COM661 Full Stack Strategies and Development

FE13. Additional URLs and Routing

Aims

- To introduce another component and route
- To extend the existing Data Service
- To introduce the Angular RouterModule
- To support multiple URLs in a single app via the RouterModule
- To define a Navigation Component
- To explore the Bootstrap navbar classes
- To integrate the navbar into the existing application

Table of Contents

13.1 Adding a New Component

Although one of the main principles of Angular is that apps are single page in nature, this essentially means that the browser is never completely re-loaded. Even though the application supports multiple URLs to provide different views of the information being delivered, this means that different chunks of HTML, CSS and TypeScript are loaded into the existing page to reflect the currently active Component. In this section, we will create a plain page to use as the homepage for our application and then see how to specify different URLs to navigate between it and the previously implemented page that displays details of businesses.

13.1.1 A Home Component

A static HTML page is easily generated by creating TypeScript, HTML and (optional) CSS files for a new Component, but by providing only the minimal TypeScript definition and having all content served by the HTML template. Create new files in the **src/app** folder for ***home.component.ts***, ***home.component.html*** and ***home.component.css*** and provide minimal content as shown below. Note that the ***home.component.ts*** file is most easily created by copying the existing ***app.component.ts*** file and deleting/modifying the appropriate elements. We will leave ***home.component.css*** empty (for now).

File: bizFE/src/app/home.component.ts

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

@Component({
  selector: 'home',
  standalone: true,
  imports: [RouterOutlet],
  templateUrl: './home.component.html',
  styleUrl: './home.component.css'
})
export class HomeComponent { }
```

File: bizFE/src/app/home.component.html

```
<h1>
    Biz Directory
</h1>
```

File: bizFE/src/app/home.component.css

```
<!-- file is empty -->
```

Do it now!	Create the files for the new Home Component as shown above.

13.1.2 Routing in an Angular Application

Angular provides a very useful Router module that manages multiple URLs within an application. In Angular, routing refers to the showing or hiding of portions of the display that correspond to Components, rather than fetching an entirely new page from the server. As users perform application tasks, so they move between the different views defined by the components of the application.

We can add the routes for the two Components in our current application by setting the **`HomeComponent`** as the default route (**/**) while the **`BusinessesComponent`** will be accessed by the URL **/businesses**. To achieve this, we modify the content of ***app.routes.ts***, importing the Components to which we want to define routes ,and specifying each route as a path (without a leading '**/**') and the component to be loaded if that path is requested. The import and the definition of the routes is illustrated in the following code box.

File: bizFE/src/app/app.routes.ts

```
import { Routes } from '@angular/router';
import { HomeComponent } from './home.component';
import { BusinessesComponent } from './businesses.component';

export const routes: Routes = [
    {
        path: '',
        component: HomeComponent
    },
    {
        path: 'businesses',
        component: BusinessesComponent
    }
];
```

Once the routes are defined, we need to specify when the output from the matching component should be injected into the application. Previously, we specified that the

`BusinessesComponent` output should be injected into ***app.component.html*** by using the **`<businesses></businesses>`** tag. We want to replace this with an instruction to instead display the output from the currently selected Component instead. First, we clear the current content of ***app.component.html*** as follows

File: bizFE/src/app/app.component.html

```
<!-- empty file -->
```

and replace it with the tag **`<router-outlet></router-outlet>`** to determine that the output of that Component that matches the currently matching path in the router should be shown.

File: bizFE/src/app/app.component.html

```
<router-outlet></router-outlet>
```

Do it now!	Add the routes specifications to ***app.routes.ts*** and make the change to ***app.component.html*** as shown above. Now run the application and confirm that the URL http://localhost:4200/ displays the new **`HomeComponent`** while http://localhost:4200/businesses displays the **`BusinessesComponent`** as shown in Figure 13.1 and Figure 13.2, below.

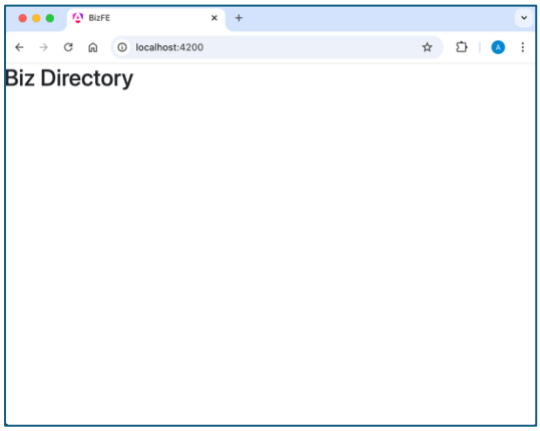

Figure 13.1 Home Page
http://localhost:4200/

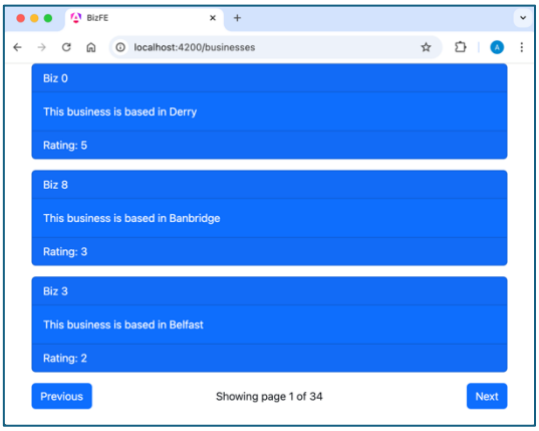

Figure 13.2 Businesses Directory
http://localhost:4200/businesses

13.2 Viewing a Single Data Item

Currently, the **`BusinessesComponent`** provides navigation through the entire collection of businesses data, but we want to add functionality such that when we click on a business entry from the main list, information about that business alone is displayed on the page. Initially, we will display the same information as is shown in the main list, but this could be used to display more complete information or allow the user to interact with the business in some way.

13.2.1 Creating the New Component

The first task is to add a new Component to our application to display a single business, where the business to be displayed is identified by its unique ID that is passed as a query string parameter.

The definition of a Component to display a single business is shown in the following code box. Note how we include the **`DataService`** in the **`import`**s and **`providers`** sections as we will want to retrieve details of the single business from the same Service that provides details on the collection of businesses, and also how we specify that the **`<business></business>`** tag is to be used as the selector for a single business.

File: bizFE/src/app/business.component.ts

```
import { Component } from '@angular/core';
import { RouterOutlet } from '@angular/router';
import { DataService } from './data.service';

@Component({
  selector: 'business',
  standalone: true,
  imports: [RouterOutlet],
  providers: [DataService],
  templateUrl: './business.component.html',
  styleUrl: './business.component.css'
})
export class BusinessComponent {}
```

Do it now!	Add the new file ***src/app/business.component.ts*** as described in the code box above. The easiest approach is to copy the code from the existing ***app.component.ts*** and make the necessary modifications.

13.2.2 Extending the Data Service

Once the new Component TypeScript definition is in place, the next step is to add functionality to the **`DataService`** to return a list containing the single business where the **`id`** field matches that passed as a parameter.

To achieve this, we add a method to the **`DataService`** that accepts an ID value as a parameter and iterates across the **`jsonData`** collection, returning that element with a matching ID. If we examine the format of the JSON dataset (as shown in Figure 13.3, below, we can see that the unique identifier is in the **`['_id']['$oid']`** property, so we use this as the comparator in the test for a matching business

Figure 13.3 Format of ID Field in the Dataset

The source of the new **`getBusiness()`** method in the **`DataService`** is shown in the code box below. Note how we continue to return the business data as a list, even though it is now a list of a single element. This will make the transition easier when we retrieve data from the live API later.

File: bizFE/src/app/data.service.ts

```
...

export class DataService {
    ...

    getBusiness(id: any) {
        let dataToReturn: any[] = [];
        jsonData.forEach( function(business) {
            if (business['_id']['$oid'] == id) {
                dataToReturn.push( business );
            }
        })
        return dataToReturn;
    }
}
```

Do it now!	Add the definition of the **`getBusiness()`** method to the **`DataService`** TypeScript file as shown above.

13.2.3 Adding the New Route

Now that the **`DataService`** functionality is in place, we can set up a route to the new **`BusinessComponent`** by importing the Component into ***app.routes.ts*** and specifying the route to an individual business. Note how the variable part of the URL (i.e. the business ID) is prefixed by a colon, giving the path **businesses/:id**.

File: bizFE/src/app/app.routes.ts

```
import { Routes } from '@angular/router';
import { HomeComponent } from './home.component';
import { BusinessesComponent } from './businesses.component';
import { BusinessComponent } from './business.component';

export const routes: Routes = [
    {
        path: '',
        component: HomeComponent
    },
    {
        path: 'businesses',
        component: BusinessesComponent
    },
    {
        path: 'businesses/:id',
        component: BusinessComponent
    }

];
```

Do it now!	Modify ***src/app/app.routes.ts*** to add the new route to the **`BusinessComponent`** as shown above. Consider how any new component can be added to the routes collection in this way.

13.2.4 Providing the Functionality

Now that the **`BusinessComponent`** has been created, the **`DataService`** has been extended, and the route is in place, we can implement the functionality of the **`BusinessComponent`** that retrieves the ID value from the URL and makes the request to the **`DataService`**.

As for the **`BusinessesComponent`**, we provide an implementation of the **`ngOnInit()`** method which makes a call to the **`DataService`** to retrieve the business information. Here, we call the **`DataService`** **`getBusiness()`** method, passing the ID defined in the route (as **businesses/:id**) as a parameter. The variable parameter is retrieved from the **`route.snapshot.param`** object by passing the query string parameter name as a parameter to the **`get()`** method.

File: bizFE/src/app/business.component.ts

```
...

export class BusinessComponent {
    business_list: any;

    ...

    ngOnInit() {
        this.business_list =
             this.dataService.getBusiness(
                      this.route.snapshot.paramMap.get('id'));
    }
}
```

However, access to **`paramMap`** requires that the **`ActivatedRoute`** Component is imported into the **`BusinessComponent`** and is injected into the Component by including it in the parameter list for the Component constructor.

File: bizFE/src/app/business.component.ts

```
...

import { RouterOutlet, ActivatedRoute } from '@angular/router';

...

export class BusinessComponent {
    ...

    constructor( public dataService: DataService,
                 private route: ActivatedRoute) {}

    ...
}
```

The complete modifications for the **`BusinessComponent`** Typescript file are presented in the code box below.

File: bizFE/src/app/business.component.ts

```
...

import { RouterOutlet, ActivatedRoute } from '@angular/router';

...

export class BusinessComponent {
    business_list: any;

    constructor( public dataService: DataService,
                 private route: ActivatedRoute) {}

    ngOnInit() {
        this.business_list =
             this.dataService.getBusiness(
                      this.route.snapshot.paramMap.get('id'));
    }
}
```

Do it now!	Complete the specification of the **`BusinessComponent`** TypeScript file as illustrated in the code box above.

13.2.5 Specifying the Front-End of the New Component

Finally, we can specify the **`BusinessComponent`** HTML template. This code is exactly as found in the **`BusinessesComponent`**, except that we omit the Bootstrap row that defines the pagination buttons. Remember that the information returned from the **`DataService`** is still formatted as list, even though it is a list containing a single element.

File: bizFE/src/app/business.component.html

```
<div class="container">
    <div class="row">
        <div class="col-sm-12">
            @for(business of business_list;
                                track business.name) {
                <div class="card text-white bg-primary mb-3">
                    <div class="card-header">
                        {{ business.name }}
                    </div>
                    <div class="card-body">
                        This business is based in
                        {{ business.town }}
                    </div>
                    <div class="card-footer">
                        Rating: {{ business.rating }}
                    </div>
                </div>
            }
        </div> <!-- col -->
    </div>    <!-- row -->
</div>      <!-- container -->}
```

We can also create a Component stylesheet file, though we choose at this stage to leave it empty since all style will be inherited from the parent application class.

File: bizFE/src/app/business.component.css

```
<!-- file is empty -->
```

Do it now!	Launch the application with **`ng serve`** and show that the **`BusinessComponent`** is working by presenting the URL http://localhost:4200/businesses/<id> to the browser, where <id> is the **`$oid`** property of one of the businesses in the file ***assets/businesses.json***. Verify that you receive output such as that presented in Figure 13.4, below.

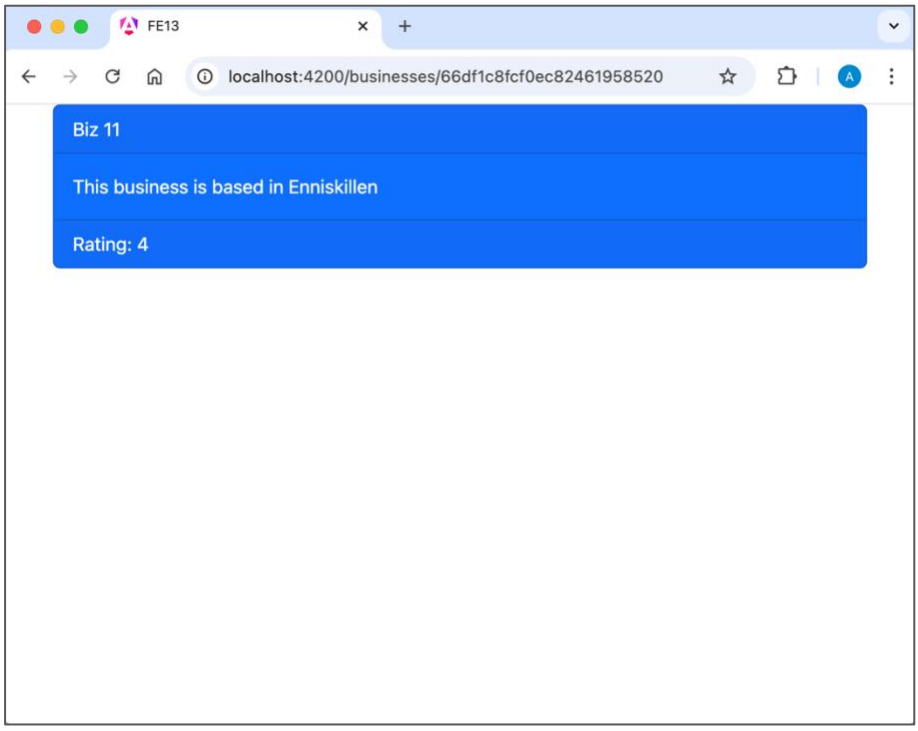

Figure 13.4 Display a Single Business

13.3 Connecting the Components

With the **`BusinessComponent`** that displays details of a single business running, we can now connect it to the **`BusinessesComponent`** that displays the collection of businesses. The way in which this is achieved is to make each element in the **`BusinessesComponent`** clickable, delivering the user to the page that displays information about the selected business.

To achieve this, we add a style rule to display the cursor as a pointer when it is positioned over the Bootstrap card that describes a business and make it clickable by binding a URL to its **`routerLink`** property. Note how the URL is specified as a list, where the component parts of the URL are individual list elements. Here the URL will be "**/businesses**" followed by the **`$oid`** field of the business **`_id`** property – hence the tatget URL will be (for example) **/businesses/1234a1234b1234c**.

File: bizFE/src/app/businesses.component.html

```
<div class="container">
    <div class="row">
        <div class="col-sm-12">
            @for(business of business_list;
                            track business.name) {
                <div class="card text-white bg-primary mb-3"
                     style = "cursor: pointer"
                     [routerLink] = "['/businesses',
                                     business._id.$oid]">
                    <div class="card-header">
                        {{ business.name }}
                    </div>

...
```

Then, to make the **`routerLink`** property available in the **`BusinessesComponent`**, we need to import **`RouterModule`** into the Component TypeScript file and add it to the Component's **`import`**s list.

File: bizFE/src/app/businesses.component.ts

```
import { Component } from '@angular/core';
import { RouterOutlet, RouterModule } from '@angular/router';
import { DataService } from './data.service';

@Component({
  selector: 'businesses',
  standalone: true,
  imports: [RouterOutlet, RouterModule],
  providers: [DataService],
  templateUrl: './businesses.component.html',
  styleUrl: './businesses.component.css'
})

...
```

Do it now!	Carefully follow the steps above to link the **`BusinessesComponent`** to the **`BusinessComponent`**. Verify that the businesses in the **`BusinessesComponent`** are now clickable and that clicking on one launches the **`BusinessComponent`** to display that business. You should obtain output such as that shown in Figures 13.5 and 13.6 below.

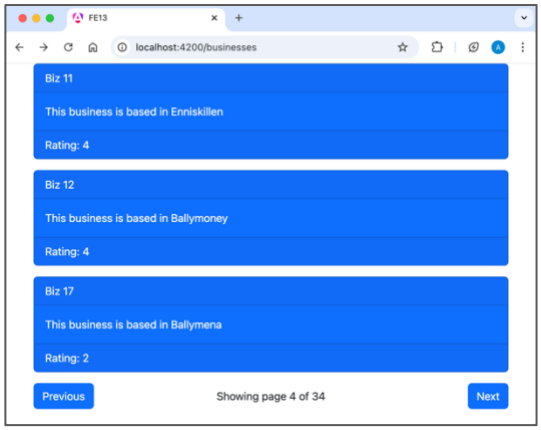

Figure 13.5 Clickable Page of Businesses

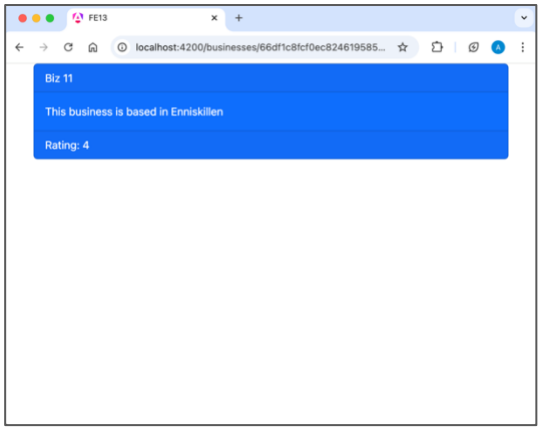

Figure 13.6 Individual Business Selected

13.4 Adding a Navigation Bar

Our Biz Directory application provides a retrieval interface to information on a collection of businesses stored in a MongoDB database. It also provides the facility for registered, logged-in users to add a review to the collection stored for each business.

In this development stage, we will implement a Bootstrap navigation bar as an Angular component, and have it automatically included on each page.

13.4.1 Create the Navigation Component

Creating the navigation component follows the now-familiar process or creating a TypeScript definition file and an HTML template. The TypeScript file can be most easily generated by copying the code from the ***app.component.ts*** file and making the necessary changes to result in the code box shown below.

File: bizFE/src/app/nav.component.ts

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

@Component({
  selector: 'navigation',
  standalone: true,
  imports: [RouterOutlet],
  templateUrl: './nav.component.html'
})

export class NavComponent { }
```

Now, we can create the **`NavComponent`** template as a Bootstrap **`navbar`** element. Most of the Bootstrap classes used are self-explanatory, but some will require additional clarification.

The navigation bar is created by specifying a **`<div>`** element with style classes that describe its physical characteristics, such as

`navbar-expand-sm` the navbar will collapse in a responsive manner when the browser window drops below the level of a "small device" (typically tablet resolution). Alternatives to **-sm** are **-xs** (small, smartphones), **-md** (medium, desktop resolution) and **-lg** (large, large desktop).

`bg-primary` the background of the navbar will be in the Bootstrap **primary** colour (blue). Other defined colour values are **`secondary`** (grey), **`success`** (green), **`danger`** (red) and **`warning`** (yellow) – although you can also define colours using the standard CSS **`background-color`** property.

`navbar-dark` characterises the navbar as **`dark`**, so prompting the use of a contrasting light text colour.

`fixed-top` fixes the **`navbar`** to the top of the browser window, allowing content to scroll beneath it.

Inside the navbar we have a text element defined as a **`<span>`** with class **`navbar-brand`**. This causes the "Biz Directory" text to be formatted as a heading within the navbar contents

Next, we have the menu items displayed as **<li>** items within a **<ul>** element. By applying the **navbar-nav** class to the **<ul>** element, we define it as a container for the navigation links, with each **<li>** specified as a **nav-item**. The actual link is an **<a>** anchor tag, specified as class **nav-link**. Note that we do not provide a **href** attribute for the **<a>**, instead using the Angular **routerLink** property to identify the route to be invoked when the link is clicked.

File: bizFE/src/app/nav.component.html

```
<div class="navbar navbar-expand-sm bg-primary
            navbar-dark fixed-top">
    <span class="navbar-brand">Biz Directory</span>
    <ul class="navbar-nav">
        <li class="nav-item">
            <a class="nav-link"
               [routerLink]="['/']">Home</a>
        </li>
        <li class="nav-item">
            <a class="nav-link"
               [routerLink]="['/businesses']">Businesses</a>
        </li>
    </ul>
</div>
```

Once again, we need to make **routerLink** available to the Component by importing **RouterModule** and adding it to the **imports** list.

File: bizFE/src/app/nav.component.ts

```
import { Component } from '@angular/core';
import { RouterOutlet, RouterModule } from '@angular/router';

@Component({
  selector: 'navigation',
  standalone: true,
  imports: [RouterOutlet, RouterModule],
  templateUrl: './nav.component.html'
})

export class NavComponent { }
```

Do it now!	Create the **`NavComponent`** by specifying the code shown above to the files ***nav.component.ts*** and ***nav.component.html***.

13.4.2 Connect the Navigation Component to the Application

With the new **`NavComponent`** created, we can now have it displayed on every page by adding it to the **`AppComponent`** template. First, we make it available to the **`AppComponent`** by **`import`**ing it and adding it to the list of Component **`imports`**.

File: bizFE/src/app/app.component.ts

```
import { Component } from '@angular/core';
import { RouterOutlet } from '@angular/router';
import { BusinessesComponent } from './businesses.component'
import { NavComponent } from './nav.component'

@Component({
  selector: 'app-root',
  standalone: true,
  imports: [RouterOutlet, BusinessesComponent, NavComponent],
  templateUrl: './app.component.html',
  styleUrl: './app.component.css'
})
export class AppComponent {
  title = 'bizFE';
}
```

Then, we place the **`<navigation>`** selector before the **`<router-outlet>`** within **`app.component.html`** to instruct the browser to render it before the content from the selected route, so effectively adding the navbar code at the top of each page.

File: bizFE/src/app/app.component.html

```
<navigation></navigation>
<router-outlet></router-outlet>
```

At this stage, we also add a **`margin-top`** property to the **`container`** **`<div>`** in the **`BusinessesComponent`**, **`BusinessComponent`** and **`HomeComponent`** template files to push the page content below the fixed navbar.

File: bizFE/src/app/businesses.component.html

File: bizFE/src/app/business.component.html

```
<div class="container" style="margin-top: 70px">

...
```

We also add similar spacing to the top of the content for the **`HomeComponent`**.

File: bizFE/src/app/home.component.html

```
<h1 style="margin-top:70px">
    Biz Directory
</h1>
```

The effect of these changes can be seen in Figure 13.7, which demonstrates the new navbar on the ***/businesses*** route.

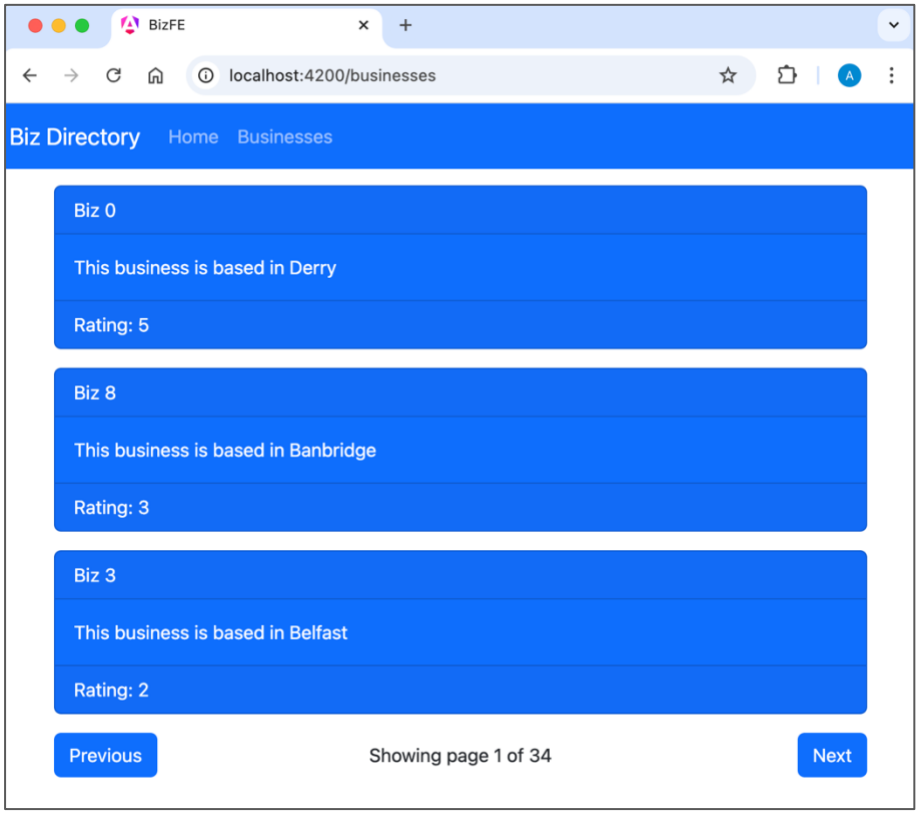

Figure 13.7 Navigation Bar Added

Do it now!	Connect the navigation bar to the application as shown above. Run the application and navigate through it, making sure that the navigation bar is present on all pages and that the links in the navigation bar work as expected.

13.5 Further Information

- https://angular.dev/guide/routing
  Angular Routing

- https://angular.dev/guide/routing/router-tutorial
  Using Angular Routes in a Single Page Application

- https://dev.to/codev206/how-to-set-up-routing-in-angular-to-create-single-page-applications-4ch2
  How to set up Routing in Angular

- https://www.typescriptlang.org/docs/handbook/variable-declarations.html
  Variable Declarations in TypeScript

- https://www.typescriptlang.org/docs/handbook/2/objects.html
  TypeScript Objects

- https://angular.dev/api/router/RouterLink
  Angular RouterLink class

- https://www.telerik.com/blogs/angular-basics-router-link-overview
  Angular RouterLink Overview

- https://css-tricks.com/almanac/properties/c/cursor/
  CSS Cursor Properties

- https://www.w3schools.com/html/html5_webstorage.asp
  HTML5 Web Storage API

- https://www.digitalocean.com/community/tutorials/how-to-use-the-javascript-developer-console
  How to use the JavaScript Developer Console

- https://getbootstrap.com/docs/5.3/components/navbar/
  The Bootstrap NavBar

- https://www.w3schools.com/bootstrap/bootstrap_navbar.asp
  Bootstrap Navigation Bar