COM661 Full Stack Strategies and Development

FE11. Introducing Angular

Aims

- To appreciate the purpose of Angular in full-stack architecture
- To understand the code structure of an Angular application
- To introduce the structure and role of an Angular component
- To demonstrate injection of data values into the web presentation
- To introduce Angular code blocks
- To install Bootstrap through the Node Package Manager
- To style the initial application using Bootstrap cards
- To appreciate the use of ther Node Package Manager in managing an Angular application (npm)

Table of Contents

11.1 Installation and First Use

Angular is a front-end framework for the development of modern, responsive and scalable single-page Web applications. In this section of the module, we will use Angular to build a front-end to the API for the Biz application that we created in section B of the module.

11.1.1 Angular and AngularJS

AngularJS was first developed at Google in 2009 as an MVC (Model, View, Controller) JavaScript-based framework for the rapid development of front-end web applications. It was publicly launched as Version 1.0 in 2012 and quickly grew in popularity as an alternative to jQuery for the development of complex interactions.

The release of Version 2 in 2014 was the result of a complete re-write of the infrastructure and significant syntax changes, resulting in incompatibility between versions 1 and 2. As Version 1 had already become the most popular JavaScript framework by 2011, the change was met with some resistance, but simplifications in code structure and performance improvements saw the updated framework (now known simply as Angular) maintain its popularity. **The current version of Angular is v20**, but it is important to recognize that versions v2 and v4–v20 (there was no v3!) are largely compatible with each other (although there were significant differences between v16 and v17). Most changes were concerned with improvements under the bonnet – rather than in syntax and code structure. To emphasise this, v1 (which maintains a large user base) is commonly known as **AngularJS**, while versions 2+ are known simply as **Angular**.

Note:	When searching online for answers to Angular queries, always be aware of the version of Angular to which the answer applies. Any articles that are aimed at AngularJS should be ignored, and it is best to include a specific version number in your search.

Note:	Angular is updated on a regular schedule, normally twice per year. The next update (to v21) is scheduled for the week of 17th November 2025 but if you are installing Angular after this date you should be sure to install v20 to maintain compatibility with this material.

One of the major changes between AngularJS and Angular is the introduction of TypeScript as a replacement for JavaScript. TypeScript was developed by Microsoft as a superset of

JavaScript, adding optional data typing to the language. All JavaScript code is valid TypeScript and, although we will use TypeScript in this section of the module, we will not formally cover it – but will point out significant features as we meet them.

11.1.2 Getting Started

Note:	Angular has been provided for you on the lab machines but if you need to install it on your own computer, you can do so (as long as npm has previously been installed) by issuing the command **C:\> npm install –g @angular/cli** **npm** (Node Package Manager) is a component of Node.js. If you do not already have Node.js available on your machine, you will need to install it first from https://nodejs.org/en/.

Note:	To install a specific version of Angular you can add the required version number after the command above as follows (e.g. for version 20): **C:\> npm install –g @angular/cli@20**

As a first example, we will create the first version of our Biz Directory Angular application called **bizFE** by the command

```
C:\Users\B00123456\Desktop> ng new bizFE
```

Note:	Given the volume of data that needs to be created/written to build an Angular application framework (typically over 350 MBytes), it is a good idea to create the application on the Desktop (on the lab machines) so that delay due to network traffic is eliminated. You can minimise the application volume (see later) and copy it to your personal drive when the session is complete.

This will take a little while to execute – even up to a couple of minutes on a high-powered computer – but we only need do it once for each application. As the command executes,

you will be asked a number of questions regarding the nature of the application. You should select the default value in each case by pressing the **<return>** key in response to each question.

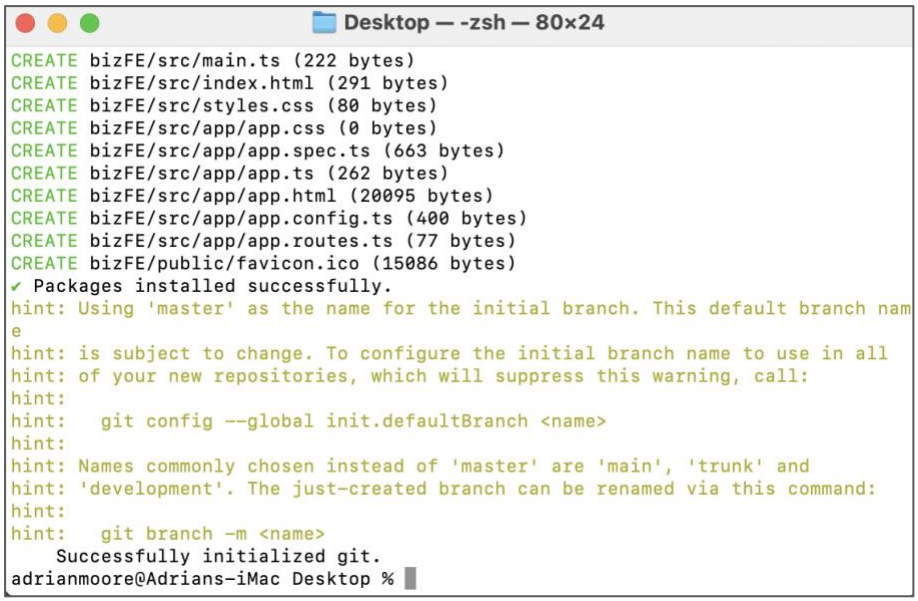

Figure 11.1 Creating a New Angular Application

Although we have not yet provided any code, we can test that the application has installed properly and that a full Angular application has been generated by navigating into the **biz** directory and launching it with the commands

```
C:\Users\B00123456\Desktop> cd bizFE
C:\Users\B00123456\Desktop\bizFE> ng serve
```

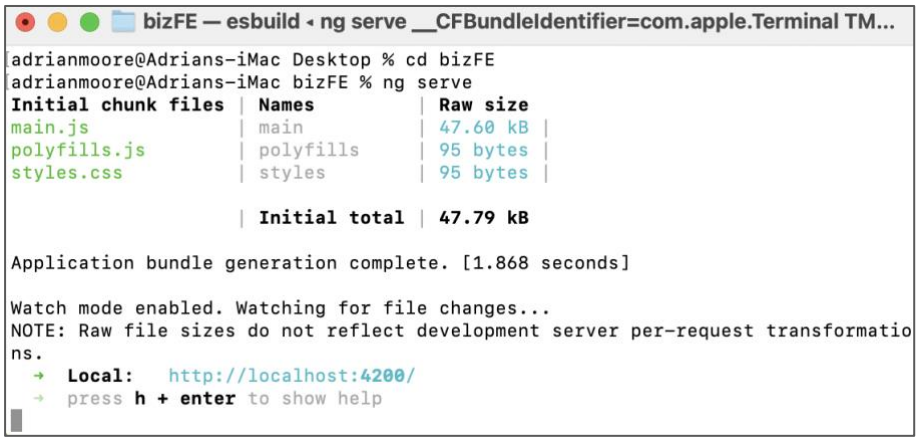

Figure 11.2 Running the Front-end Server

Now that the server is running, we can open a web browser and load the URL http://localhost:4200 to see the default Angular application homepage as shown below.

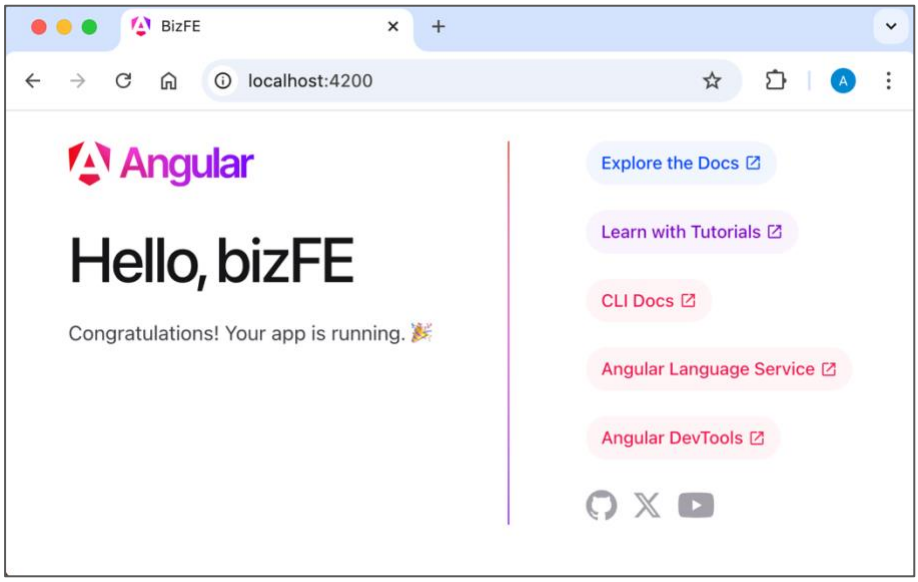

Figure 11.3 Default Angular Server Output

Do it now!	Create a default Angular application called **bizFE** by following the steps outlined above. Run the application and visit the URL http://localhost:4200 in a web browser to ensure that you are presented with the default Angular page shown in Figure 11.3 above.

Note:	Your default home page may look different depending on the version of Angular CLI that is installed. We will delete all of the code that specifies this page anyway, so any difference will not matter**.**

11.1.3 Basic Structure of an Angular App

The Angular CLI creates all the folders and files that will support our application development. All the files that we will edit (and add) will be in the **src** directory – the other directories and files are Angular system files to support the application's development and execution.

The base of the frontend application is the file ***index.html***, located in the **src** directory. If you examine the code in this file (shown in the code box below), you will see a standard HTML skeleton for a web application – enhanced by a curious **`<app-root></app-root>`** tag in the **`<body>`** section. This is an example of an Angular **Component** – the basic building block of an Angular application. Components enable us to inject content onto the page at a position of our choosing and it is worth exploring the code to illustrate how this default component generates the display shown in the browser

File: bizFE/src/index.html

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

A Controller is specified by a TypeScript file and an optional HTML template and CSS stylesheet. The files for components are defined in the ***app*** sub-directory of ***src***, so first look at the default component TypeScript file, ***app.ts***.

File: bizFE/src/app/app.ts

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

@Component({
  selector: 'app-root',
  imports: [RouterOutlet],
  templateUrl: './app.html',
  styleUrl: './app.css'
})
export class AppComponent {
  protected title = 'FE11';
}
```

The component TypeScript file consists of 3 sections

i) The **`import`** statements that enable the file to make use of the facilities of the Angular **`Component`** and **`RouterOutlet`** libraries

ii) The **Decorator** that specifies the selector (HTML tag) to be used to refer to the component (here, **`app-root`** – corresponding to the **`<app-root>`** tag used in

index.html) and the optional URLs for an HTML template and stylesheets to be used when rendering the component.

iii) Finally, the **class definition** that contains the logic for the component. Here, we set up a single variable called **`title`** with the value '**bizFE**'. Note that the variable is by default **`protected`**. This means that it will also be accessible in any subclasses of this class that we might define later.

To best understand the structure of the default Angular application, we will replace all the code in ***app.html*** with that provided below.

File: bizFE/src/app/app.html

```
<h1>
    Welcome to {{ title }}
</h1>
```

Note here how the component variable **`title`** defined in the class definition can be used within the template. As we see in the browser, the value of the variable is inserted into the **`<h1>`** tag by enclosing it in **`{{ }}`**. This is an example of data binding – a central concept of Angular, of which we will see (much) more later.

Do it now!	Modify the code of ***app.html*** as shown above and verify that you receive output such as that illustrated by Figure 11.4 below.

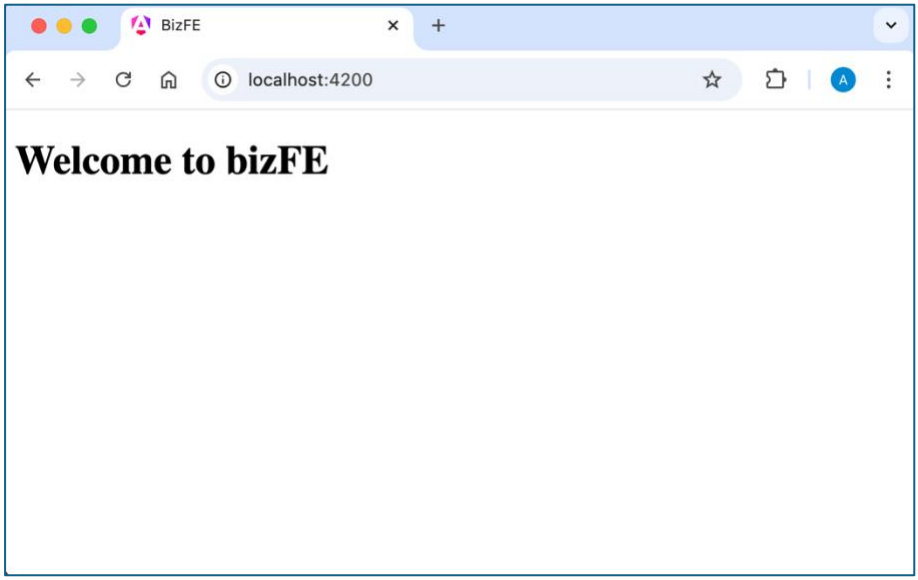

Figure 11.4 Modified Default Output

Try it now!	Develop your understanding of Angular code structure by attempting the following: i) Change the value of the title variable in the component TypeScript file and verify that the browser output changes accordingly. ii) Change the HTML template by adding some additional content of your own (but leave the **`<h1>`** element in place). iii) Add a style rule to the CSS template file to have the **`<h1>`** element displayed in red text on a yellow background. iv) Make the required changes so that the **`<app-root>`** component is called by the modified tag **`<app-title>`**. (Note: return it to **`<app-root>`** when you have shown that you can change it – we will refer to this element again later) v) Note how the changes take effect in the browser immediately when you save the source file – the application is automatically re-loaded after each change.

11.2 Specifying Custom Components

In this section, we will add a custom component to our front-end application. Eventually, this component will house our collection of businesses, but initially we will just display a simple heading.

11.2.1 Creating a Component

One of the characteristics of Angular applications is the large number of separate code files that are generated. To help manage the code, we will create a custom folder called ***components*** within ***src/app*** into which we will create our custom components.

The easiest way to do this in Visual Studio Code is to right click on the app folder in the project explorer on the left-hand side and select "***Open Integrated Terminal***" from the menu, as shown in Figure 11.5 below.

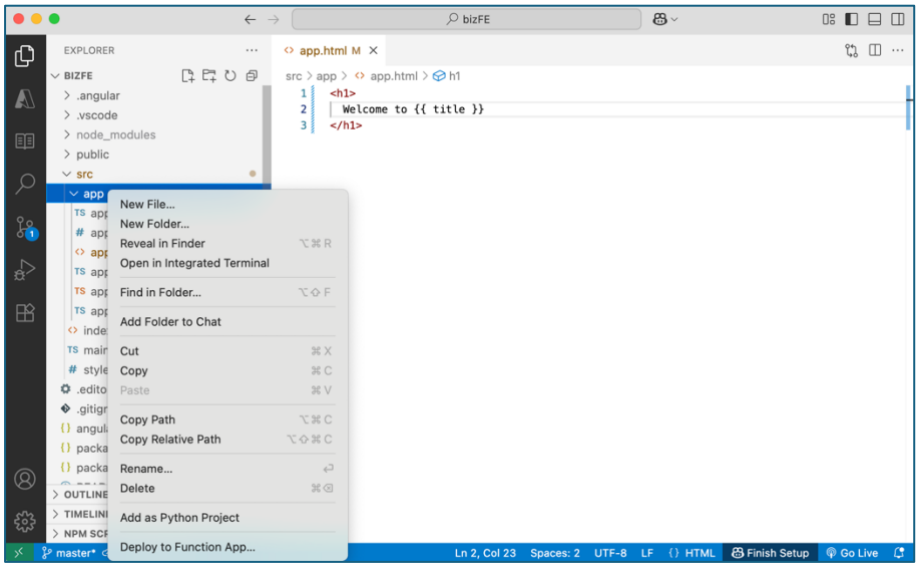

Figure 11.5 Open Integrated Terminal

Now we create our components folder by the command **`mkdir components`**, navigate into it by **`cd components`** and then finally create our new component called **`businesses`** by the command **`ng generate component businesses`**, as seen in Figure 11.6, below.

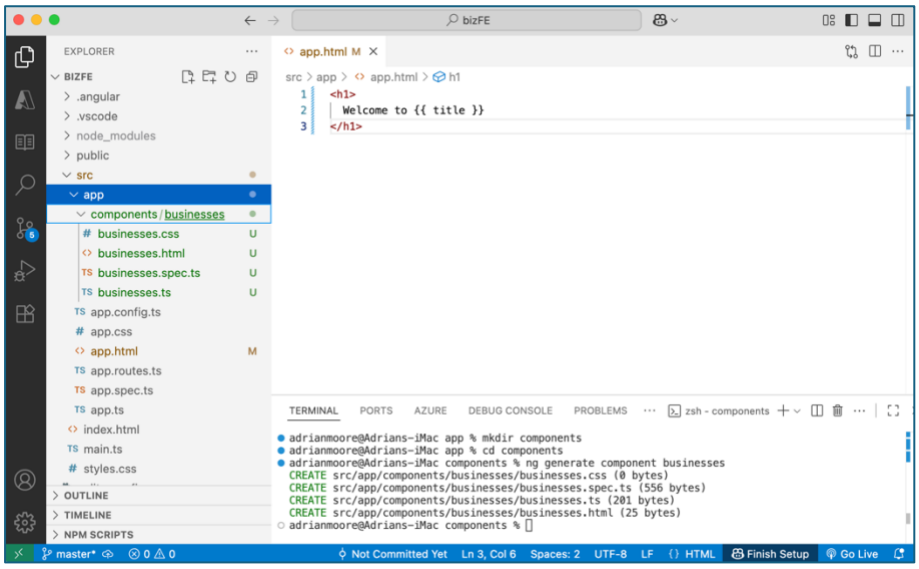

Figure 11.6 Generate the Businesses Component

This creates 4 files defining the new component as follows

- ***components/businesses/businesses.ts***,
- ***components/businesses/businesses.html***,
- ***components/businesses/businesses.css*** and
- ***components/businesses/businesses.spec.ts***

In a component definition, the ***.ts*** file specifies the logic, the ***.html*** and ***.css*** files define the on-screen presentation and the ***.ts.spec*** file is used in component testing (we will see the use of this later).

Now, we specify the content to be rendered by the component by replacing the default content in the HTML file by a simple **<h1>** element.

File: components/businesses/businesses.html

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

Checking the file ***businesses.ts*** reveals that the default component class structure has been defined for us as shown below.

File: components/businesses/businesses.ts

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

@Component({
  selector: 'app-businesses',
  imports: [],
  templateUrl: './businesses.html',
  styleUrl: './businesses.css'
})
export class Businesses { }
```

Next, we use the new selector **app-businesses** by adding an **<app-businesses></app-businesses>** tag to the ***app.html*** file to render an instance of our new **Businesses** Component.

File: bizFE/src/app/app.html

```
<h1>
     Welcome to {{ title }}
</h1>

<app-businesses></app-businesses>
```

If we now examine the browser, we can see that the application fails to display our new content but instead produces an error message in the Terminal window reporting that the **<app-businesses>** element is unknown, as shown in Figure 11.7 below.

Figure 11.7 Unregistered Component

11.2.2 Connecting a Component to the Application

There are two further things we need to do to register the new component with the application. First, we need to **`import`** the new **`Businesses`** component into ***app.ts***, as the component is rendered within its template. Then, we need to register the new component by adding it to the **`imports`** list.

File: bizFE/src/app/app.ts

```
import { Component } from '@angular/core';
import { RouterOutlet } from '@angular/router';
import { Businesses } from './components/businesses/businesses';

@Component({
  selector: 'app-root',
  imports: [RouterOutlet, Businesses],
  templateUrl: './app.html',
  styleUrl: './app.css'
})
export class App {
  protected title = 'bizFE';
}
```

Now, when we view the content in the browser, we can see that our new component has been properly rendered on the page, as shown in Figure 11.8 below.

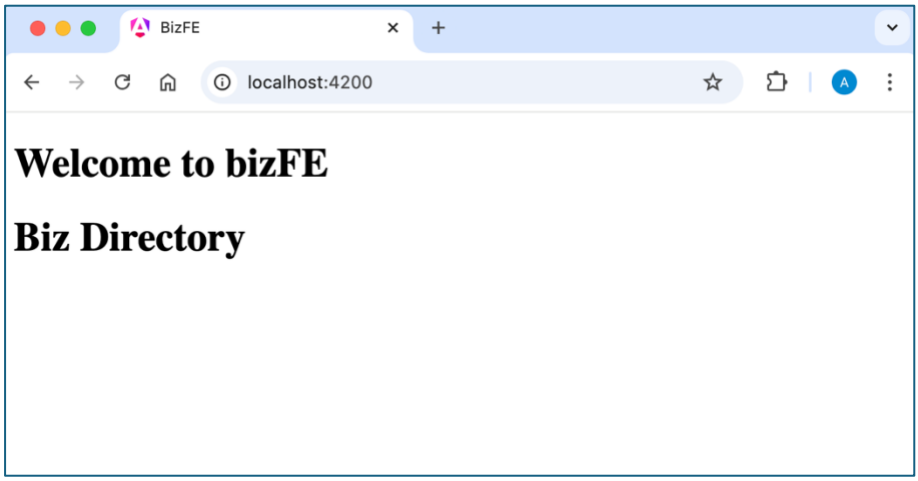

Figure 11.8 New Component Correctly Rendered

Do it now!	Register the **`Businesses`** component with the application as shown above. Verify that you receive output as illustrated in Figure 11.8.

11.3 Angular Data Processing

So far, although our new component is connected to the application, its impact is limited to the display of a simple, static HTML element. We will now enhance the functionality of the component by having it process JSON data that we specify in the controller's TypeScript file.

11.3.1 Manipulating JSON Data

In the **`Businesses`** class of the ***businesses.ts*** file, we define a variable **`business_list`** as a JSON object holding information about 3 business objects. For each business, we specify value for fields "*name*", "*town*" and "*rating*" (reflecting 3 of the fields in our data set from the Biz Directory API application).

File: components/businesses/businesses.ts

```
...

export class Businesses {

    business_list =  [
        {
            "name" : "Pizza Mountain",
            "town" : "Coleraine",
            "rating" : 5
        },
        {
            "name" : "Wine Lake",
            "town" : "Ballymoney",
            "rating" : 3
        },
        {
            "name" : "Sweet Desert",
            "town" : "Ballymena",
            "rating" : 4
        }
    ]

}
```

Now, we add code to the ***businesses.html*** file to display the size of the JSON structure (i.e. the number of elements it contains) and the **`name`** property of the first element.

File: components/businesses/businesses.html

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

<h2>
    There are {{ business_list.length }}
    businesses in the directory.
</h2>

<h2>
    The first business is
    {{ business_list[0].name }}
</h2>
```

Examining the browser contents reveals that the information from the JSON structure is now embedded into the code that specifies the web page as shown in Figure 11.9 below.

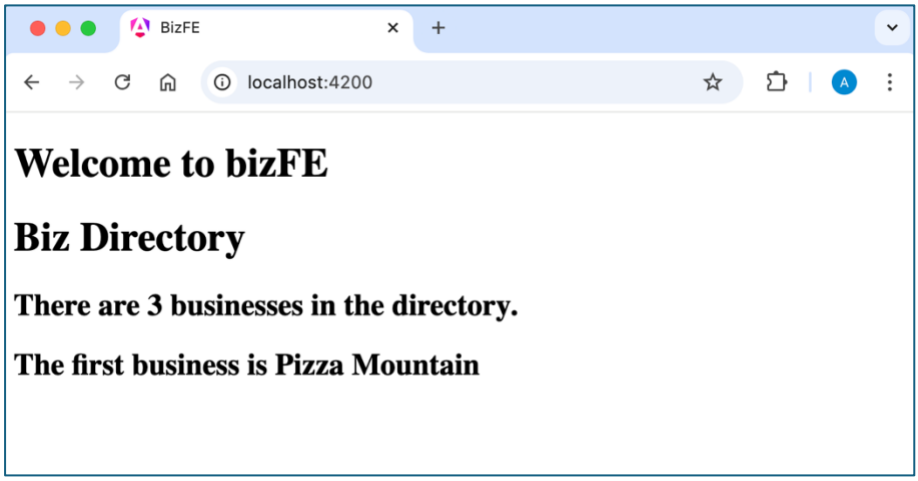

Figure 11.9 Injecting Data into the Web Page

Do it now!	Add the collection of data to ***businesses.ts*** and add the code shown above to ***businesses.html*** to report on the number of businesses in the collection and the name of the first business.

11.3.2 Iterating Across Data with the `@for` Block

Angular provides a number of very useful control flow blocks that enable us to introduce common programming constructs into our HTML specification. One of the most useful of these is the **`@for`** block, that provides a looping mechanism.

`@for` takes the form "***`variable`*** `of` ***`collection`***", by which the **`variable`** iterates across the **`collection`**, taking on each value in turn. To illustrate this, consider the code box below where the **`@for`** block is applied to a block of code that contains a paragraph that presents the *name*, *town* and *rating* properties of each element of **`business_list`** in turn. Note the requirement for the **`track`** element where one of the fields is nominated as the "key" which is then tracked by Angular as it iterates across the collection.

File: components/businesses/businesses.html

```
...

@for (business of business_list; track business.name) {
    <p>
        Name: {{ business.name }} <br>
        Town: {{ business.town }} <br>
        Rating: {{ business.rating }}
    </p>
}
```

The effect of this is to generate separate **<p>** elements for each business as shown in Figure 11.10 below.

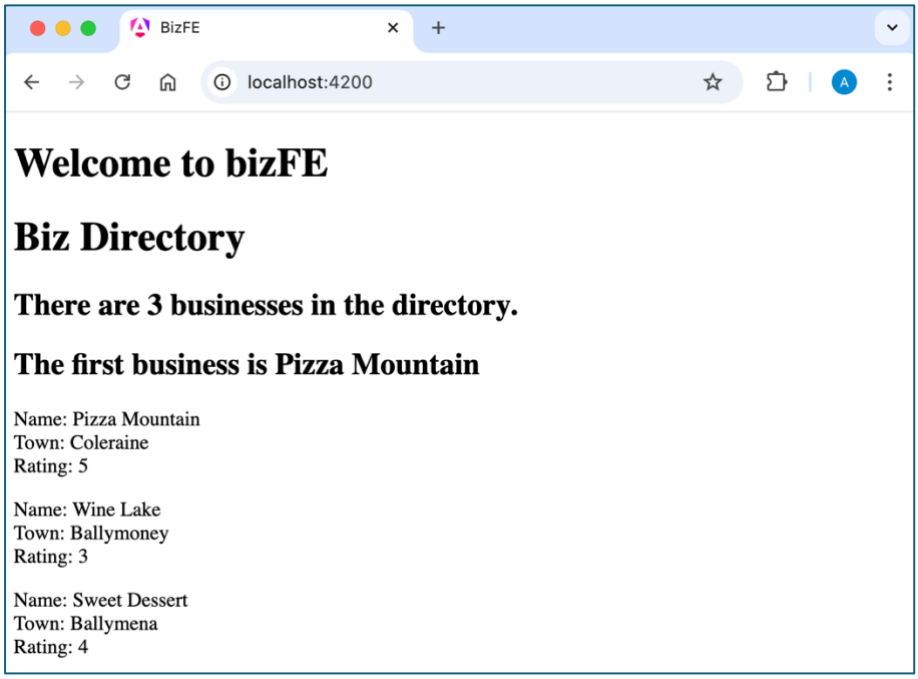

Figure 11.10 Using ***`@for()`*** *to traverse a dataset*

Do it now!	Verify the operation of the **`@for`** block by adding the code above to ***businesses.component.html*** and make sure that you obtain output such as that shown in Figure 11.10, above.

11.4 Using Bootstrap

As we build the front-end of the Biz application, we will style the interface using Bootstrap – a popular CSS framework for responsive front-end development. This will provide an easy

way of creating an attractive interface, as well as automatically providing dynamic adjustment for different devices and browser characteristics.

11.4.1 Installing Bootstrap

We could use Bootstrap by linking to a local copy or a CDN, but **npm** also provides it as a package that we can install into our application. To install Bootstrap and its dependent package jQuery, kill any currently running Angular server (using CTRL-C), navigate back to the **bizFE** directory and issue the following command to install the packages using the **Node Package Manager** (npm). (Note – the following command should be issued all on a single line.)

```
C:\Users\B00123456\Desktop\bizFE> npm install bootstrap
jquery --save
```

Now, we need to tell the Angular application that the Bootstrap CSS library is available by adding it to the list of global stylesheets. Open the file **/angular.json** in the code editor and locate the **`styles`** entry (Inside the `projects|FE11|architect|build|options` object). Now add the location of the file ***bootstrap.min.css*** to the styles entry as shown below.

File: bizFE/angular.json

```
...

   "styles": [
       "src/styles.css",
       "node_modules/bootstrap/dist/css/bootstrap.min.css"
   ],

...
```

We can check that Bootstrap has been installed by re-running the application (with **`ng serve`**), exploring the **node_modules** folder in our IDE and navigating to **node_modules/bootstrap/dist/css** to check that the Bootstrap files are now present.

Note:	You can also see visually that Bootstrap has been installed by simply observing the change in font and layout of your application contents in the browser as shown in Figure 11.11, below.

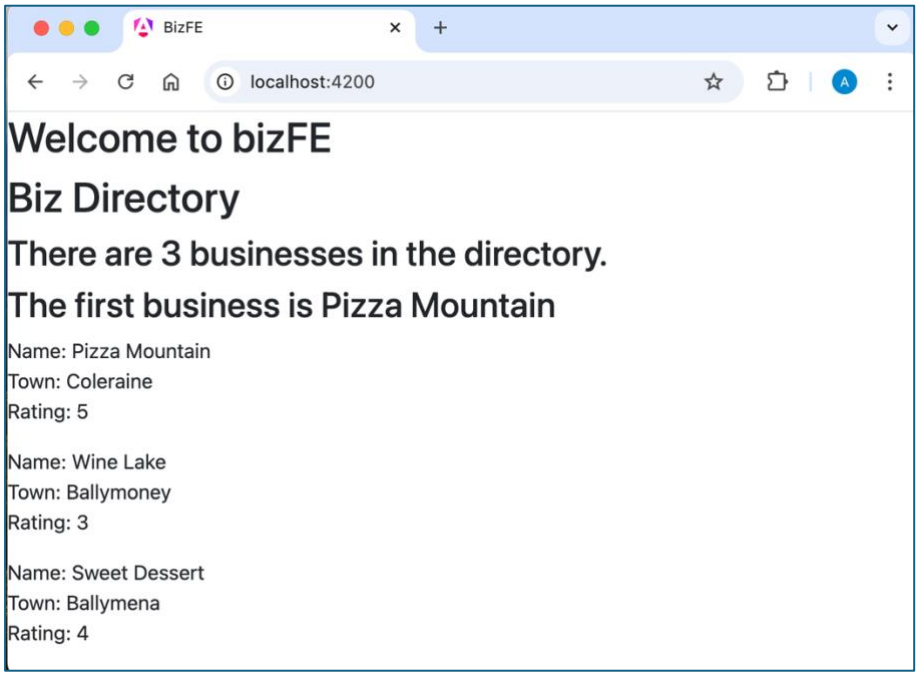

Figure 11.11 Verifying that Bootstrap is installed

Do it now!	Install Bootstrap and verify that it is available to the application by following the steps above.

11.4.2 Styling with Bootstrap

As a quick example of Bootstrap styling, we will present our JSON data as Bootstrap **`card`** elements. A Bootstrap card is a flexible and extensible content container. It includes options for headers and footers, a wide variety of content, contextual background colors, and powerful display options. The card is specified by applying style classes to **`<div>`** elements that define the card and its optional header, body and footer components as shown below.

```
<div class = "card">
     <div class = "card-header">
          <!-- the card header text -->
     </div>

     <div class = "card-body">
          <!-- the card body text -->
     </div>

     <div class = "card-footer">
          <!-- the card footer text -->
     </div>
</div>
```

The remaining CSS classes applied (e.g. **`text-white`**, **`bg-primary`**, etc.) are Bootstrap classes to set text and background colours. A full specification of Bootstrap cards can be seen at https://getbootstrap.com/docs/5.3/components/card/. The code box below shows the full revised specification for ***businesses.html***.

File: components/businesses/businesses.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 -->
```

Finally, we remove all other code from ***app. html*** so that the entire presentation consists of our own work in the **`businesses`** component.

File: bizFE/src/app/app.html

```
<app-businesses></app-businesses>
```

The effect of these changes can be seen in Figure 11.12, below.

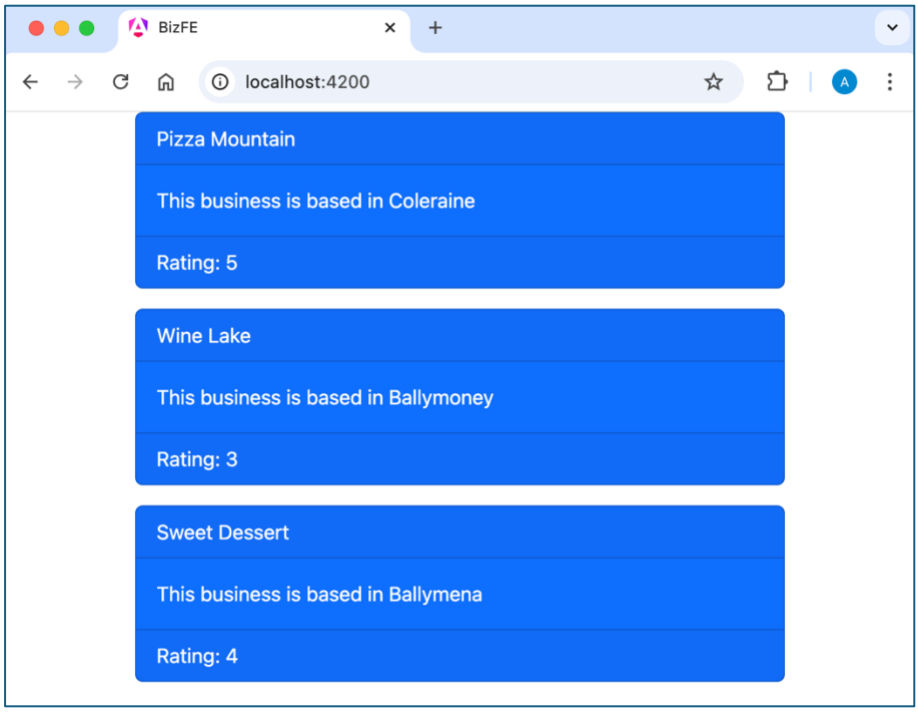

Figure 11.12 Using Bootstrap Cards

Do it now!	Apply Bootstrap styling to the application by following the instructions set out in this sub-section. Ensure that you obtain output as illustrated by Figure 11.12 above.

Try it now!	Add more businesses to the data set and experiment with Bootstrap and Cards to generate the most attractive layout with cards displayed 3 or 4 to a row.

11.5 npm and Package Management

In the previous section, we introduced the Node Package Manager (npm) as a tool for the installation of external components into our Angular application. npm is a **Node.js** tool that allows us to define and manage dependencies between code modules, libraries and packages. This helps promote the notion of re-usable code, where members of the Node.js community develop and share useful packages that others can include in their projects.

All npm-based applications contain a file, usually in the project root, called ***package.json***. This file holds various metadata relevant to the project and is used to give information to npm that allows it to identify the project as well as handle the project's dependencies. It can also contain other metadata such as a project description, the version of the project in a particular distribution, license information and configuration data.

Do it now!	Examine the file ***/package.json*** and see how it contains the specification of all of the components in our application. Note the entries for **bootstrap** and **jquery** that were added to ***package.json*** by our use of the **`npm install`** command in the previous section.

Note:	When using **`npm install`**, the **`--save`** flag is the command that tells npm to add entries for the components being added to *package.json*. Without specifying **`--save`**, the components would be added, but *package.json* would remain unchanged.

11.5.1 Understanding Version Numbers

npm packages are identified using the **Semantic Version Specification** (SemVer), which expresses software versions as three values identified as the **Major** version, the **Minor** version and the **Patch** version. If you examine the entry in ***package.json*** for Bootstrap, you will see that it is in the form

```
"bootstrap": "^5.3.8",
```

which denotes Major version **5**, Minor version **3** and Patch version 8. In the SemVer scheme, given a version number in the format Major.Minor.Patch, developers should increment the

- MAJOR version when a change results in previous versions of the API being incompatible
- MINOR version when new functionality is added in a backwards compatible manner (i.e. existing software will still work)
- PATCH version when changes are backwards-compatible bug fixes

In other words, a user of a package should always be able to safely upgrade to the latest Patch or Minor version without fear of incompatibility but upgrading a Major version (i.e. to Bootstrap 6.x.x) may cause existing functionality to break and should only be done after thorough testing.

The use of the **^** symbol in the ***package.json*** entry

```
"bootstrap": "^5.3.8",
```

instructs npm that although it may automatically download the latest Patch and Minor versions as they become available, it should not automatically upgrade to any new Major version that may be released. This should future proof our application against updates to any of the components that we use.

11.5.2 Backup and Restore of the Angular Application

One of the most useful features of ***package.json*** is that it allows us to store or transfer a greatly reduced subset of the application code by first deleting the ***node_modules*** folder that contains all of the supporting packages, leaving only the application structure and our own code files.

If you examine the ***bizFE*** folder for our current application, you will find that it occupies in the region of 300 Mbytes across approximately 25,000 files. However, if you then examine the ***node_modules*** folder, you will find that it accounts for almost all of this bulk. Since ***package.json*** is essentially a script that tells npm how to recreate ***node_modules***, we can therefore safely delete the entire ***node_modules*** folder.

When we want to restore the application, we simply use the command **`npm install`** from the application directory (the same directory where ***package.json*** is found) which opens ***package.json***, creates a ***node_modules*** folder and installs all of the components listed in ***package.json*** into it.

Do it now!	Stop the Angular server and delete the folder ***node_modules*** from the **bizFE** application. Then, from a Command prompt in the ***bizFE*** folder, issue the command **`npm install`** to re-create the application. Once installed, run it by the command **`ng serve`** and verify that everything works as previously.

11.5 Further Information

- https://angular.dev/
  Angular home page

- https://kinsta.com/knowledgebase/install-angular/
  How To Install Angular on Windows, macOS, and Linux

- https://angular.dev/tutorials/learn-angular
  Getting Started with Angular

- https://angular.dev/guide/components
  Angular – Introduction to Components

- https://www.typescriptlang.org/
  TypeScript home page

- https://nodejs.org/en/
  Node.Js home page

- https://nodesource.com/blog/an-absolute-beginners-guide-to-using-npm/
  An Absolute Beginner's Guide to Using npm

- https://semver.org/
  Semantic Versioning

- https://getbootstrap.com/
  Bootstrap home page

- https://getbootstrap.com/docs/5.3/layout/grid/
  The Bootstrap Grid System – containers, rows and columns

- https://getbootstrap.com/docs/5.3/components/card/
  Bootstrap cards

- https://getbootstrap.com/docs/5.3/utilities/colors/
  Bootstrap colours