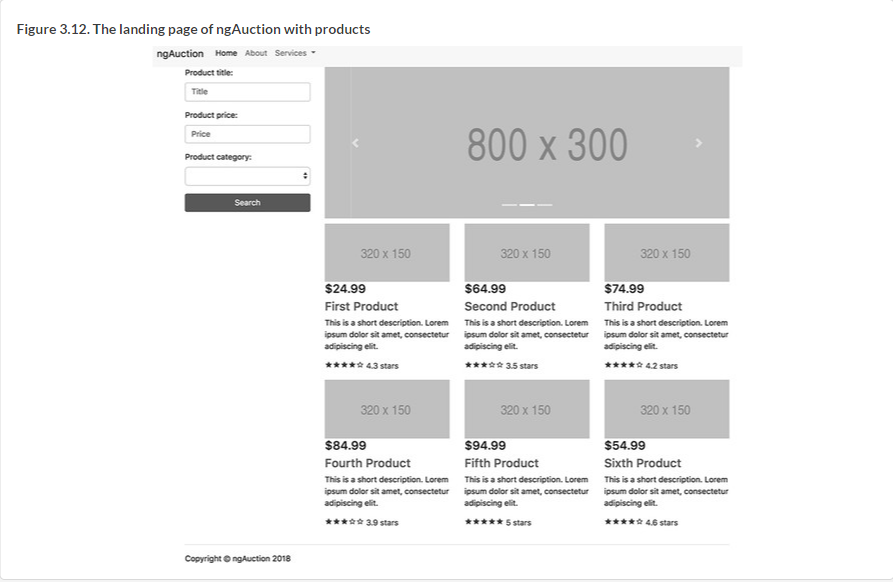
Now that you’ve learned router basic features, let’s see how you can apply them in your ngAuction application.

### 3.7. Hands-on: Adding navigation to the online auction

##### NOTE

Source code for this chapter can be found at <https://github.com/Farata/angulartypescript> and [www.manning.com/books/angular-development-with-typescript-second-edition](http://www.manning.com/books/angular-development-with-typescript-second-edition).

This hands-on exercise starts where we left off in [chapter 2](https://livebook.manning.com/book/angular-development-with-typescript-second-edition/chapter-3/158#!/book/angular-development-with-typescript-second-edition/chapter-2/ch02). So far, you’ve partially implemented the landing page of ngAuction; your goal is to render several product items under the carousel component so the landing page looks as shown in [figure 3.12](https://livebook.manning.com/book/angular-development-with-typescript-second-edition/chapter-3/158#!/book/angular-development-with-typescript-second-edition/chapter-3/ch03fig12).



The data for this view will be provided by ProductService. This hands-on exercise contains instructions for injecting the ProductService into the HomeComponent. You’ll also implement the navigation so that when the user clicks the product title, the Router will render the ProductDetail component in the <router-outlet> area.

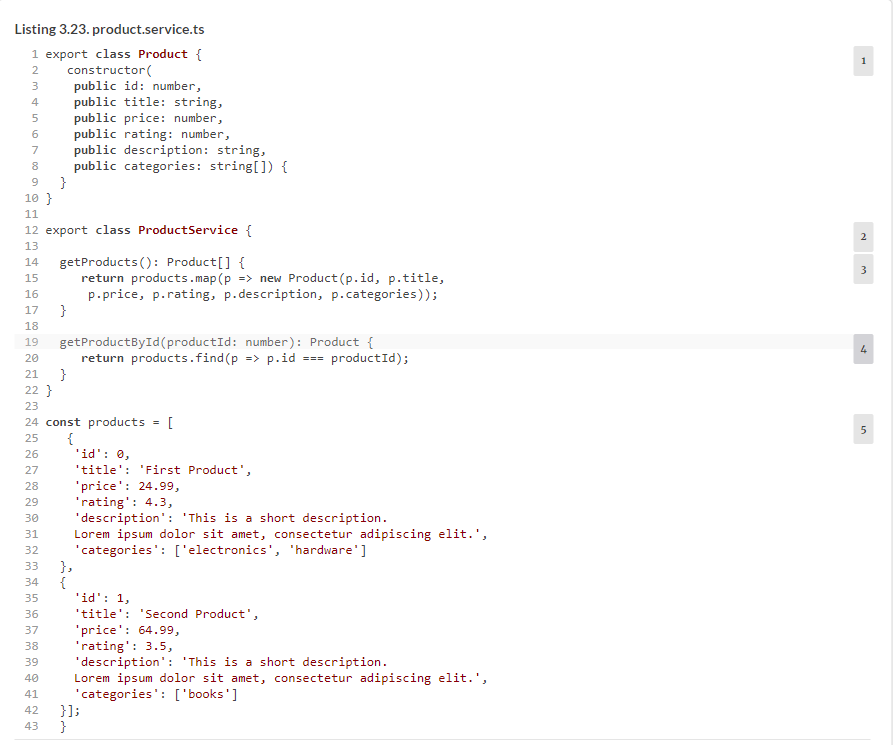
Your ProductService will contain hardcoded data about the products. Adding ProductService as an argument to the constructor of HomeComponent will instruct Angular to instantiate and inject the product object into this component.

As a starting point, you’ll use the project in the chapter3/ngAuction folder, which, for the most part, is the same as chapter2/ngAuction with one addition: the shared/product.service.ts file contains the code to provide product data.

To start working on this exercise, open the chapter3/ngAuction folder in your IDE and install the project dependencies by running the npm install command.

#### 3.7.1. ProductService

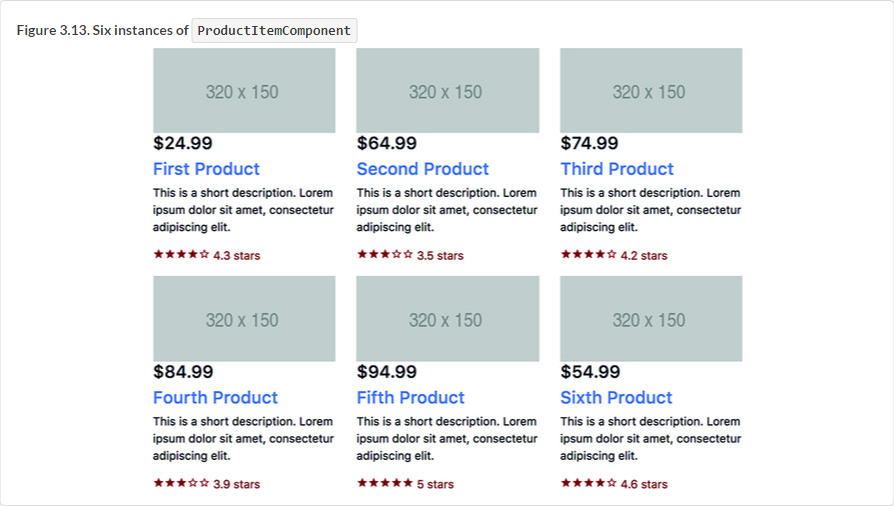
ProductService contains hardcoded data about products and the API to retrieve them. Let’s review the code in product.service.ts shown in the following listing (we removed the majority of the hardcoded data for brevity).



Shortly, you’ll be adding the code that will have Angular create an instance of the ProductService class and inject it into ProductItemComponent and ProductDetailComponent so they can invoke the getProducts() and getProductById() methods on the service.

#### 3.7.2. ProductItemComponent

[Figure 3.13](https://livebook.manning.com/book/angular-development-with-typescript-second-edition/chapter-3/ch03fig13) shows six products, each an instance of ProductItemComponent.



ProductItemComponent knows how to render one product based on the product provided by its parent (HomeComponent). Modify product-item.component.ts to look like the following listing.



The product to render will be given to ProductItemComponent via its property product decorated with @Input(). The @Input() properties are described in [section 8.2.1](https://livebook.manning.com/book/angular-development-with-typescript-second-edition/chapter-3/177#!/book/angular-development-with-typescript-second-edition/chapter-8/ch08lev2sec1). in [chapter 8](https://livebook.manning.com/book/angular-development-with-typescript-second-edition/chapter-3/177#!/book/angular-development-with-typescript-second-edition/chapter-8/ch08).

Modify product-item.component.html with the content shown in the following listing.



Note that you bind the product’s price, title, and id properties in the component’s template. You also use an Angular built-in pipe, currency, for price formatting. For now, you’ll keep the <nga-stars> component commented out because the code of the StarsComponent isn’t ready yet. Note that product.title is a routerLinkthat will navigate to ProductDetailComponent when the user clicks it. The instance of ProductItemComponent will be hosted by HomeComponent, which you’ll update next.

#### 3.7.3. HomeComponent

The home component will

* Use the injected ProductService to retrieve all featured products and store them in the productsarray.
* Render the ProductItemComponent for each product located in the products array.

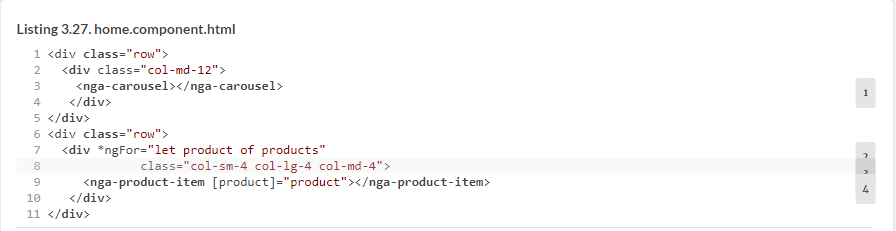
In [section 2.7.8](https://livebook.manning.com/book/angular-development-with-typescript-second-edition/chapter-3/180#!/book/angular-development-with-typescript-second-edition/chapter-2/ch02lev2sec11) in [chapter 2](https://livebook.manning.com/book/angular-development-with-typescript-second-edition/chapter-3/180#!/book/angular-development-with-typescript-second-edition/chapter-2/ch02), you implemented the first version of the HomeComponent and added the carousel to its template. Now, you need to modify the constructor to inject the ProductService and retrieve the products in the ngOnInit() method. Modify the code in home.component.ts to look like the following listing.



When Angular instantiates HomeComponent, it injects the instance of ProductService. Because you used the private qualifier, the generated JavaScript will have an instance variable, productService.

Angular invokes the component lifecycle method ngOnInit() after the constructor, and you invoke the getProducts() method there. In [section 9.2](https://livebook.manning.com/book/angular-development-with-typescript-second-edition/chapter-3/187#!/book/angular-development-with-typescript-second-edition/chapter-9/ch09lev1sec2). in [chapter 9](https://livebook.manning.com/book/angular-development-with-typescript-second-edition/chapter-3/187#!/book/angular-development-with-typescript-second-edition/chapter-9/ch09), we’ll discuss the component lifecycle methods, and you’ll see why ngOnInit() is the right place for fetching data.

Modify home.component.html to loop through the array products with the structural directive \*ngFor and render each product.



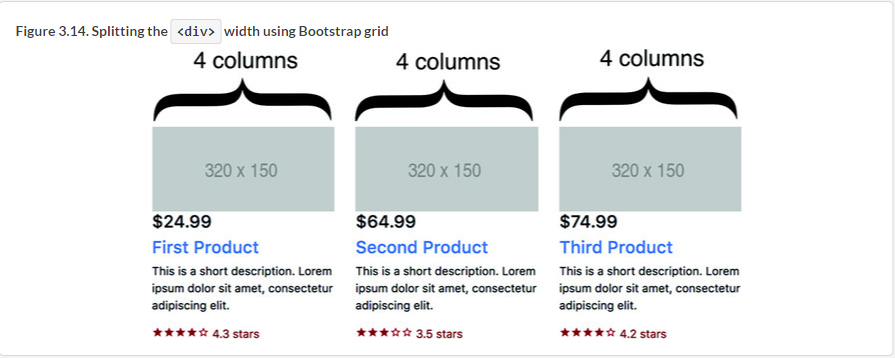
Each product is represented by the same HTML fragment on the web page. Because there are multiple products, you need to render the same HTML multiple times. The NgFor directive is used inside the component template to loop through the list of items in the data collection, rendering HTML markup for each item. In component templates, \*ngFor represents the NgFor directive.

Because the \*ngFor directive is located inside a <div>, each loop iteration will render a <div> with the content of the corresponding <nga-product-item> inside. To pass an instance of a product to ProductItemComponent, you use the square brackets for property binding:



The [product] on the left refers to the property named product inside the <nga-product-item> component, and product on the right is a local template variable declared on the fly in the \*ngFor directive as let product.

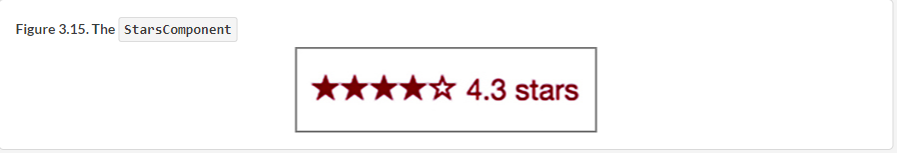
The Bootstrap’s grid styles class="col-sm-4 col-lg-4 col-md-4" instruct the browser to split the width of the <div> by evenly allocating 4 columns (out of 12) to each product on small, large, and medium devices, as shown in [figure 3.14](https://livebook.manning.com/book/angular-development-with-typescript-second-edition/chapter-3/191#!/book/angular-development-with-typescript-second-edition/chapter-3/ch03fig14). Try to remove this class, and see how it affects the UI.



Run the ng serve -o command, and you’ll see six products rendered under the carousel, as shown in [figure 3.14](https://livebook.manning.com/book/angular-development-with-typescript-second-edition/chapter-3/195#!/book/angular-development-with-typescript-second-edition/chapter-3/ch03fig14), except there won’t be any stars with product ratings. We’ll take care of the stars next.

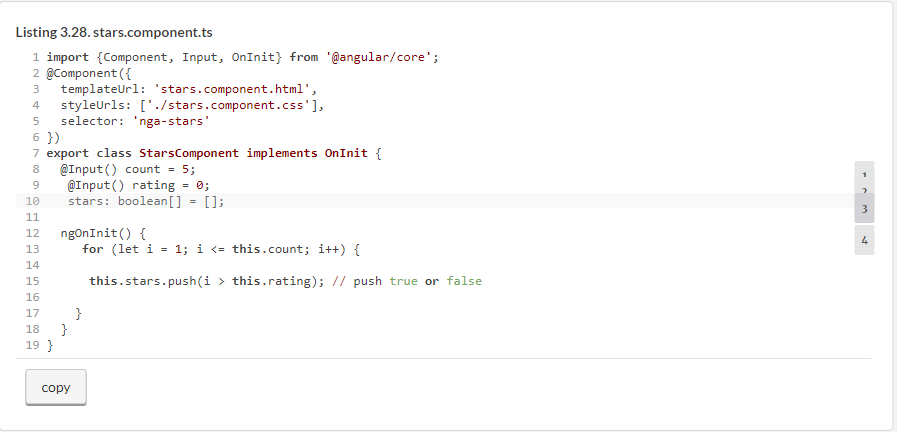
#### 3.7.4. StarsComponent

The StarsComponent will render stars to display the product rating, as shown in [figure 3.15](https://livebook.manning.com/book/angular-development-with-typescript-second-edition/chapter-3/197#!/book/angular-development-with-typescript-second-edition/chapter-3/ch03fig15).

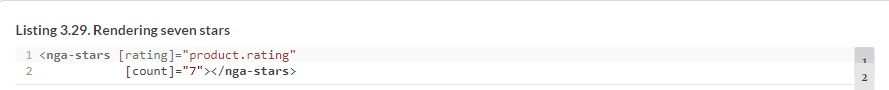


On the landing page of ngAuction, the StarsComponent will be a child component of ProductItemComponent. Eventually, we’ll reuse it in the ProductDetailComponent as well.

Modify the code of the generated stars.component.ts file to look as follows.



The count property specifies the total number of stars to be rendered. If the parent component doesn’t provide the value for this input property, five stars will be rendered by default. The StarsComponent can render more or fewer stars if need be. The following listing shows how you can render seven stars.



The elements of the stars array with the false value represent stars without a color, and those with truerepresent stars filled with color. The rating property stores the average product rating that determines how many stars should be filled with color and how many should remain empty.

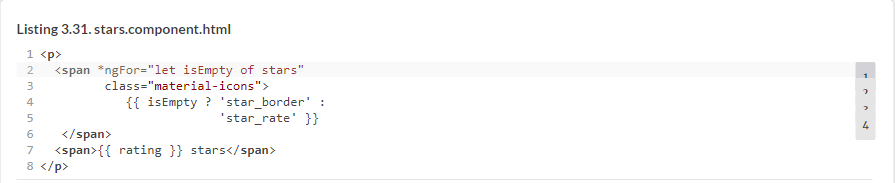
The Bootstrap 4 framework doesn’t include images that render stars. There are several popular libraries of icon fonts out there (Material Design Icons, Font Awesome, Octicons, and so on); we’ll use Material Design Icons. To keep them local in the project, install these icons as follows:



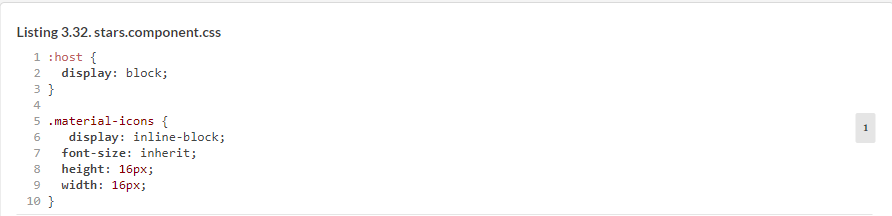
Then add these fonts to the styles section of the .angular-cli.json file so it looks like the following listing.



Modify the content of stars.component.html to look like the following listing.



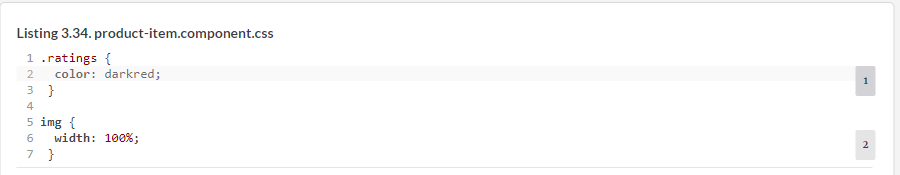
Note how you bind either one CSS class or another (double curly brackets). To style the star icon, add the following styles to stars.component.css.



The ProductItemComponent will be the parent of the StarsComponent. To make it a child of the ProductItemComponent, uncomment the <div> in the product-item.component.html file created earlier.



The CSS selector ratings will be defined in the product-item.component.css file. You’ll make the stars dark red and add some padding by using the following listing’s style in the product-item.component.css file.



You add this style to the parent of StarsComponent to be able to pick different star colors in the child component if need be. If another component will need to render stars, you can choose another color there. Now your ProductItemComponent renders ratings for each child product, as shown in [figure 3.14](https://livebook.manning.com/book/angular-development-with-typescript-second-edition/chapter-3/217#!/book/angular-development-with-typescript-second-edition/chapter-3/ch03fig14).

It’s time to implement navigation with the Router.

#### 3.7.5. ProductDetailComponent

In [chapter 2](https://livebook.manning.com/book/angular-development-with-typescript-second-edition/chapter-3/219#!/book/angular-development-with-typescript-second-edition/chapter-2/ch02), you generated the routing module, but it has only one configured route, which renders HomeComponent, as you can see in the following listing.



You want to add another route so that when the user clicks on the product title in the ProductItemComponent, the Router replaces HomeComponent with ProductDetailComponent. During this navigation, you want to pass the ID of the selected product to ProductDetailComponent. Modify the routes configuration to look like the following listing, and don’t forget to import ProductDetailComponent.

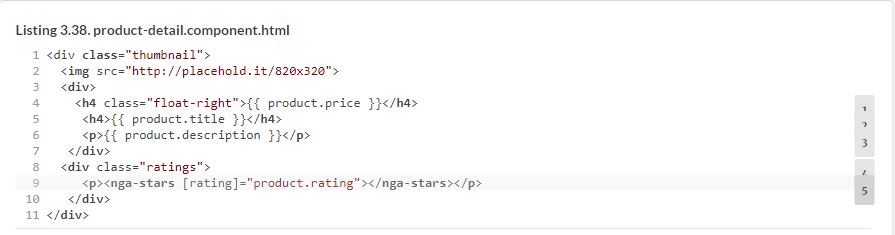


The ProductDetailComponent will receive the selected product ID from the parent via the injected ActivatedRouteand then make a request to ProductService to retrieve product details. Because ProductDetailComponent will reuse the instance of ProductService that Angular created for you on app startup, add this service to the constructor’s arguments. Modify the code in product-detail.component.ts to look like the following listing.

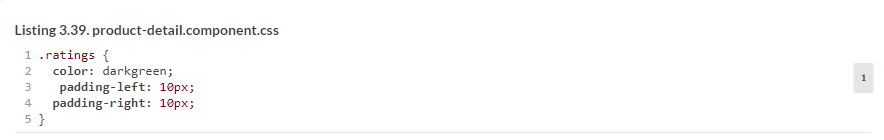


The values of the properties of the instance variable product will be bound to the component template and rendered by the browser. The template of ProductDetailComponent will contain the product image (a gray rectangle) with product price, title, and description.

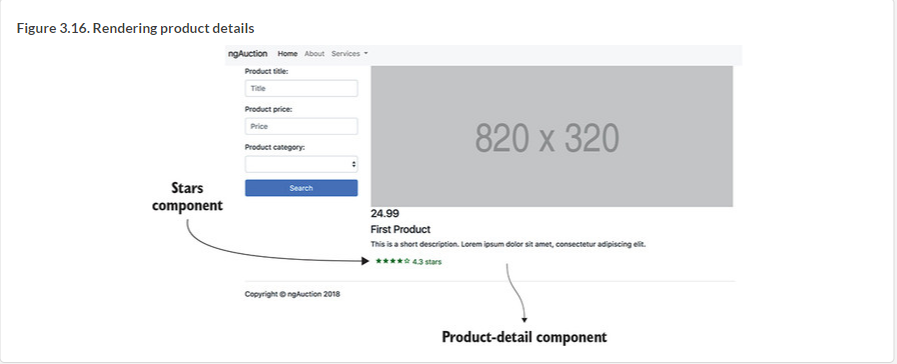
Modify the content of product-detail.component.html to look like the following listing.



ProductDetailComponent also uses <nga-stars>. For a change, let’s paint the stars dark green by adding the following listing’s style in product-detail.component.css.



Run the app with ng serve -o—the navigation to the product-detail view will work. Click the title of the first product, and the Router will create an instance of the ProductDetailComponent. The browser will show the product details, as shown in [figure 3.16](https://livebook.manning.com/book/angular-development-with-typescript-second-edition/chapter-3/1#!/book/angular-development-with-typescript-second-edition/chapter-3/ch03fig16).



Run this app with ng serve -o to see the landing page of ngAuction. Note that in the product-detail view, the stars are shown in the dark green color, whereas on the landing page, they’re dark red.

### Summary

* You can configure routes in parent and child components.
* You can pass data to routes during navigation.
* During navigation, the router renders components in the area defined by the <router-outlet> tag.