UNIT-IV

Adding Dynamic Front End with Angular

Creating an Angular application with Typescript: getting up and running with Angular, working with angular components, getting data from an API, putting and Angular application into production.

Building a single-page application with Angular: Foundations: Adding navigation in an Angular SPA, building a modular app using multiple nested components, adding geo-location to find places near you, and safely binding HTML content.

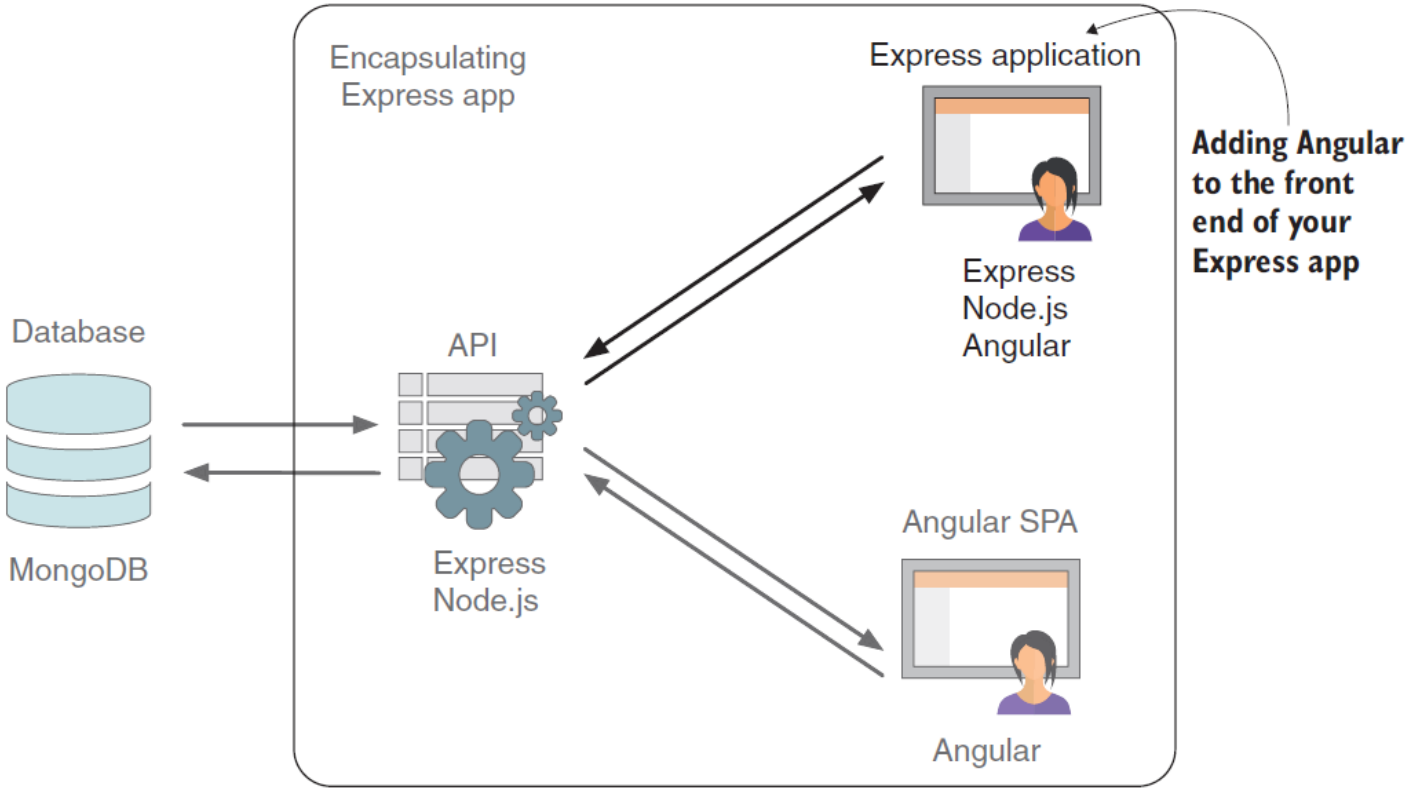


Figure 8.1 This chapter focuses on adding Angular to the front end of the existing Express application

Creating an Angular application with TypeScript

TypeScript

- TypeScript is a superset of JavaScript, so it's JavaScript with some additional bits and pieces.
- TypeScript is the preferred language for creating Angular applications.

```
PS D:\My Stuff\mine\UDAYDIVYA\uday\AANBA\25-26\Sem-1\MEAN Stack\lab\Loc8r\Loc8r> npm i -g typescript

added 1 package in 4s
PS D:\My Stuff\mine\UDAYDIVYA\uday\AANBA\25-26\Sem-1\MEAN Stack\lab\Loc8r\Loc8r> tsc -v
Version 5.9.2
```

TypeScript

Relationship between TypeScript and JavaScript

TypeScript

```
1 class Helloworld{
2     constructor(public message:string)
3     {}
4 }
5
6 var hello=new Helloworld('Hello TypeScript');
7 console.log(hello.message);
8
```

TypeScript code transplied to JavaScript code

JavaScript

```
1 var Helloworld = (function () {
2     function Helloworld(message) {
3         this.message = message;
4     }
5     return Helloworld;
6 }());
7 var hello = new Helloworld('Hello TypeScript');
8 console.log(hello.message);
9
```

From the above code, the TypeScript class Helloworld is converted to a self-invoking function in JavaScript when transpiled.

You can use TypeScript's online playground editor to see how TypeScript gets converted into JavaScript.

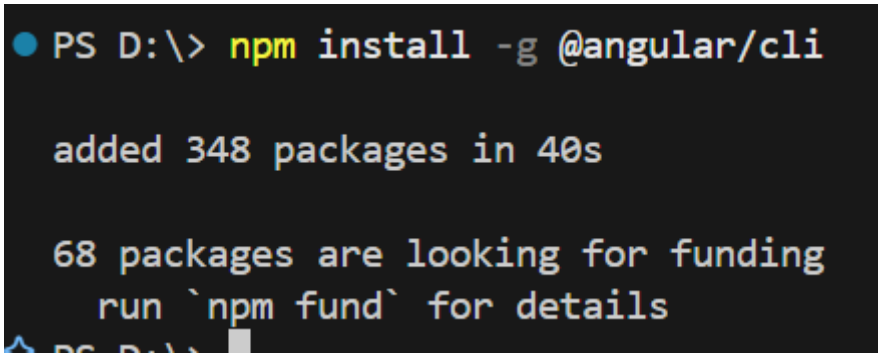
1. Getting up and running with Angular

1. Using the command line to create a boilerplate Angular app

Installing Angular

Angular is simple to install as long as you have Node and npm already installed. What you actually install is the Angular CLI as a global npm package. To do so, run the following command in terminal:

```
$ npm install -g @angular/cli
```



1. Getting up and running with Angular

1. Using the command line to create a boilerplate Angular app

Run the **ng help** in the command line. The options you're interested in are the following:

- **--skipGit**, to skip the default Git initialisation and first commit. By default, ng new initialises the folder as a new Git repository, but you don't need to do that, because you're going to create it inside an existing Git repo.
- **--skipTests**, to skip installation of some testing files.
- **--directory**, to specify the folder where you want the application to be generated.
- **--defaults** forces default Angular settings to be used.

Putting all this together, you'll use a command to create a boilerplate Angular application inside a new folder called app_public:

```
ng new loc8r-public --skipGit=true --skipTests=true –
defaults=true –directory app_public
```

1. Getting up and running with Angular

1. Using the command line to create a boilerplate Angular app

When everything is installed, the contents of your app_public folder should look like figure 8.2. You may notice that this project has its own package.json file and node_modules folder, so it looks a lot like a Node application. The src folder is where you'll do most of your work.

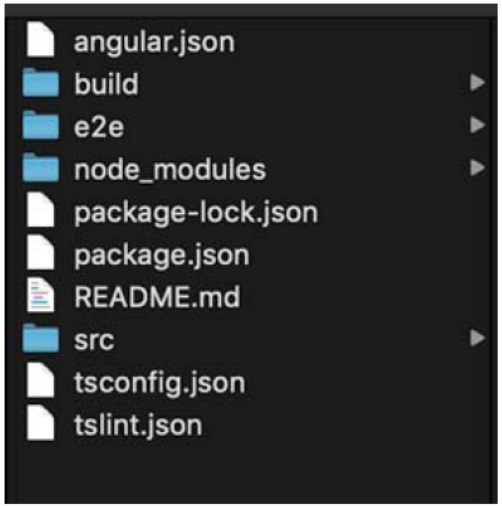


Figure 8.2 Default contents of a freshly generated Angular project

1. Getting up and running with Angular

2. Running the Angular app

- To run the app, head to your **app_public** folder in terminal, and run the following command:

  $ ng serve

  ?wdm?: Compiled successfully

- When you see this above message, your app is ready to view on port 4200

  http://localhost:4200

1. Getting up and running with Angular

2. Running the Angular app

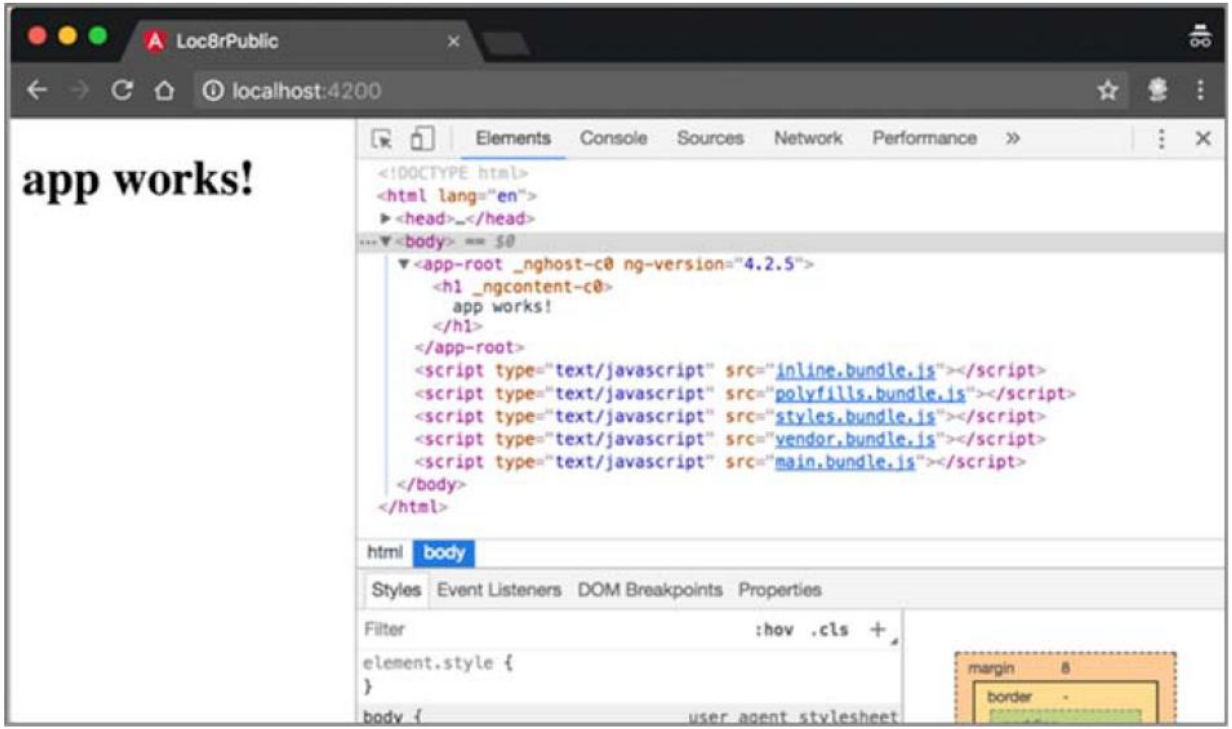


Figure 8.3 The autogenerated Angular app working in the browser alongside the generated HTML

1. Getting up and running with Angular

3. The source code behind the application

- Angular applications are built with components, which are compiled into modules.
- ***Component*** and ***module*** are terms that are often used loosely to label the building blocks of an application.
- A **component** handles a specific piece of functionality, and a **module** contains one or more components working together.

1. Getting up and running with Angular

3. The source code behind the application

Listing 8.1 The default contents of the src/index.html file

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>Loc8rPublic</title>
  <base href="/">

  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="icon" type="image/x-icon" href="favicon.ico">
</head>
<body>
  <app-root></app-root>
</body>
</html>
```

1 The title has been created from the application name.

2 The only tag in the body is the app-root.

1. Getting up and running with Angular

3. The source code behind the application

THE MAIN MODULE

Angular applications are built with components, which are compiled into modules.

In **src/app**,

- **app.module.ts** is the central point of your Angular module, and it's where all of the components are brought together.

This file does the following things:

- Imports various pieces of Angular functionality that the app will use
- Imports the components that the app will use
- Describes the module by using a decorator
- Exports the module

1. Getting up and running with Angular

3. The source code behind the application

THE MAIN MODULE

Listing 8.2 The default contents of the src/app/app.module.ts file

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';

import { AppComponent } from './app.component';

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

Imports various Angular modules that the application will use

Imports a component from the file system

Describes the module by using a decorator . . .

. . . including the entry point into the application

Exports the module

1. Getting up and running with Angular

3. The source code behind the application

THE DEFAULT BOOTSTRAPPED COMPONENT

- In the **app_public/src/app** folder, alongside the module file, you can see three app.component files:
  - *app.component.css*
  - *app.component.html*
  - *app.component.ts*

- These files are typical for any component. The CSS and HTML files define the styles and markup for the component, and the TS file defines the behavior in TypeScript.
- The CSS file is empty, but the HTML file contains the following code:

1. Getting up and running with Angular

3. The source code behind the application

THE DEFAULT BOOTSTRAPPED COMPONENT **app.component.html**

```
<div class="text-center">
  <h1>🚀 Welcome to My Angular Project</h1>
  <p>Built with Angular {{ title }}!</p>

  <nav>
    <ul style="list-style:none; padding:0;">
      <li><a href="https://angular.io/tutorial" target="_blank" rel="noopener">Tour of Heroes</a><
      <li><a href="https://angular.dev/cli" target="_blank" rel="noopener">CLI Documentation</a></
      <li><a href="https://blog.angular.io/" target="_blank" rel="noopener">Angular Blog</a></li>
    </ul>
  </nav>
</div>
```

1. Getting up and running with Angular

3. The source code behind the application

THE DEFAULT BOOTSTRAPPED COMPONENT

This component file does three main things:

- Imports what it needs from Angular
- Decorates the component, giving it the information that the app needs to run it
- Exports the component as a class

Listing 8.3 The default contents of app.component.ts

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

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'loc8r-public';
}
```

Imports the Component from the Angular core

Decorates the component

Exports the component as a class

1. Getting up and running with Angular

3. The source code behind the application

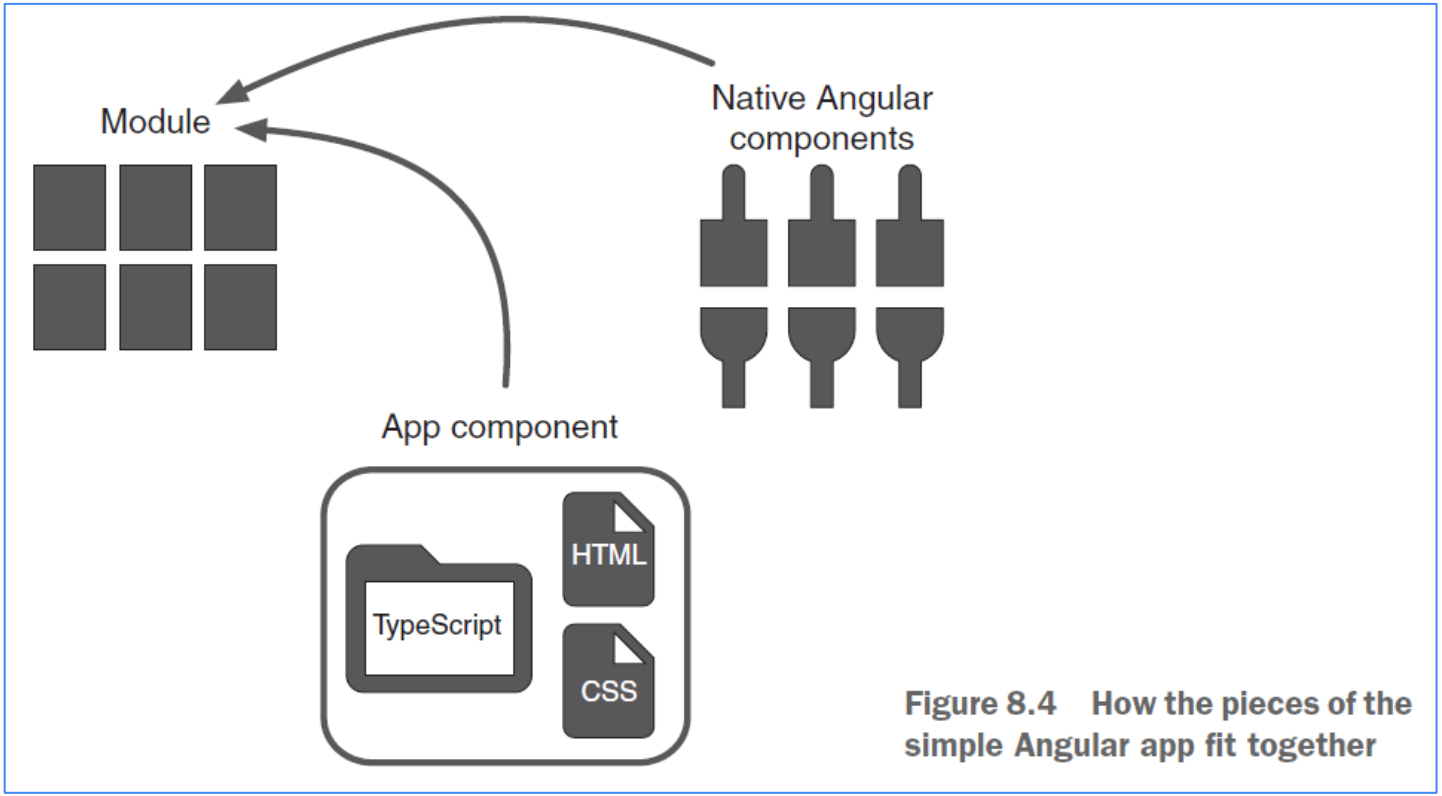


Figure 8.4 How the pieces of the simple Angular app fit together

2. Working with Angular components

1. Creating a new home-list component

- In terminal, from within the **app_public** folder, run the following command:
  $ ng generate component home-list

- This command creates a new folder called **home-list** within the ***src*** folder.

- Create the **TypeScript, HTML,** and **CSS** files inside it, and also update the **app.module.ts** file to tell the module about the new component.

- You'll also see a **spec.ts** file in the new component folder. This file is a template for unit testing.

2. Working with Angular components

1. Creating a new home-list component

MAKING IT THE DEFAULT COMPONENT

- The new home-list component will be the basis for this Angular module, so you need to make it the default component.

- You do this inside the **app.module.ts** file by changing the bootstrap value inside the module decorator from **AppComponent** to **HomeListComponent**.

- **AppComponent** is no longer needed, so you can remove the import statement, remove it from the declarations, and even delete the files.

- The changes to **app.module.ts** are shown in the following listing.

2. Working with Angular components

1. Creating a new home-list component

MAKING IT THE DEFAULT COMPONENT

Listing 8.4 Changing to the new component in app.module.ts

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';

import { HomeListComponent } from './home-list/home-list.component';

@NgModule({
  declarations: [
    HomeListComponent
  ],
  imports: [
    BrowserModule
  ],
  providers: [],
  bootstrap: [HomeListComponent]
})
```

This line was added by the Angular CLI; delete the AppComponent import, as it's no longer needed.

Deletes AppComponent from the declarations array

Changes AppComponent to HomeListComponent for the bootstrap value

2. Working with Angular components

1. Creating a new home-list component

SETTING THE HTML TAG FOR THE COMPONENT

home-list.component.ts

Selector defines the tag on the page that the component will bind to.

```
@Component({
  selector: 'app-home-list',
  templateUrl: './home-list.component.html',
  styleUrls: ['./home-list.component.css']
})
```

2. Working with Angular components

1. Creating a new home-list component

SETTING THE HTML TAG FOR THE COMPONENT

Replace *AppComponent* with *HomeListComponent* and file path in *main.server.ts* file

app_public > src > TS main.server.ts

```ts
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app/app.component';
import { config } from './app/app.config.server';

const bootstrap = () => bootstrapApplication(AppComponent, config);

export default bootstrap;
```

app_public > src > TS main.server.ts

```ts
import { bootstrapApplication } from '@angular/platform-browser';
import { HomeListComponent } from './app/home-list/home-list.component';
import { config } from './app/app.config.server';

const bootstrap = () => bootstrapApplication(HomeListComponent, config);

export default bootstrap;
```

2. Working with Angular components

1. Creating a new home-list component

MAKING IT THE DEFAULT COMPONENT

- Open the **index.html** file in the **src** folder, and change the **app-root** tag to **app-home-list** so that it looks like this:

```
<body>
<app-home-list></app-home-list>
</body>
```

Figure 8.7 Confirmation that the new `home-list` component is working as the default in the application

2. Working with Angular components

2. Creating the HTML template

GETTING THE HTML MARKUP

The easiest way to get the HTML is to run the Express app and go to the homepage in a browser. Different browsers have slightly different ways of getting the HTML, but are similar to the following procedure in Chrome:

1. Right-click in the HTML area and choose Inspect Element from the contextual menu.
2. Highlight the element.
3. Select Copy, and then Copy Outer HTML.

Paste this into home-list.component.html, replacing the existing contents, and you should see something like the following.

2. Working with Angular components

2. Creating the HTML template

GETTING THE HTML MARKUP

Listing 8.5 Some static HTML for home-list.component.html to get started

```
<div class="card">
  <div class="card-block">
    <h4>
      <a href="/location/590d8dc7a7cb5b8e3f1bfc48">Costy</a>
      <small> 
        <i class="far fa-star"></i>
        <i class="far fa-star"></i>
        <i class="far fa-star"></i>
        <i class="far fa-star"></i>
        <i class="far fa-star"></i>
      </small>
      <span class="badge badge-pill badge-default float-
      ➥right">14.0km</span>

    </h4>
    <p class="address">High Street, Reading</p>
    <div class="facilities">
      <span class="badge badge-warning">hot drinks</span>
      <span class="badge badge-warning">food</span>
      <span class="badge badge-warning">power</span>
    </div>
  </div>
</div>
```

2. Working with Angular components

2. Creating the HTML template

CORS?

Browsers aren't allowed to access or request certain resources from a different domain, including requesting font files and making AJAX calls. This policy is known as the *same-origin policy*.

CORS (cross-origin resource sharing) is a mechanism that allows this to happen but can be set only from the server that hosts the resources. If the server denies you, there's nothing you can do from the browser side to change it.

2. Working with Angular components

2. Creating the HTML template

CORS?

Listing 8.6 Adding the CSS files to index.html for the Angular app

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>Loc8rPublic</title>
  <base href="/">

  <link rel="stylesheet" href="assets/stylesheets/bootstrap.min.css">
  <link rel="stylesheet" href="assets/stylesheets/all.min.css">
  <link rel="stylesheet" href="assets/stylesheets/style.css">

  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="icon" type="image/x-icon" href="favicon.ico">
</head>
<body>
  <app-home-list></app-home-list>

  <script src="https://code.jquery.com/jquery-3.3.1.slim.min.js"
          integrity="sha384-q8i/X+965DzO0rT7abK41JStQIAqVgRVzpbo5smXKp4YfRvH+8abtTE1Pi6jizo"
          crossorigin="anonymous">
  </script>

  <script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.14.3/umd/popper.min.js"
          integrity="sha384-ZMP7rVo3mIykV+9J3UJ46jBk0WLaUAAn689aCwoqbBJ1SnjAK/1WcWPIpm49"
          crossorigin="anonymous">
  </script>

  <script src="assets/javascripts/bootstrap.min.js"></script>
</body>
</html>
```

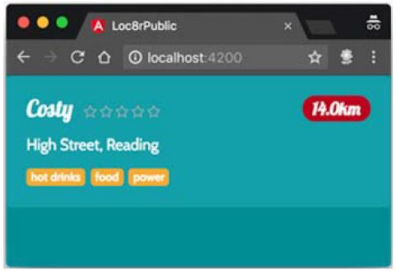


Figure 8.8 The Angular app displaying static content and using the styles and fonts

2. Working with Angular components

3. Moving data out of the template into the code

DEFINING A CLASS TO GIVE STRUCTURE TO DATA

Types in TypeScript

The different data types that TypeScript accepts are as follows:

- `String`—Text values.
- `number`—Any numerical value; integers and decimals are treated the same way.
- `boolean`—True or false.
- `Array`—An array of a given type of data.
- `enum`—A way of giving friendly names to a set of numeric values.
- `Any`—This data type can be anything, like how JavaScript is by default.
- `Void`—The absence of a type, typically used for functions that don't return anything.

2. Working with Angular components

3. Moving data out of the template into the code

DEFINING A CLASS TO GIVE STRUCTURE TO DATA

Listing 8.7 Defining the `Location` class in home-list.component.ts

```
import { Component, OnInit } from '@angular/core';

export class Location {
  _id: string;
  name: string;
  distance: number;
  address: string;
  rating: number;
  facilities: string[];
}
```

Creates and exports a class called Location

Defines the class members and their types . . .

. . . including an array of strings

2. Working with Angular components

3. Moving data out of the template into the code

CREATING AN INSTANCE OF THE LOCATION CLASS

Listing 8.8 Defining a `location` with the `Location` class in home-list.component.ts

```
export class HomeListComponent implements OnInit {

  constructor() { }

  location: Location = {
    _id: '590d8dc7a7cb5b8e3f1bfc48',
    name: 'Costy',
    distance: 14.0,
    address: 'High Street, Reading',
    rating: 3,
    facilities: ['hot drinks', 'food', 'power']
  };

  ngOnInit() {
  }

}
```

2. Working with Angular components

4. Using class member data in the HTML template

Listing 8.9 Binding the first pieces of data in home-list.component.html

```
<div class="card">
  <div class="card-block">
    <h4>
      <a href="/location/{{location._id}}">{{location.name}}</a>
      <small> 
        <i class="far fa-star"></i>
        <i class="far fa-star"></i>
        <i class="far fa-star"></i>
        <i class="far fa-star"></i>
        <i class="far fa-star"></i>
      </small>
      <span class="badge badge-pill badge-default float-
      ➥right">{{location.distance}}km</span>
    </h4>
    <p class="address">{{location.address}}</p>
    <div class="facilities">
      <span class="badge badge-warning">hot drinks</span>
      <span class="badge badge-warning">food</span>
      <span class="badge badge-warning">power</span>
    </div>
  </div>
</div>
```

2. Working with Angular components

4. Using class member data in the HTML template

FACILITIES: Looping through an array of items in an html template

Listing 8.10 Using *ngFor to loop through an array in home-list.component.html

```
<div class="facilities">
  <span *ngFor="let facility of location.facilities" class="badge
  ➥badge-warning">{{facility}}</span>
</div>
```

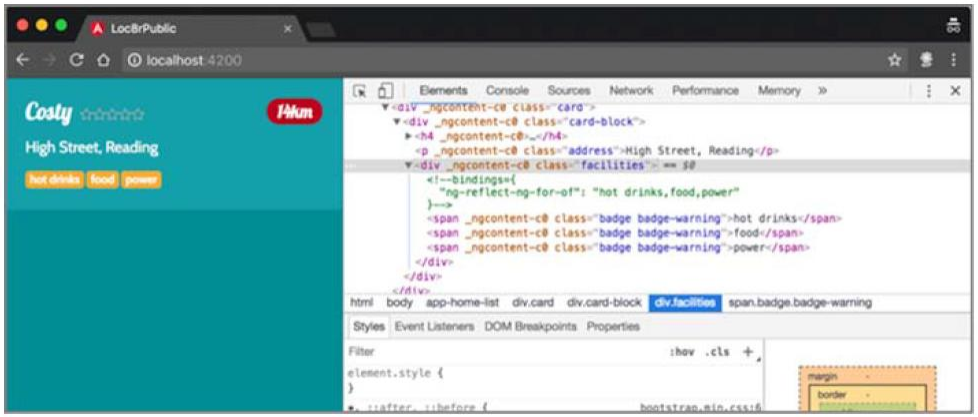


Figure 8.9 The output of Angular looping through the array of facilities

2. Working with Angular components

4. Using class member data in the HTML template

RATING STARS: USING ANGULAR EXPRESSIONS TO SET CSS CLASSES

Listing 8.11 Binding the ternary expressions to generate ratings-stars classes

```
<small> 
  <i class="fa{{ location.rating < 1 ? 'r' : 's' }} fa-star"></i>
  <i class="fa{{ location.rating < 2 ? 'r' : 's' }} fa-star"></i>
  <i class="fa{{ location.rating < 3 ? 'r' : 's' }} fa-star"></i>
  <i class="fa{{ location.rating < 4 ? 'r' : 's' }} fa-star"></i>
  <i class="fa{{ location.rating < 5 ? 'r' : 's' }} fa-star"></i>
</small>
```

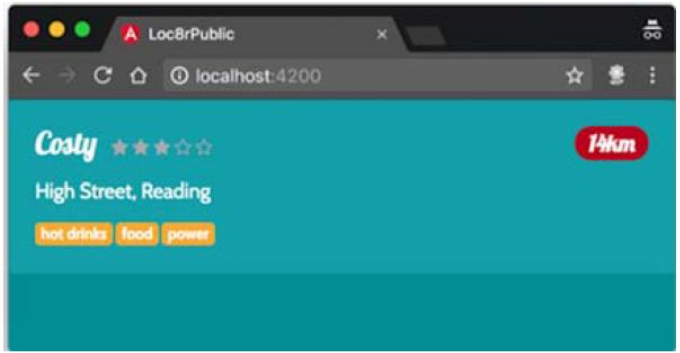


Figure 8.10 Showing the rating stars correctly, using Angular expression bindings to generate the correct class

2. Working with Angular components

4. Using class member data in the HTML template

DISTANCES: CREATING A CUSTOM PIPE

Listing 8.12 Creating the distance format pipe in distance.pipe.ts

```
transform(distance: number): string {
  const isNumeric = function (n) {
    return !isNaN(parseFloat(n)) && isFinite(n);
  };

  if (distance && isNumeric(distance)) {
    let thisDistance = '0';
    let unit = 'm';
    if (distance > 1000) {
      thisDistance = (distance / 1000).toFixed(1);
      unit = 'km';
    } else {
      thisDistance = Math.floor(distance).toString();
    }
    return thisDistance + unit;
  } else {
    return '?';
  }
}
```

2. Working with Angular components

4. Using class member data in the HTML template

DISTANCES: CREATING A CUSTOM PIPE

```
<span class="badge badge-pill badge-default float-
➥right">{{location.distance | distance}}</span>
```

You can also check this out in the browser (see figure 8.11).

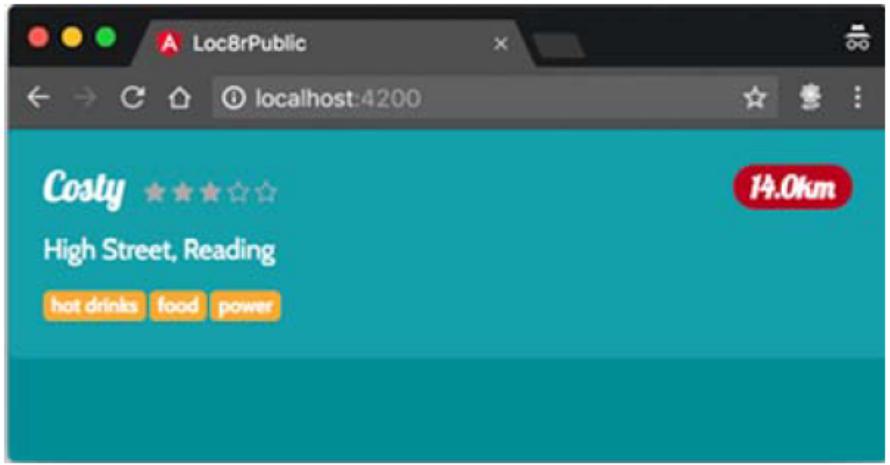


Figure 8.11 Using the Angular pipe to format the distance supplied in meters

2. Working with Angular components

4. Using class member data in the HTML template

DISTANCES: CREATING A CUSTOM PIPE

Listing 8.13 Changing the locations instantiation to an array in home-list.component.ts

```
locations: Location[] = [{
  _id: '590d8dc7a7cb5b8e3f1bfc48',
  name: 'Costy',
  distance: 14000.1234,
  address: 'High Street, Reading',
  rating: 3,
  facilities: ['hot drinks', 'food', 'power']
}, {
  _id: '590d8dc7a7cb5b8e3f1bfc48',
  name: 'Starcups',
  distance: 120.542,
  address: 'High Street, Reading',
  rating: 5,
  facilities: ['wifi', 'food', 'hot drinks']
}];
```

2. Working with Angular components

4. Using class member data in the HTML template

DISTANCES: CREATING A CUSTOM PIPE

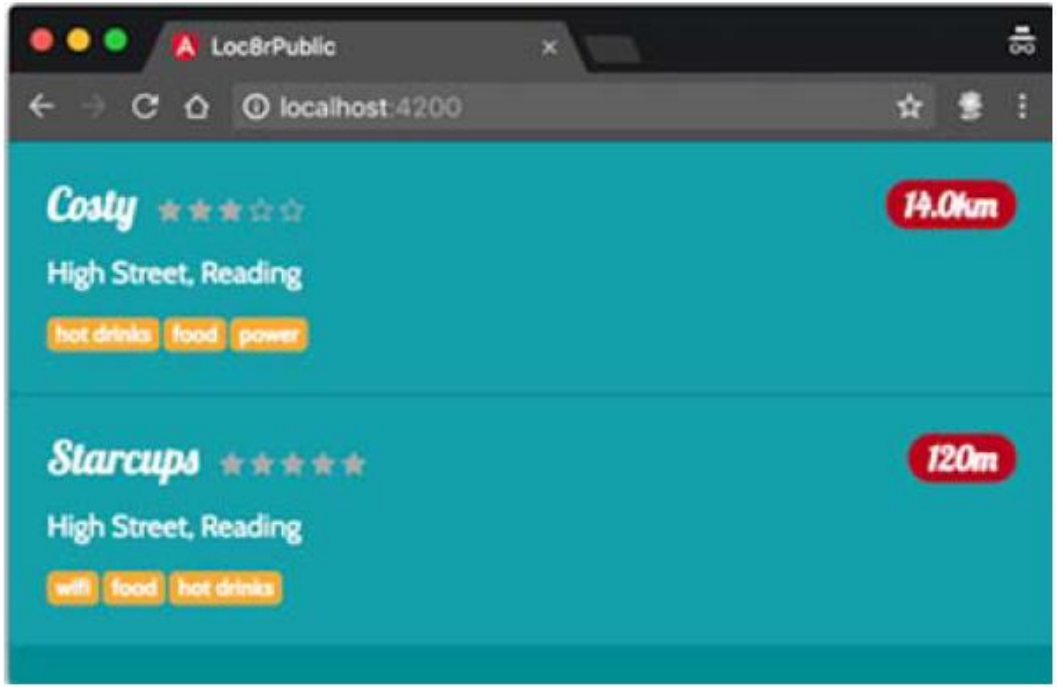


Figure 8.12 Updating the component to display multiple locations in a list

3. Getting data from an API

1. Creating a data service

- To interact with an API, you need to use another building block of Angular applications: a ***service***. A service works in the background and isn't directly connected to the user interface

In the **app_public** folder, run the following command in terminal
$ ng generate service loc8r-data

Services are generated with a **providedIn** value passed to the **Injectable** decorator, which defaults to **'root'**.

```
import { Injectable } from '@angular/core';

@Injectable({
  providedIn: 'root'
})
export class Loc8rDataService {

  constructor() { }

}
```

3. Getting data from an API

1. Creating a data service

Enabling HTTP requests and promise handling in a service

Observables and Promises

Observables and Promises are great ways of handling asynchronous requests. Observables return chunks of data in a stream, whereas Promises return complete sets of data. Angular includes the RxJS library for working with observables, including converting them into Promises.

Making **HTTP** requests

Making the HTTP request to the API is straightforward, involving only a few steps:

1. Build the URL to call.
2. Tell the HTTP service to make a request for the URL.
3. Convert the Observable response to a Promise.
4. Convert the response to JSON.
5. Return the response.
6. Catch, handle, and return errors.

3. Getting data from an API

1. Creating a data service

Making **HTTP** requests

Listing 8.14 Making and returning the HTTP request to your API in loc8r-data.service.ts

```
private apiBaseUrl = 'http://localhost:3000/api';

public getLocations(): Promise<Location[]> {
  const lng: number = -0.7992599;
  const lat: number = 51.378091;
  const maxDistance: number = 20;
  const url: string = `${this.apiBaseUrl}/locations?lng=
    ➥${lng}&lat=${lat}&maxDistance=${maxDistance}`;
  return this.http
    .get(url)
    .toPromise()
    .then(response => response as Location[])
    .catch(this.handleError);
}

private handleError(error: any): Promise<any> {
  console.error('Something has gone wrong', error);
  return Promise.reject(error.message || error);
}
```

Builds the URL to the API, using parameters for future enhancements

Returns the Promise

Makes the HTTP GET call to the URL you built

Converts the Observable response to a Promise

Converts the response to a JSON object of type Location

Handles and returns any errors

3. Getting data from an API

2. Using a data service

Importing the service into the component

Listing 8.15 Making your service available to the component in home-list.component.ts

```
import { Component, OnInit } from '@angular/core';
import { Loc8rDataService } from '../loc8r-data.service';

export class Location {
  _id: string;
  name: string;
  distance: number;
  address: string;
  rating: number;
  facilities: [string];
}

@Component({
  selector: 'app-home-list',
  templateUrl: './home-list.component.html',
  styleUrls: ['./home-list.component.css']
})
export class HomeListComponent implements OnInit {

  constructor(private loc8rDataService: Loc8rDataService) { }
```

Imports the service from the source code file

Injects the service into the component using the constructor

3. Getting data from an API

2. Using a data service

Importing the service into the component

Listing 8.16 Creating a function to call the data service from home-list.component.ts

Changes the locations declaration to have no default value

```
public locations: Location[];

private getLocations(): void {
  this.loc8rDataService
    .getLocations()
      .then(foundLocations => this.locations = foundLocations);
}
```

Defines a getLocations method that accepts no parameters and returns nothing

Calls your data service method

Updates the locations array with the contents of the response

3. Getting data from an API

2. Using a data service

Importing the service into the component

Head back to app.js in the root of the application, and add the following bold font lines before the routes are used:

```
app.use('/api', (req, res, next) => {
  res.header('Access-Control-Allow-Origin', 'http://localhost:4200');
  res.header('Access-Control-Allow-Headers', 'Origin, X-Requested-With,
  ➥Content-Type, Accept');
  next();
});
app.use('/', indexRouter);
app.use('/api', apiRouter);
```

3. Getting data from an API

2. Using a data service

Importing the service into the component

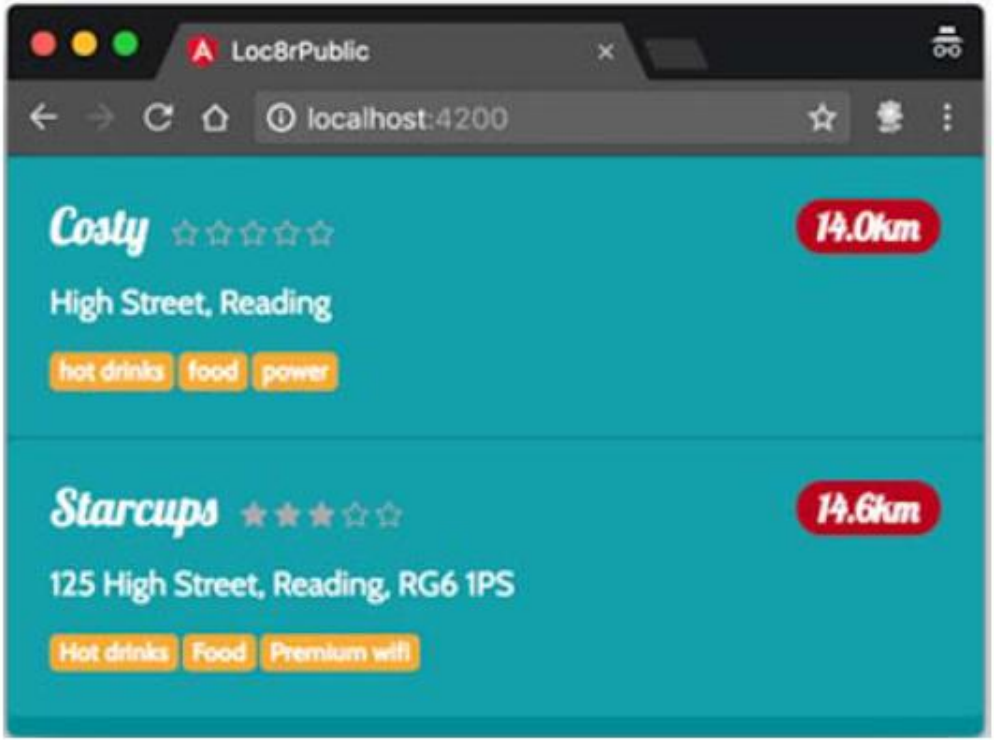


Figure 8.13 Your Angular component is now displaying data brought in from the API.

4. Putting an Angular application into production

1. Building an Angular application for production

- To create a production build of your application in the folder app_public/build, run the following command in terminal from the app_public folder:

  $ **ng build --prod --output-path build**

UNIT-IV

Adding Dynamic Front End with Angular

Creating an Angular application with Typescript: getting up and running with Angular, working with angular components, getting data from an API, putting and Angular application into production.

Building a single-page application with Angular: Foundations: Adding navigation in an Angular SPA, building a modular app using multiple nested components, adding geo-location to find places near you, and safely binding HTML content.

Building a single-page application with Angular: Foundations

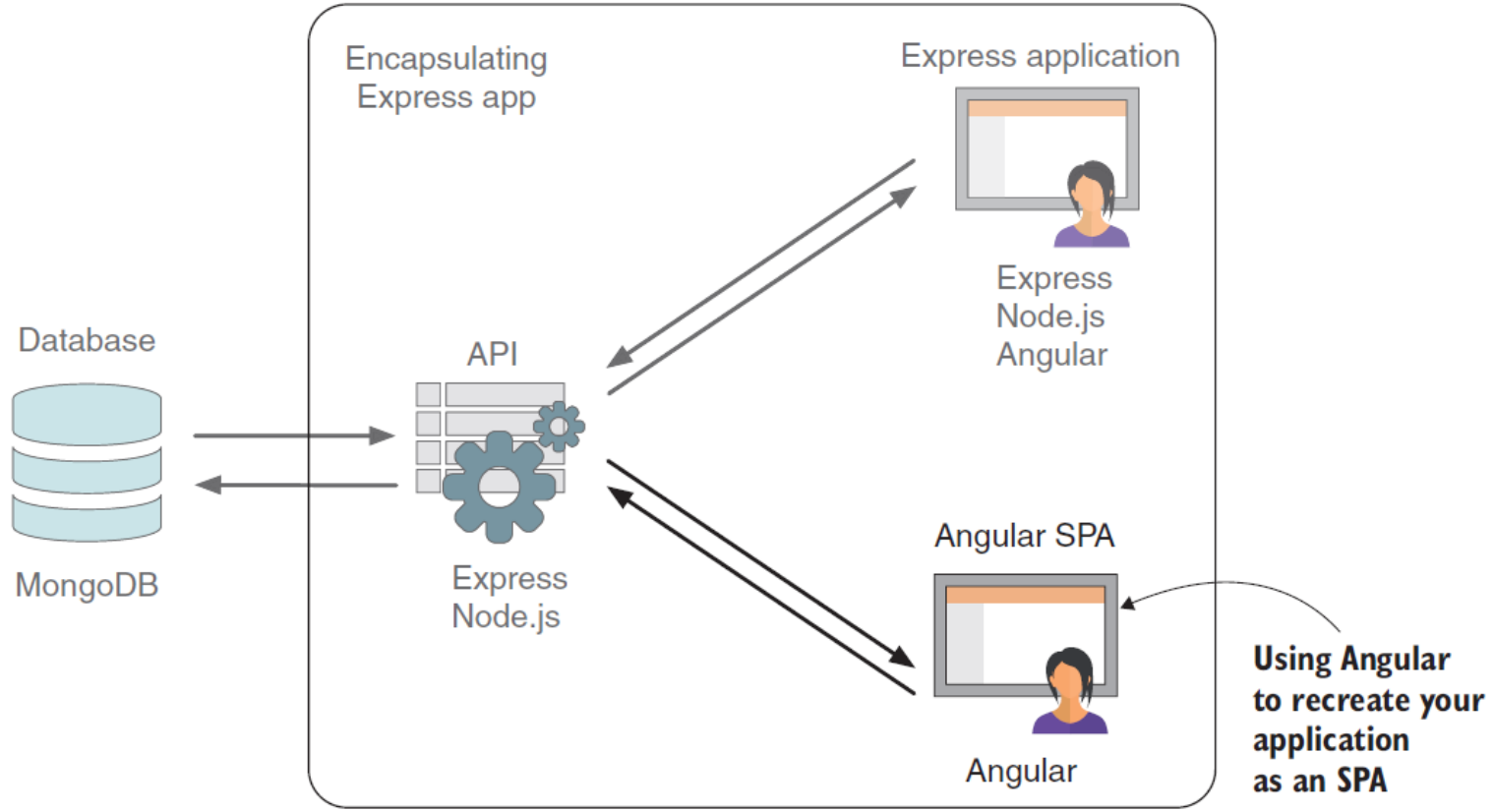


Figure 9.1 This chapter recreates the Loc8r application as an Angular SPA, moving the application logic from the back end to the front end.

1. Adding navigation in an Angular SPA

1. Importing the Angular router and defining the first route

The Angular router needs to be imported into app.module.ts, which is also where you'll define the routes. The router is imported from `@angular/router` as `RouterModule`, which should be placed with the other Angular imports at the top of app.module.ts.

Listing 9.1 Adding the `RouterModule` to the list of imports in app.module.ts

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { HttpClientModule } from '@angular/common/http';
import { RouterModule } from '@angular/router';
```

Imports the Angular RouterModule

In the same file, in the `@NgModule` decorator, all these modules are listed in the imports section. You need to do the same with `RouterModule`, but in this case, you also need to pass it the routing configuration you want.

1. Adding navigation in an Angular SPA

2. Routing configuration

The routing configuration is an array of objects, each object specifying one route. The properties for each route are

- path—The URL path to match
- component—The name of the Angular component to use

Listing 9.2 Adding the routing configuration to the decorator in app.module.ts

```
@NgModule({
  declarations: [
    HomeListComponent,
    DistancePipe
  ],
```

1. Adding navigation in an Angular SPA

2. Routing configuration

```
  imports: [
    BrowserModule,
    HttpClientModule,
    RouterModule.forRoot([
      {
        path: '',
        component: HomeListComponent
      }
    ])
  ],
  providers: [],
  bootstrap: [HomeListComponent]
})
```

Adds the RouterModule to the imports, calling the forRoot method

Defines the homepage route as an empty string

Specifies the HomeListComponent as the one to use for this route

1. Adding navigation in an Angular SPA

3. Creating a component for the framework and navigation

First, create a new component called `framework` by running the following in terminal from the app_public directory:

```
$ ng generate component framework
```

This command creates a new framework folder inside app_public/src/app/ and also generates all the files you need. Find the framework.component.html file, and add all the HTML shown in the following listing, which is pretty much what the HTML content of layout.pug would look like if converted to HTML

1. Adding navigation in an Angular SPA

3. Creating a component for the framework and navigation

Listing 9.3 Adding the HTML for the framework in framework.component.html

```
<nav class="navbar fixed-top navbar-expand-md navbar-light">
  <div class=”container”>
    <a href="/" class="navbar-brand">Loc8r</a>
      <button type="button" data-toggle="collapse" data-target=
      ➥"#navbarMain"class="navbar-toggler">
        <span class="navbar-toggler-icon"></span>
      </button>

    <div id="navbarMain" class="navbar-collapse collapse">
      <ul class="navbar-nav mr-auto">
        <li class="nav-item">
          <a href="/about/" class="nav-link">About</a>
        </li>
      </ul>
```

Sets up the navigation section

1. Adding navigation in an Angular SPA

3. Creating a component for the framework and navigation

```
    </div>
  </div>
</nav>
<div class="container content">
  <footer>
    <div class="row">
      <div class="col-12">
        <small>&copy; Getting Mean - Simon Holmes/Clive Harber 2018</small>
      </div>
    </div>
  </footer>
</div>
```

Creates the main container

Nests the footer inside the main container

Component:

bootstrap: [**FrameworkComponent**]

```
@Component({
  selector: 'app-framework',
  templateUrl: './framework.component.html',
  styleUrls: ['./framework.component.css']
})
```

Listing 9.4 Updating index.html file to use the new `framework` component

```
<body>
  <app-framework></app-framework>
</body>
```

Replaces the home-list component for the app-framework

1. Adding navigation in an Angular SPA

3. Creating a component for the framework and navigation

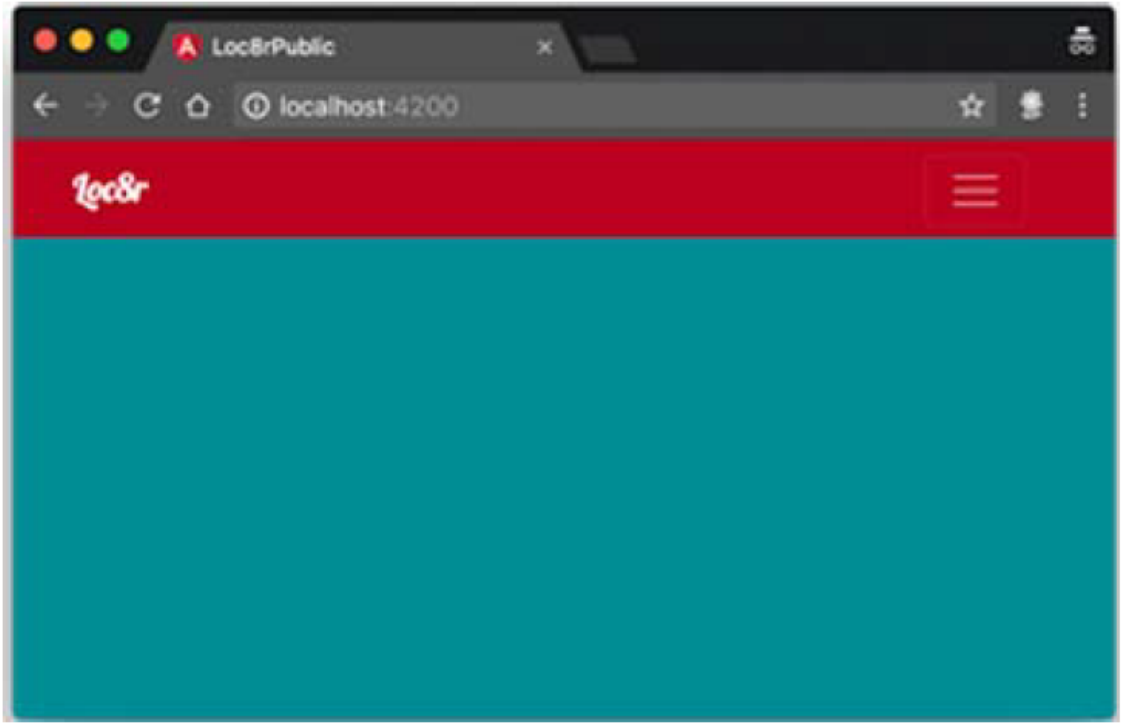


Figure 9.2 Showing the `framework` component by default instead of the listing

1. Adding navigation in an Angular SPA

4. Defining where to display the content using router-outlet

Listing 9.5 Adding `router-outlet` to framework.component.html

```
<div class="container">
  <router-outlet></router-outlet>
  <footer>
    <div class="row">
      <div class="col-12"><small>&copy; Getting Mean - Simon Holmes/Clive
     Harber 2018</small></div>
    </div>
  </footer>
</div>
```

Outlet for the router; Angular uses the URL to find the component and injects it here.

1. Adding navigation in an Angular SPA

5. Navigating between pages

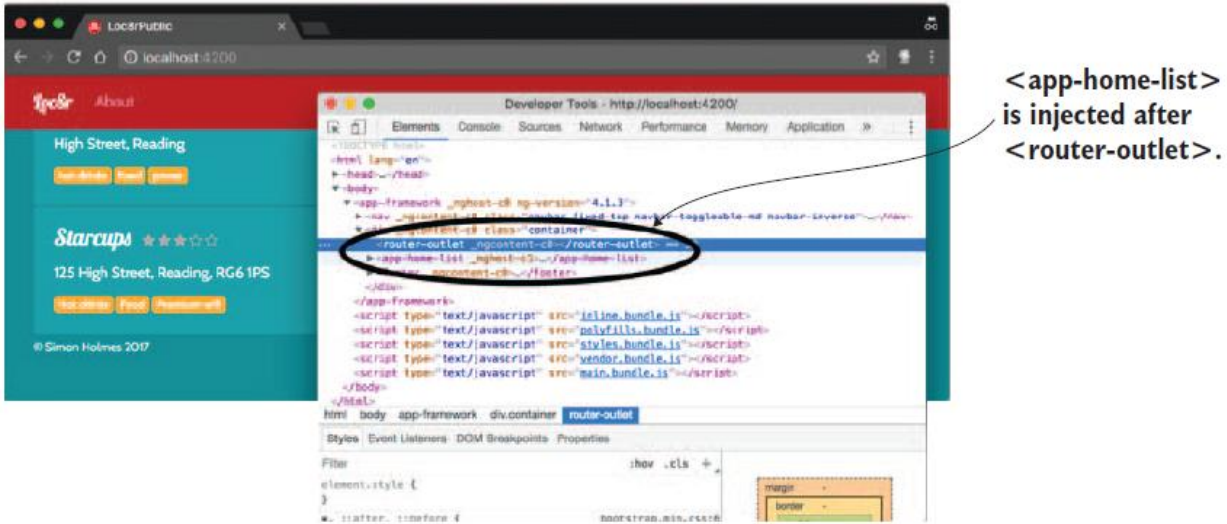


Figure 9.3 The routed component—the listing information—is now being displayed on the homepage route, with the HTML being injected after the `<router-outlet>` tag.

Creating the `about` component with Angular CLI should be familiar by now. In terminal, in the app_public folder, run the following `generate` command:

```
$ ng generate component about
```

1. Adding navigation in an Angular SPA

5. Navigating between pages

Defining A New Route

- To configure the route for the About page in **app.module.ts.**
- Open **about.component.ts** to find it in the export line:
  export class AboutComponent implements OnInit.
- Add the new route in **app.module.ts.**

Listing 9.6 Defining the new about route in app.module.ts

```
RouterModule.forRoot([
  {
    path: '',
    component: HomeListComponent
  },
  {
    path: 'about',
    component: AboutComponent
  }
])
```

1. Adding navigation in an Angular SPA

5. Navigating between pages

Setting Angular Navigation Links

Listing 9.7 Defining the navigation router links in framework.component.html

```
<a routerLink="" class="navbar-brand">Loc8r</a>
<div id="navbarMain" class="navbar-collapse collapse">
  <ul class="navbar-nav mr-auto">
    <li class="nav-item">
      <a routerLink="about" class="nav-link">About</a>
    </li>
  </ul>
</div>
```

Empty routerLink path pointing to the default component

about path to cause navigation to the about component

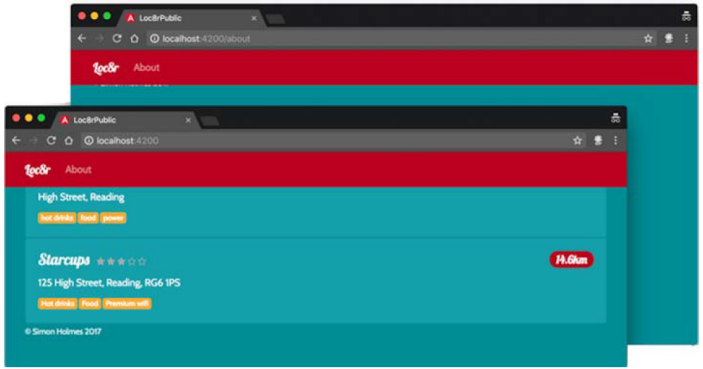


Figure 9.4 Using the navigation buttons to switch between the homepage and the About page—an Angular SPA!

2. Building a modular app using multiple nested components

- This process gives you a modular application, so you can reuse pieces in different places in the application.

The homepage has three main sections:

- Page header
- List of locations
- Sidebar

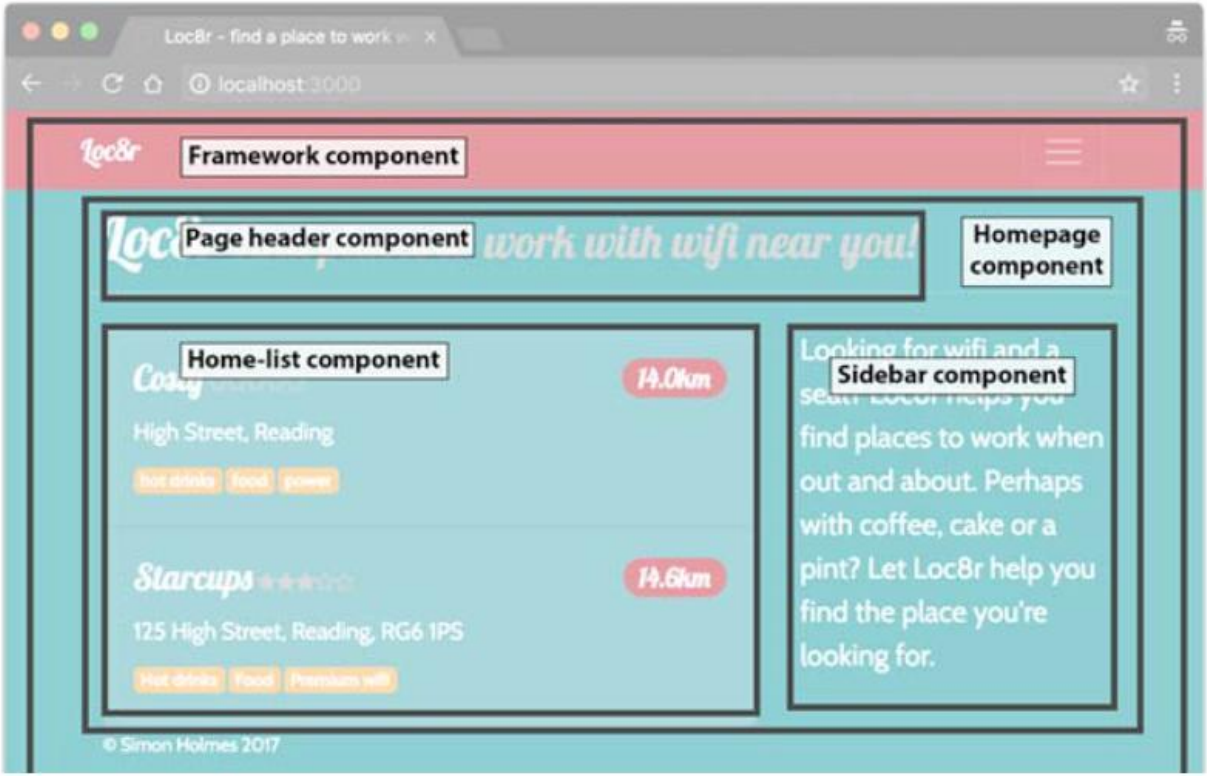


Figure 9.6 Breaking the homepage layout into components, using two levels of nesting

2.Building a modular app using multiple nested components

1. Creating the main homepage component

The `homepage` component contains all the HTML and information for the homepage— everything between the header and the footer.

Start by using the Angular CLI to generate the component in the now-familiar way (in terminal from the app_public folder):

`$ ng generate component homepage`

```
RouterModule.forRoot([
  {
    path: '',
    component: HomepageComponent
  },
  {
    path: 'about',
    component: AboutComponent
  }
])
```

2.Building a modular app using multiple nested components

1. Creating the main homepage component

Listing 9.8 Putting the HTML for homepage content in homepage.component.html

```
<div class="row banner">                          <-- The page header
  <div class="col-12">
    <h1>Loc8r
      <small>Find places to work with wifi near you!</small>
    </h1>
  </div>
</div>
<div class="row">
  <div class="col-12 col-md-8">                   <-- Container for the homepage listing component
    <div class="error"></div>
    <app-home-list></app-home-list>
  </div>                                          <-- The sidebar
  <div class="col-12 col-md-4">
   <p class="lead">Looking for wifi and a seat? Loc8r helps you
   ➥find places to work when out and about. Perhaps with coffee,
   ➥cake or a pint? Let Loc8r help you find the place you're
   ➥ looking for.</p>
 </div>
</div>
```

2.Building a modular app using multiple nested components

1. Creating the main homepage component

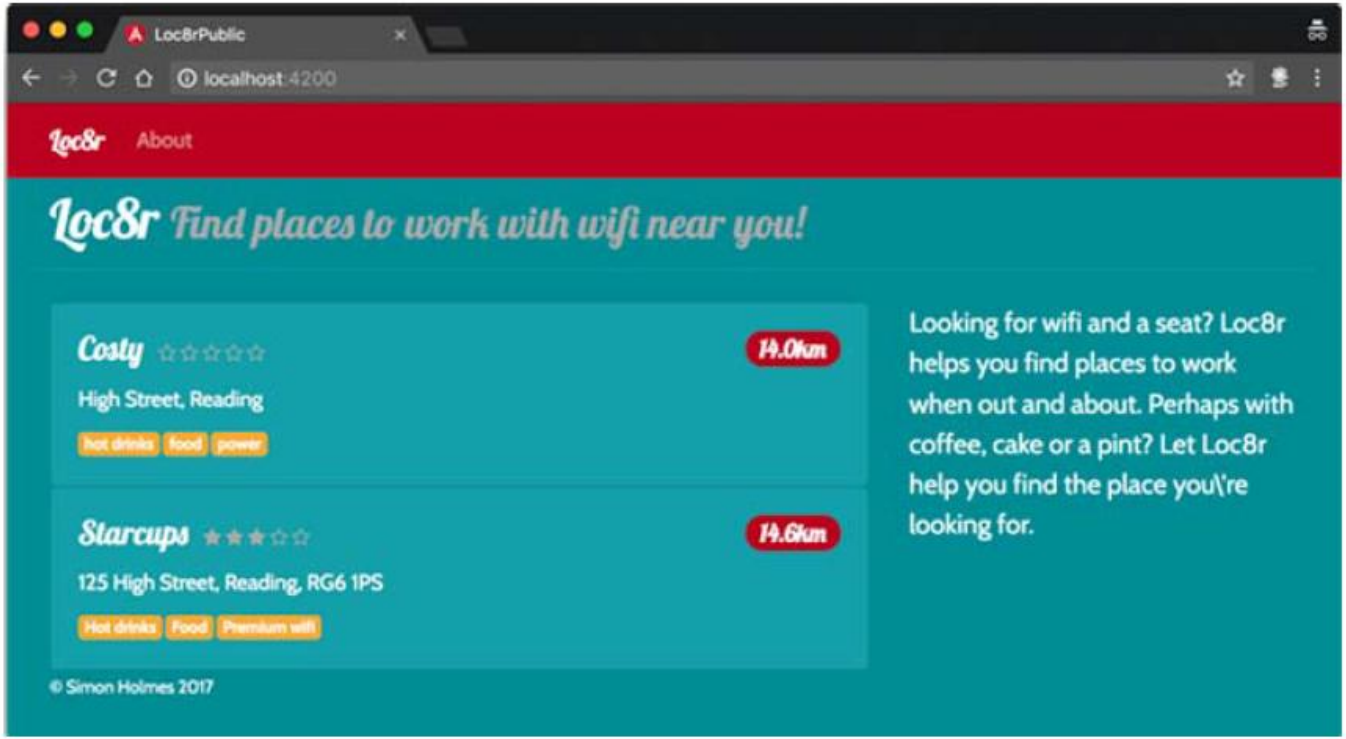


Figure 9.7 The homepage in Angular with the page header and sidebar hardcoded in the homepage component

2.Building a modular app using multiple nested components

2. Creating and using reusable subcomponents

Creating The Page-header Component

The first step is issuing the familiar component generation command in terminal:

```
$ ng generate component page-header
```

Following that command, copy the header content from the homepage HTML and paste it into page-header.component.html:

```
<div class="row banner">
  <div class="col-12">
    <h1>Loc8r
      <small>Find places to work with wifi near you!</small>
    </h1>
  </div>
</div>
```

2.Building a modular app using multiple nested components

2. Creating and using reusable subcomponents

Creating The Page-header Component

Listing 9.9 Replacing the page header HTML in homepage.component.html

```
<app-page-header></app-page-header>
<div class="row">
  <div class="col-12 col-md-8">
    <div class="error"></div>
    <app-home-list>Loading...</app-home-list>
  </div>
  <div class="col-12 col-md-4">
    <p class="lead">Looking for wifi and a seat? Loc8r helps you find
    ➥places to work when out and about. Perhaps with coffee, cake
    ➥or a pint? Let Loc8r help you find the place you\'re looking
    ➥for.</p>
  </div>
</div>
```

2.Building a modular app using multiple nested components

2. Creating and using reusable subcomponents

Defining The Data For The Page-header Component On The Homepage

Listing 9.10 Defining the homepage page header content in homepage.component.ts

```
export class HomepageComponent implements OnInit {
  constructor() { }

  ngOnInit() {
  }

  public pageContent = {
    header: {
      title: 'Loc8r',
      strapline: 'Find places to work with wifi near you!'
    }
  };

}
```

Creates a new class member to hold the page header content

2.Building a modular app using multiple nested components

2. Creating and using reusable subcomponents

Passing Data Into The Page-header Component

- The homepage class member **pageContent** is now available to the homepage HTML, but rather than use the data directly, you want to pass it through to the **page-header** component.
- Data is passed through to the **nested component** via a special binding in the HTML.
- The name of the binding is a property you define in the nested component, so it can be anything you want.
- In homepage.component.html, update <app-page-header> to include the binding:

<app-page-header[content]="pageContent.header"></app-page-header>

2.Building a modular app using multiple nested components

2. Creating and using reusable subcomponents

Accepting And Displaying Incoming Data In A Component

Listing 9.11 Telling page-header.component.ts to accept content as an `Input`

```
import { Component, OnInit, Input } from '@angular/core';

@Component({
  selector: 'app-page-header',
  templateUrl: './page-header.component.html',
  styleUrls: ['./page-header.component.css']
})
export class PageHeaderComponent implements OnInit {

  @Input() content: any;

  constructor() { }

  ngOnInit() {
  }

}
```

Imports Input from the Angular core

Defines content as a class member that accepts an input of any type

2.Building a modular app using multiple nested components

2. Creating and using reusable subcomponents

Accepting And Displaying Incoming Data In A Component

Listing 9.12 Putting the data bindings in page-header.component.html

```
<div class="row banner">
  <div class="col-12">
    <h1>{{ content.title }}
      <small>{{ content.strapline }}</small>
    </h1>
  </div>
</div>
```

2.Building a modular app using multiple nested components

2. Creating and using reusable subcomponents

Creating The Sidebar Component

First, generate the component:

```
$ ng generate component sidebar
```

Second, grab the sidebar HTML from homepage.component.html, and paste it into sidebar.component.html. When you do, replace the text content with a binding to content:

```
<div class="col-12 col-md-4">
  <p class="lead">{{ content }}</p>
</div>
```

2.Building a modular app using multiple nested components

2. Creating and using reusable subcomponents

Creating The Sidebar Component

Third, allow the sidebar component to receive data by importing `Input` from Angular core and defining the `content` property—of type `string`—with the `@Input` decorator:

```
import { Component, OnInit, Input } from '@angular/core';

@Component({
  selector: 'app-sidebar',
  templateUrl: './sidebar.component.html',
  styleUrls: ['./sidebar.component.css']
})
export class SidebarComponent implements OnInit {

  @Input() content: string;

  constructor() { }

  ngOnInit() {
  }

}
```

2.Building a modular app using multiple nested components

2. Creating and using reusable subcomponents

Creating The Sidebar Component

Fourth, update the pageContent member in homepage.component.ts to contain the sidebar data:

```
public pageContent = {
  header : {
    title : 'Loc8r',
    strapline : 'Find places to work with wifi near you!'
  },
  sidebar : 'Looking for wifi and a seat? Loc8r helps you find places
  ➥to work when out and about. Perhaps with coffee, cake or a pint?
  ➥Let Loc8r help you find the place you\'re looking for.'
};
```

2.Building a modular app using multiple nested components

2. Creating and using reusable subcomponents

Creating The Sidebar Component

Fifth, update the homepage.component.html to use the new sidebar component, pass the data through as content:

```
<app-page-header [content]="pageContent.header"></app-page-header>
<div class="row">
  <div class="col-12 col-md-8">
    <div class="error"></div>
    <app-home-list>Loading...</app-home-list>
  </div>
  <app-sidebar [content]="pageContent.sidebar"></app-sidebar>
</div>
```

2.Building a modular app using multiple nested components

2. Creating and using reusable subcomponents

Creating The Sidebar Component

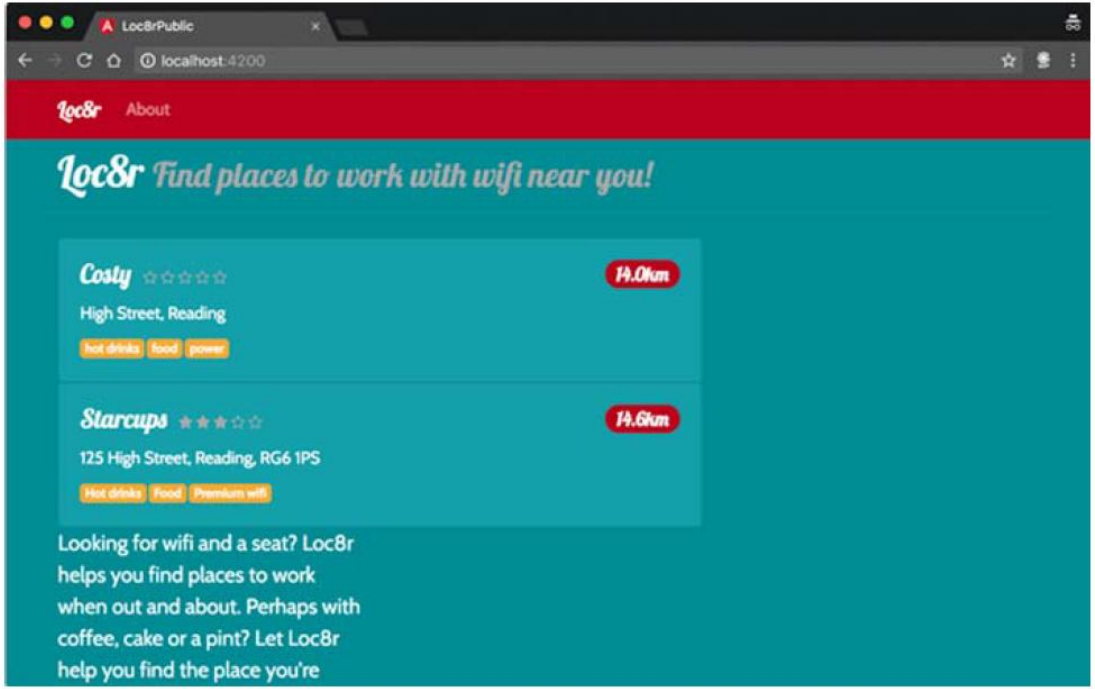


Figure 9.8 The new sidebar component is in and working, but it's below the main content instead of where it should be.

3. Adding geolocation to find places near you

To get geolocation working, you'll need to do the following things:

- Add a call to the HTML5 location API to your Angular application.
- Query the Express API if location details are available.
- Pass the coordinates to your Angular data service, removing the hardcoded location.
- Output messages along the way so the user knows what's going on.

Starting at the top, you'll add the geolocation JavaScript function by creating a new service.

3. Adding geolocation to find places near you

1. Creating an Angular geolocation service

To create the skeleton of the geolocation service, run the following in terminal from app_public:

```
$ ng generate service geolocation
```

Listing 9.14 Creating a `geolocation` service using a callback to get current position

```
import { Injectable } from '@angular/core';

@Injectable({
  providedIn: 'root'
})
export class GeolocationService {

  constructor() { }

  public getPosition(cbSuccess, cbError, cbNoGeo): void {
    if (navigator.geolocation) {
      navigator.geolocation.getCurrentPosition(cbSuccess, cbError);
    } else {
      cbNoGeo();
    }
  }
}
```

Defines a public member called getPosition that accepts three callback functions for success, error, and not supported

If geolocation isn't supported, invokes the not supported callback

If geolocation is supported, calls the native method, passing through success and error callbacks

3. Adding geolocation to find places near you

2. Adding the geolocation service to the application

To use your new geolocation service, you need to import it into the `home-list` component, as you did for your data service. You need to do the following:

- Import the service into the component.
- Add the service to the providers in the decorator.
- Add the service to the class constructor.

The following listing highlights in bold the additions you need to make to the `home-list` component definition to import and register the geolocation service.

3. Adding geolocation to find places near you

2. Adding the geolocation service to the application

Listing 9.15 Updating home-list.component.ts to bring in the geolocation service

```
import { Component, OnInit } from '@angular/core';
import { Loc8rDataService } from '../loc8r-data.service';
import { GeolocationService } from '../geolocation.service';

export class Location {
  _id: string;
  name: string;
  distance: number;
  address: string;
  rating: number;
  facilities: string[];
}

@Component({
  selector: 'app-home-list',
  templateUrl: './home-list.component.html',
  styleUrls: ['./home-list.component.css']
})
export class HomeListComponent implements OnInit {

  constructor(
    private loc8rDataService: Loc8rDataService,
    private geolocationService: GeolocationService
   ) { }
```

Imports the geolocation service

Passes the service into the class constructor

3. Adding geolocation to find places near you

3. Using the geolocation service from the home-list component

Creating The Geolocation Callback Functions

Inside the component, create three new private members, one for each of the possible geolocation outcomes:

- Successful geolocation attempt
- Unsuccessful geolocation attempt
- Geolocation not supported

3. Adding geolocation to find places near you

3. Using the geolocation service from the home-list component

Creating The Geolocation Callback Functions

Listing 9.16 Setting up the geolocation callback functions in home-list.component.ts

```
export class HomeListComponent implements OnInit {

  constructor(
    private loc8rDataService: Loc8rDataService,
    private geolocationService: GeolocationService
  ) { }

  public locations: Location[];

  public message: string;

  private getLocations(position: any): void {
    this.message = 'Searching for nearby places';
    this.loc8rDataService
      .getLocations()
      .then(foundLocations => {
        this.message = foundLocations.length > 0 ? '' :
        ➥'No locations found';
        this.locations = foundLocations;
      });
  }
```

Defines the message property of type string

Sets some messages inside the existing getLocations member

3. Adding geolocation to find places near you

3. Using the geolocation service from the home-list component

Creating The Geolocation Callback Functions

The function to run if geolocation isn't supported by browser

The function to run if geolocation is supported but not successful

```
  private showError(error: any): void {
    this.message = error.message;
  };

  private noGeo(): void {
    this.message = 'Geolocation not supported by this browser.';
  };

  ngOnInit() {
    this.getLocations();
  }

}
```

3. Adding geolocation to find places near you

3. Using the geolocation service from the home-list component

Calling The Geolocation Service

To call the getPosition method of your geolocation service, you'll need to create a new member in the home-list component and call it on init instead of calling the getLocations method directly.

```
private getPosition(): void {
  this.message = 'Getting your location...';
  this.geolocationService.getPosition(
    this.getLocations,
    this.showError,
    this.noGeo);
}
```

Then, you need to call this member when the component is initialized, instead of the getLocations method, so replace the call in ngOnInit to be this new member:

```
ngOnInit() {
  this.getPosition();
}
```

3. Adding geolocation to find places near you

3. Using the geolocation service from the home-list component

Calling The Geolocation Service

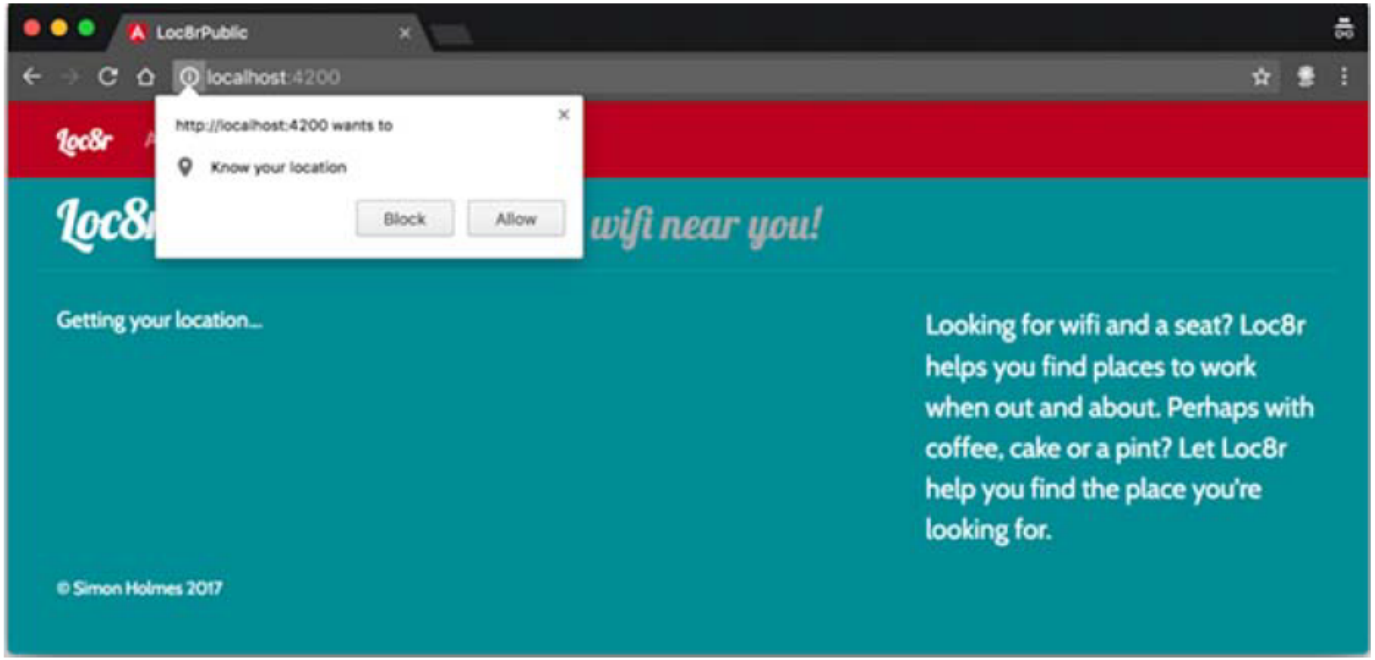


Figure 9.10 A successful call to your geolocation service is marked by a browser request to know your location.

3. Adding geolocation to find places near you

3. Using the geolocation service from the home-list component

Working With This In Callbacks Across Components And Services

Listing 9.17 Binding `this` to geolocation callback functions in home-list.component.ts

```
private getPosition(): void {
  this.message = 'Getting your location...';
  this.geolocationService.getPosition(
    this.getLocations.bind(this),
    this.showError.bind(this),
    this.noGeo.bind(this)
  );
}
```

3. Adding geolocation to find places near you

3. Using the geolocation service from the home-list component

Using The Geolocation Coordinates To Query The API

Listing 9.18 Updating home-list.component.ts to use the geolocation position

```
private getLocations(position: any): void {
  this.message = 'Searching for nearby places';
  const lat: number = position.coords.latitude;
  const lng: number = position.coords.longitude;
  this.loc8rDataService
    .getLocations(lat, lng)
    .then(foundLocations => {
      this.message = foundLocations.length > 0 ? '' : 'No locations found';
      this.locations = foundLocations;
    });
}
```

Accepts the position as a parameter

Extracts the latitude and longitude coordinates from the position

Passes the coordinates to the data service call

3. Adding geolocation to find places near you

3. Using the geolocation service from the home-list component

Using The Geolocation Coordinates To Query The API

Listing 9.19 Updating loc8r-data.service.ts to use the geolocation coordinates

```
public getLocations(lat: number, lng: number): Promise<Location[]> {
  const maxDistance: number = 20000;
  const url: string = `${this.apiBaseUrl}/locations?lng=${lng}&lat=${lat}&
  ➥maxDistance=${maxDistance}`;
  return this.http
    .get(url)
    .toPromise()
    .then(response => response.json() as Location[])
    .catch(this.handleError);
}
```

Accepts lat and lng parameters of type number

Deletes the hardcoded values you had for lat and lng before

3. Adding geolocation to find places near you

3. Using the geolocation service from the home-list component

Using The Geolocation Coordinates To Query The API

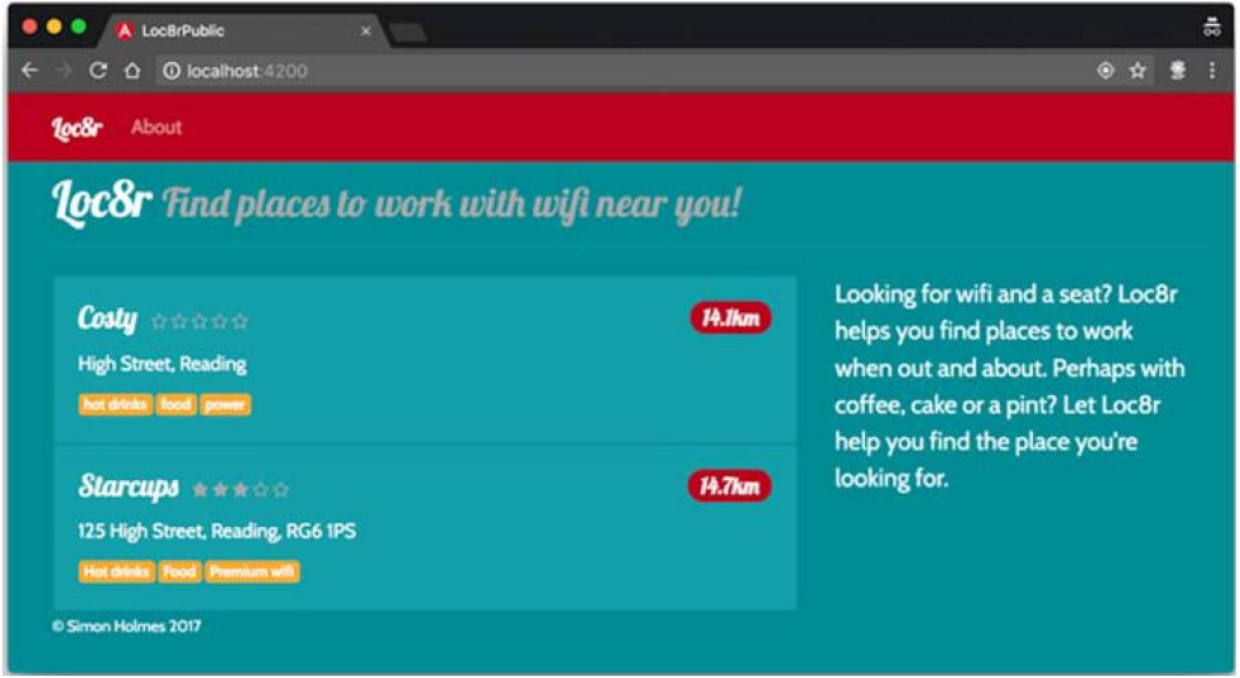


Figure 9.12 The Loc8r homepage as an Angular app, using geolocation to find places nearby from your own API

4. Safely binding HTML content

1. Adding the About page content to the app

Listing 9.20 Creating the Angular controller for the About page

```
export class AboutComponent implements OnInit {

  constructor() { }

  ngOnInit() {
  }

  public pageContent = {
    header : {
      title : 'About Loc8r',
      strapline : ''
    },
    content : 'Loc8r was created to help people find places to sit
    ➥down and get a bit of work done.\n\nLorem ipsum dolor sit
    ➥amet, consectetur adipiscing elit.'
  };
}
```

4. Safely binding HTML content

2. Creating a pipe to transform the line breaks

Creating An htmlLineBreaks Pipe

the following command in terminal to generate the files and register the pipe with the application:

$ ng generate pipe html-line-breaks

The pipe itself is fairly straightforward. It needs to accept incoming text as a string value. Replace each \n with a
, and then return a string value. Update the main content of html-line-breaks.html to look like the following snippet:

```
export class HtmlLineBreaksPipe implements PipeTransform {

  transform(text: string): string {
    return text.replace(/\n/g, '<br/>');
  }

}
```

4. Safely binding HTML content

2. Creating a pipe to transform the line breaks

Applying The Pipe To The Binding

Applying a pipe to a binding is simple; you've already done it a few times. In the HTML, add the pipe character (|) after the data object being bound, and follow it with the name of the filter like this:

```
<div class="col-12 col-lg-8">{{ pageContent.content | htmlLineBreaks }}</div>
```

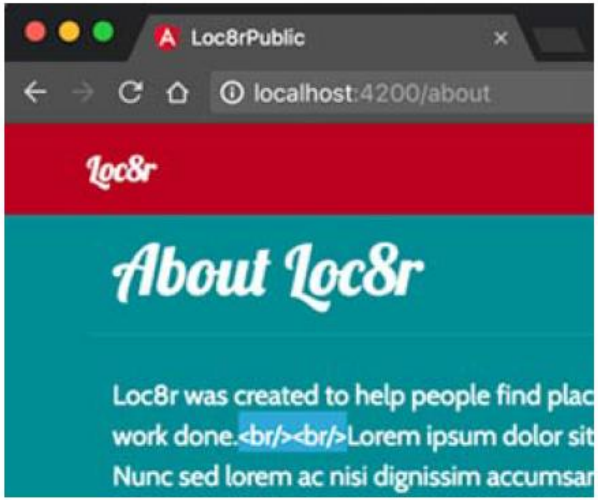


Figure 9.14 The
 tags being inserted with your filter are being rendered as text rather than HTML tags.

4. Safely binding HTML content

3. Safely binding HTML by using a property binding

Property bindings are denoted by wrapping square brackets around them and then passing the value. You can remove the content binding in about.component .html and use a property binding:

```
<div class="col-12 col-lg-8" [innerHTML]="pageContent.content |
     htmlLineBreaks"></div>
```

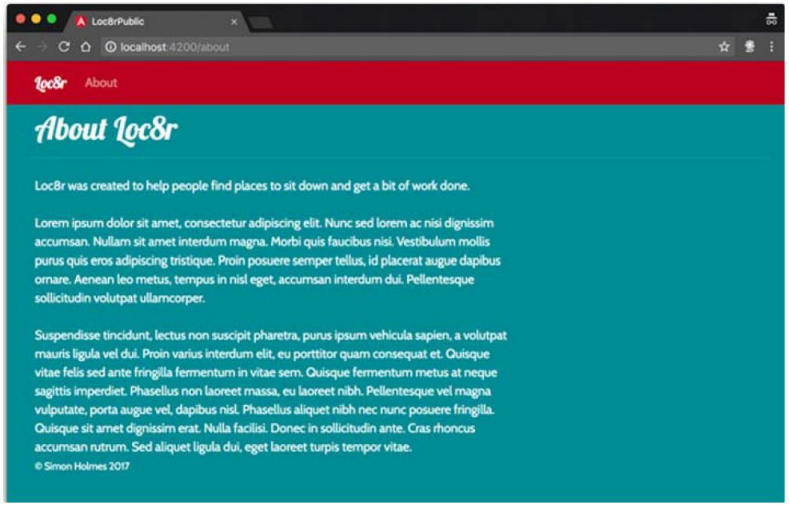


Figure 9.15 Using the `htmlLineBreaks` pipe in conjunction with the property binding, you now see the line breaks rendering as intended.

5. Challenge

This new component should

- Accept an incoming number value (the rating)
- Display the correct number of solid stars based on the rating
- Be reusable many times on a single page

As a clue, your elements should look something like this:

```
<app-rating-stars [rating]="location.rating"></app-rating-stars>
```

Thank You