

# COM661 Full Stack Strategies and Development

## Practical D1: Introducing Angular

---

### Aims

- To appreciate the purpose of Angular in full-stack architecture
- To create a default Angular application
- To understand the code structure of an Angular applicaton
- To introduce the structure and role of an Angular controller
- To demonstrate injection of data values into the web presentation
- To introduce Angular directives
- 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)

### Contents

<b>D1.1 INSTALLATION AND FIRST USE .....</b>	<b>2</b>
D1.1.1 ANGULARJS AND ANGULAR .....	2
D1.1.2 GETTING STARTED .....	3
D1.1.3 BASIC STRUCTURE OF AN ANGULAR APP .....	5
<b>D1.2 SPECIFYING CUSTOM COMPONENTS.....</b>	<b>8</b>
D1.2.1 CREATING A COMPONENT .....	8
D1.2.2 CONNECTING A COMPONENT TO THE APPLICATION .....	10
<b>D1.3 USING ANGULAR DIRECTIVES.....</b>	<b>12</b>
D1.3.1 MANIPULATING JSON DATA .....	12
D1.3.2 USING THE *NGFOR DIRECTIVE .....	13
<b>D1.4 USING BOOTSTRAP .....</b>	<b>14</b>
D1.4.1 INSTALLING BOOTSTRAP.....	14
D1.4.2 STYLING WITH BOOTSTRAP .....	15
<b>D1.5 NPM AND PACKAGE MANAGEMENT .....</b>	<b>17</b>
D1.5.1 UNDERSTANDING VERSION NUMBERS .....	18
D1.5.2 BACKUP AND RESTORE OF THE ANGULAR APPLICATION .....	19
<b>D1.6 FURTHER INFORMATION.....</b>	<b>20</b>

## D1.1 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.

### D1.1.1 AngularJS and Angular

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 publically 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 (Ref: <https://www.infoworld.com/article/2612250/application-development/application-development-the-10-hottest-javascript-framework-projects.html>) 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 v12, but it is important to recognize that versions v2 and v4–v12 (there was no v3!) are compatible with each other. 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.

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.

## D1.1.2 Getting Started

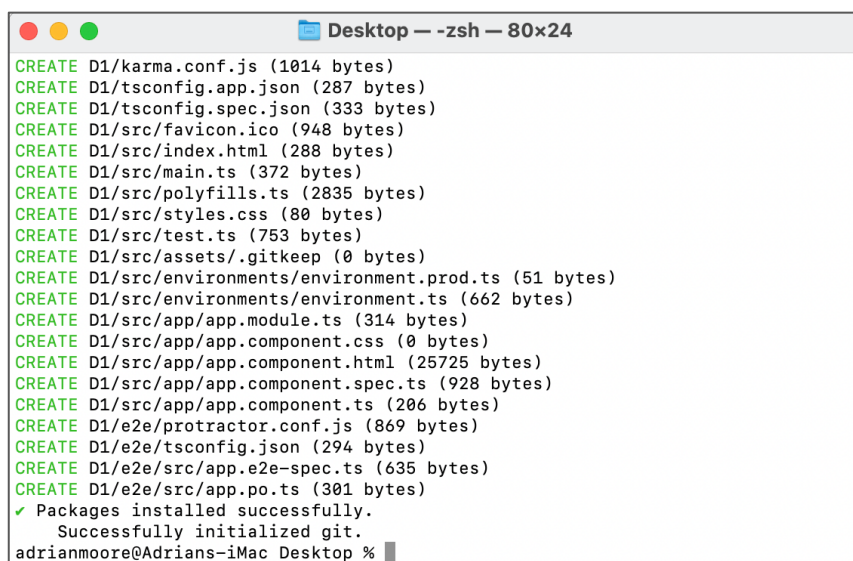
**Note:** Angular has been provided for you on the lab machines but if you need to install it on your own computer, you can do so (as long as **npm** has previously been installed) by issuing the command  
**C:\> npm install -g @angular/cli**  
**npm (Node Package Manager)** is a component of Node.js.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

We will build up our front-end application in 6 stages, so we will create a directory called **D1** for the first stage and build a default Angular application by the command

```
C:\Users\B00123456\Desktop> ng new D1
```

**Note:** Given the volume of data that needs to be created/written to build an Angular application framework (typically over 500 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.

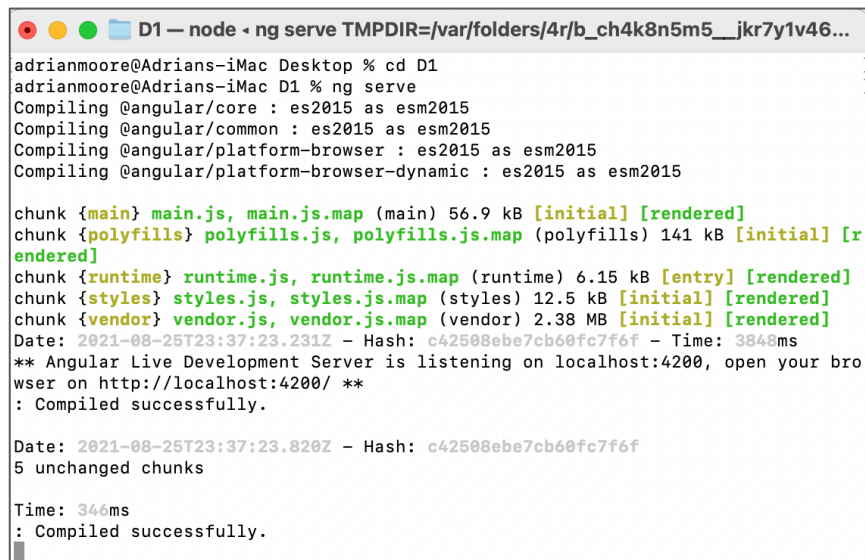


```
CREATE D1/karma.conf.js (1014 bytes)
CREATE D1/tsconfig.app.json (287 bytes)
CREATE D1/tsconfig.spec.json (333 bytes)
CREATE D1/src/favicon.ico (948 bytes)
CREATE D1/src/index.html (288 bytes)
CREATE D1/src/main.ts (372 bytes)
CREATE D1/src/polyfills.ts (2835 bytes)
CREATE D1/src/styles.css (80 bytes)
CREATE D1/src/test.ts (753 bytes)
CREATE D1/src/assets/.gitkeep (0 bytes)
CREATE D1/src/environments/environment.prod.ts (51 bytes)
CREATE D1/src/environments/environment.ts (662 bytes)
CREATE D1/src/app/app.module.ts (314 bytes)
CREATE D1/src/app/app.component.css (0 bytes)
CREATE D1/src/app/app.component.html (25725 bytes)
CREATE D1/src/app/app.component.spec.ts (928 bytes)
CREATE D1/src/app/app.component.ts (206 bytes)
CREATE D1/e2e/protractor.conf.js (869 bytes)
CREATE D1/e2e/tsconfig.json (294 bytes)
CREATE D1/e2e/src/app.e2e-spec.ts (635 bytes)
CREATE D1/e2e/src/app.po.ts (301 bytes)
✓ Packages installed successfully.
  Successfully initialized git.
adrianmoore@Adrians-iMac Desktop %
```

*Figure D1.1 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 **D1** directory and launching it with the commands

```
C:\Users\B00123456\Desktop> cd D1
C:\Users\B00123456\Desktop\D1> ng serve
```



```
D1 — node · ng serve TMPDIR=/var/folders/4r/b_ch4k8n5m5__jkr7y1v46...
adrianmoore@Adrians-iMac Desktop % cd D1
adrianmoore@Adrians-iMac D1 % ng serve
Compiling @angular/core : es2015 as esm2015
Compiling @angular/common : es2015 as esm2015
Compiling @angular/platform-browser : es2015 as esm2015
Compiling @angular/platform-browser-dynamic : es2015 as esm2015

chunk {main} main.js, main.js.map (main) 56.9 kB [initial] [rendered]
chunk {polyfills} polyfills.js, polyfills.js.map (polyfills) 141 kB [initial] [rendered]
chunk {runtime} runtime.js, runtime.js.map (runtime) 6.15 kB [entry] [rendered]
chunk {styles} styles.js, styles.js.map (styles) 12.5 kB [initial] [rendered]
chunk {vendor} vendor.js, vendor.js.map (vendor) 2.38 MB [initial] [rendered]
Date: 2021-08-25T23:37:23.231Z - Hash: c42508ebe7cb60fc7f6f - Time: 3840ms
** Angular Live Development Server is listening on localhost:4200, open your browser on http://localhost:4200/ **
: Compiled successfully.

Date: 2021-08-25T23:37:23.820Z - Hash: c42508ebe7cb60fc7f6f
5 unchanged chunks

Time: 344ms
: Compiled successfully.
```

*Figure D1.2 Running the frontend 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 homepage as shown below.

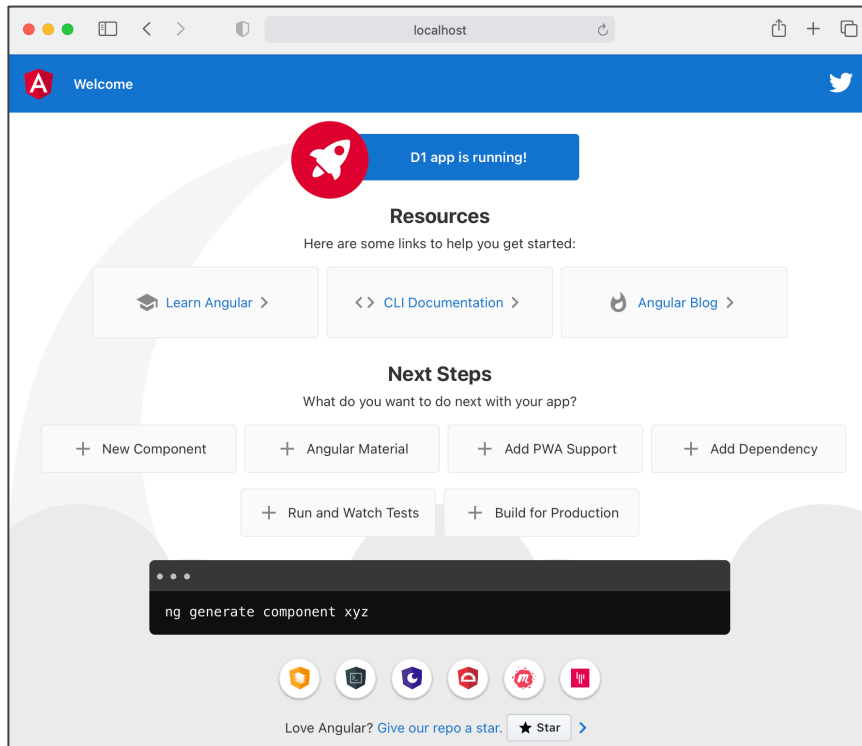


Figure D1.3 Default Server Output

**Do it now!** Create a default Angular application called **D1** 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 D1.3 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

### D1.1.3 Basic Structure of an Angular App

The Angular CLI creates all of the folders and files that will support our application development. All of 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 Controller** – the basic building block of an Angular application. Controllers 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 controller generates the display shown in the browser.

**File: D1/src/index.html**

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>D1</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 controllers are defined in the **app** sub-directory of **src**, so first look at the default controller TypeScript file, **app.component.ts**.

**File: D1/src/app/app.component.ts**

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

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'D1';
}
```

The controller TypeScript file consists of 3 sections

- i) The **import** statement that enables the file to make use of the facilities of the Angular **Component** library
- 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. Note that there is a single HTML template while the stylesheets are specified as a list.

- iii) Finally, the **class definition** that contains the logic for the component. Here, we set up a single variable called **title** with the value '**D1**'

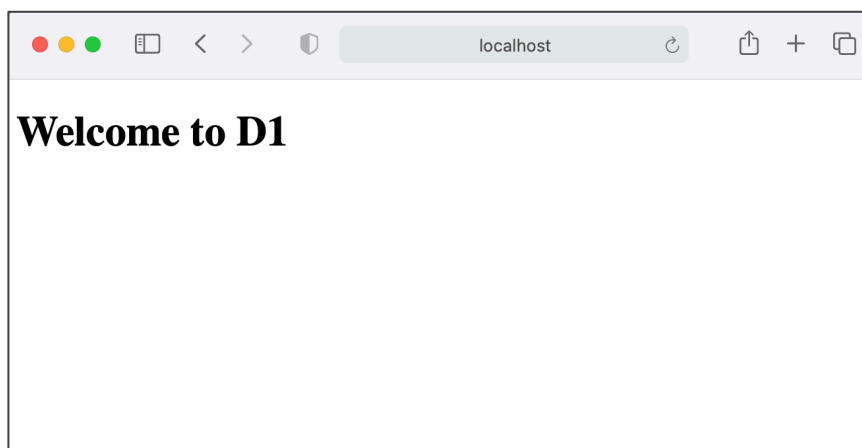
To best understand the structure of the default Angular application, we will replace all of the code in ***app.component.html*** with that provided below.

**File: D1/src/app/app.component.ts**

```
<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 **{{ }}**.

**Do it now!** Modify the code of ***app.component.html*** as shown above and verify that you receive output such as that illustrated by Figure D1.4 below.



*Figure D1.4 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.

## D1.2 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.

### D1.2.1 Creating a Component

The first stage is to create the files that will be required to house our component. Our component will be called ***BusinessesComponent***, so the first step is to create 3 empty files called

- *businesses.component.ts*,
- *businesses.component.html* and
- *businesses.component.css*

in the ***src/app*** folder of the **D1** application.

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



**File: D1/src/app/businesses.component.html**

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

and create the component TypeScript file by copying and pasting the content from ***app.component.ts*** into our new ***businesses.component.ts*** and changing the selector, template, style and class names to identify the new component as highlighted below.

**File: D1/src/app/businesses.component.ts**

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

@Component({
  selector: 'businesses',
  templateUrl: './businesses.component.html',
  styleUrls: ['./businesses.component.css']
})
export class BusinessesComponent { }
```

Next, we use the new selector by adding a **<businesses></businesses>** tag to the ***app.component.html*** file to render our new ***BusinessesComponent*** component.

**File: D1/src/app/app.component.html**

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

<businesses></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 ***businesses*** element is unknown, as shown in Figure D1.5 below.

```
D1 — node - ng serve TMPDIR=/var/folders/4r/b_ch4k8n5m5__jkr7y1v467r0000gn/T/ __CFB...
Date: 2021-08-26T00:01:52.032Z - Hash: be6b0def939c2c2e5123
4 unchanged chunks
chunk {main} main.js, main.js.map (main) 8.76 kB [initial] [rendered]
Time: 121ms
: Compiled successfully.

ERROR in src/app/app.component.html:5:1 - error NG8001: 'businesses' is not a known element:
1. If 'businesses' is an Angular component, then verify that it is part of this module.
2. To allow any element add 'NO_ERRORS_SCHEMA' to the '@NgModule.schemas' of this component.

5 <businesses></businesses>
  ~~~~~

src/app/app.component.ts:5:16
5   templateUrl: './app.component.html',
  ~~~~~
Error occurs in the template of component AppComponent.
```

Figure D1.5 Unregistered component

**Do it now!** Create the new **businesses.component** files as described above and verify that you receive the “unregistered component” error in the Terminal window as shown in Figure D1.5

**Note:** Versions of Angular earlier than v10 reported this error in the Browser console rather than the Terminal window. You should always keep an eye on both when developing Angular code.

### D1.2.2 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 **BusinessesComponent** component into **app.component.ts**, as the component is rendered within its template.

File: D1/src/app/app.component.ts

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

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'D1';
}
```

Then, we need to register the new component in the main **app.module.ts** file by **importing** it and adding it to the **app.module declarations** list.

**File: D1/src/app/app.module.ts**

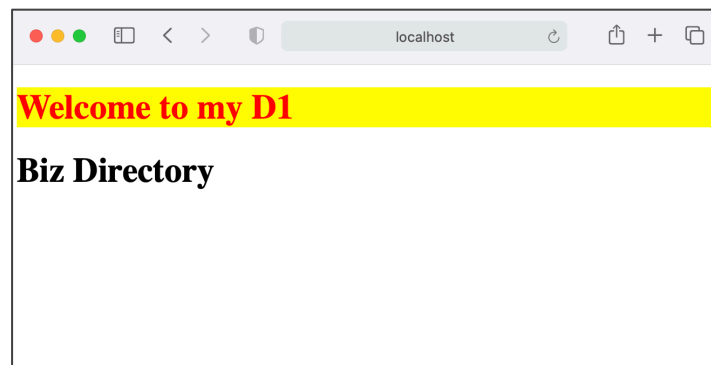
```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';

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

@NgModule({
  declarations: [
    AppComponent, BusinessesComponent
  ],
  imports: [
    BrowserModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})

export class AppModule { }
```

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 D1.6 below.



*Figure D1.6 New component correctly rendered*

<b>Do it now!</b>	Register the <b>BusinessesComponent</b> with the application as shown above. Verify that you receive output as illustrated in Figure D1.6.
-------------------	--

## D1.3 Using Angular Directives

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.

### D1.3.1 Manipulating JSON Data

In the **BusinessesComponent** class of the *businesses.component.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", "city" and "review\_count" (reflecting 3 of the fields in our data set from the *Biz* API application).

File: D1/src/app/businesses.component.ts

```
...

export class BusinessesComponent {
  business_list = [
    { "name": "Pizza Place",
      "city": "Coleraine",
      "review_count": 10 },
    { "name": "Wine Lake",
      "city": "Ballymoney",
      "review_count": 7 },
    { "name": "Beer Tavern",
      "city": "Ballymena",
      "review_count": 12 }
  ];
}
```

Now, we add code to the *businesses.component.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: D1/src/app/businesses.component.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 D1.7 below.

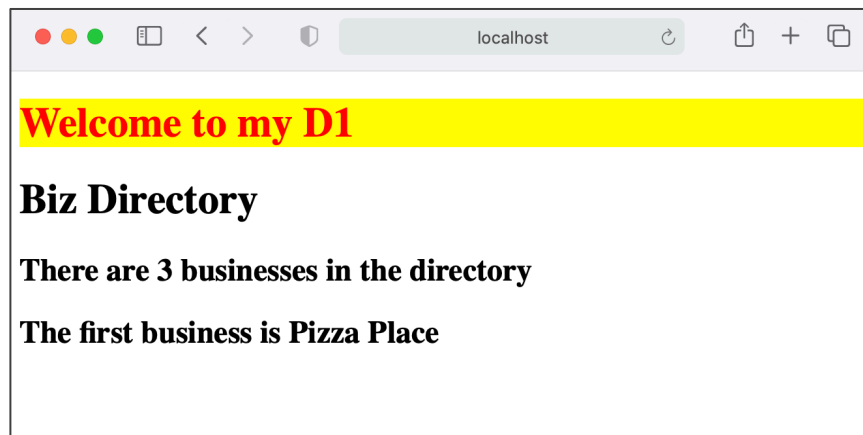


Figure D1.7 Injecting data into the web page

### D1.3.2 Using the **\*ngFor** Directive

Angular provides a number of very useful **directives** that enable us to introduce common programming constructs into our HTML specification. One of the most useful of these is the **\*ngFor** directive, that provides a looping mechanism.

**\*ngFor** takes a value in the form “let **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 **\*ngFor** element is applied to a **<div>** element that contains a paragraph that presents the *name*, *city* and *review\_count* properties of each element of **business\_list** in turn.

File: D1/src/app/businesses.component.html

```
...  
  
<div *ngFor="let business of business_list">  
  <p>Name: {{ business.name }} <br>  
    City: {{ business.city }} <br>  
    Reviews: {{ business.review_count }}  
  </p>  
</div>
```

The effect of this is to generate separate **<div>** elements for each business as shown in Figure D1.8 below.

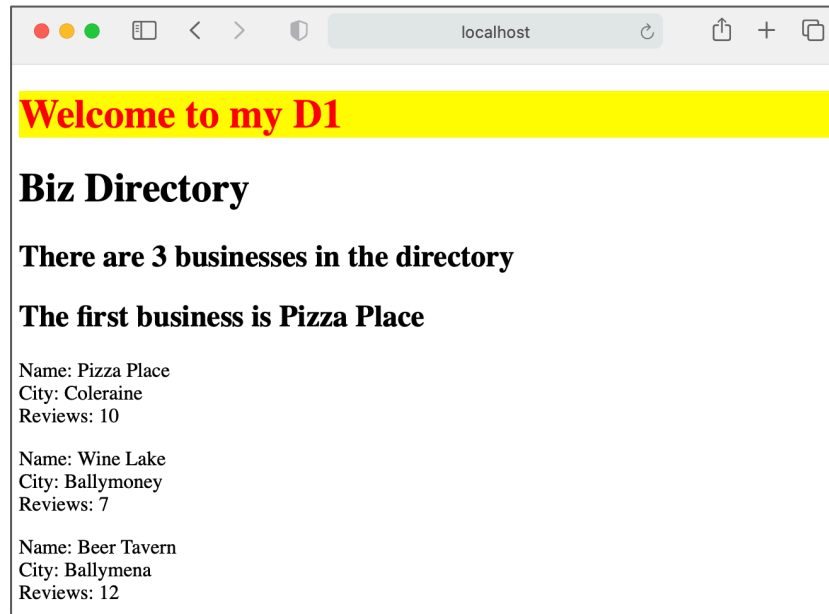


Figure D1.8 Using *\*ngFor* to traverse a dataset

## D1.4 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.

### D1.4.1 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 packages **jQuery** and **Popper**, kill any currently running Angular server (using CTRL-C), navigate back to the **D1** 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\D1> npm install bootstrap  
jquery popper --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 | D1 | architect | build | options** object). Now add the location of the file *bootstrap.min.css* to the styles entry as shown below.

File: D1/.angular.json

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

We can check that Bootstrap has been installed by running the application by the command **ng serve**, opening the browser console on the *Elements* tab and examining the **head|styles** section. Expanding the **styles** area verifies that Bootstrap has been included, as shown in Figure D1.9 below.

**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.

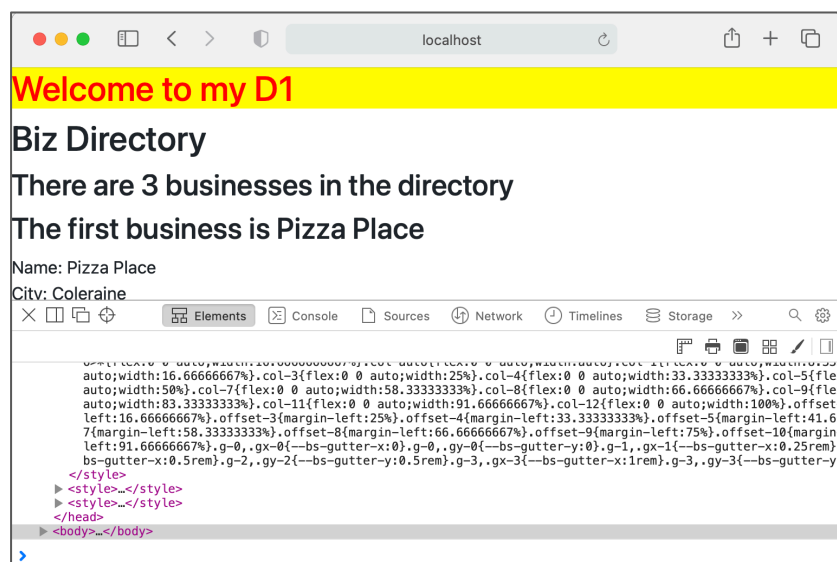


Figure D1.9 Verifying that Bootstrap is installed

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

### D1.4.2 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 options 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/5.1/components/card/>. The code box below shows the full revised specification for *businesses.component.html*.

**File: D1 src/app/businesses.component.html**

```
<div class="container">
  <div class="row">
    <div class="col-sm-12">
      <div *ngFor = "let business of business_list">
        <div class="card text-white bg-primary mb-3"
          style = "width: 50rem; margin: auto">
          <div class="card-header">
            {{ business.name }}
          </div>
          <div class="card-body">
            This business is based in
            {{ business.city }}
          </div>
          <div class="card-footer">
            {{ business.review_count }}
            reviews available
          </div>
        </div>
      </div>
    </div> <!-- col -->
  </div> <!-- row -->
</div> <!-- container -->
```

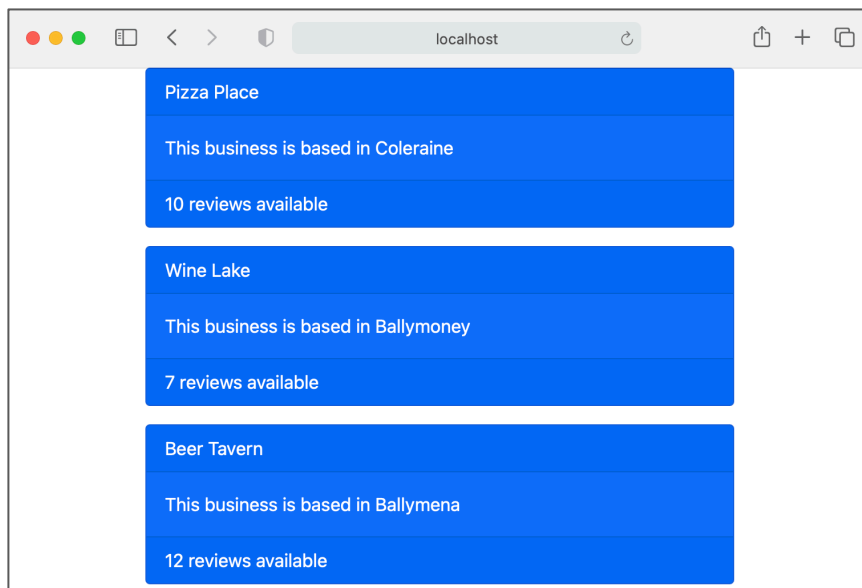


Finally, we remove all other code from ***app.component.html*** so that the entire presentation consists of our own work in the ***businesses*** component.

**File: D1 src/app/app.component.html**

```
<businesses></businesses>
```

The effect of these changes can be seen in Figure D1.10, below.



*Figure D1.10 Using Bootstrap cards*

<b>Do it now!</b>	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 D1.10 above.
-------------------	---

<b>Try it now!</b>	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.
--------------------	--

## D1.5 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.

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

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

### D1.5.1 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": "^5.1.0",
```

which denotes Major version 5, Minor version 1 and Patch version 0. 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": "^5.1.0",
```

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.

### D1.5.2 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 **D1** folder for our current application, you will find that it occupies in the region of 520 Mbytes across over 40,000 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**, which opens *package.json*, creates a **node\_modules** folder and installs all of the components listed in *package.json* into it.

<b>Do it now!</b>	Stop the Angular server and delete the folder <b>node_modules</b> from the D1 application. Then, from a Command prompt in the D1 folder, issue the command <b>npm install</b> to re-create the application. Once installed, run it by the command <b>ng serve</b> and verify that everything works as previously.
-------------------	---

## D1.6 Further Information

- <https://angular.io/>  
Angular home page
- <https://www.codingforentrepreneurs.com/blog/angular-setup-guide>  
Angular Setup, Install & Build Guide
- <https://peakup.org/blog/how-to-setup-angular-6-environment-on-windows/>  
Setting up Angular in Windows (works for all Angular >= v6)
- <https://angular.io/start>  
Getting Started with Angular – Your First App
- <https://angular.io/guide/architecture-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/5.1/layout/grid/>  
The Bootstrap Grid System – containers, rows and columns
- <https://getbootstrap.com/docs/5.1/components/card/>  
Bootstrap cards
- <https://getbootstrap.com/docs/5.1/utilities/colors/>  
Bootstrap colours