COM661 Full Stack Strategies and Development

FE14. Data-driven Functionality

Aims

- To introduce Angular Directives
- To represent numeric data in a visual fashion
- To integrate Google Maps into the application
- To make use of representative placeholder content to populate the interface of an application
- To harvest data from an external API
- To introduce SVG for graphical representation of data
- To demonstrate layout design for effective representation of information

Table of Contents

14.1 Data Visualisation

In this practical we will demonstrate how we can use various techniques and plug-ins to generate additional dynamic functionality from our data. Currently, each of the business in our collection is described by a name, a town, a rating, a coordinate location and a set of profit/loss figures for the previous three years, and although displaying these as text within a Bootstrap Card does impart the information, we can to this in a much more interesting way by considering how the information might otherwise be represented on the web page.

First, let's consider how we display the business rating as a value from 1-5. Presently, it is displayed as a simple text message within the footer of the Bootstrap card, but it might be more clearly displayed by a star representation, where the number of stars is the figure for the business rating.

We have already seen the Angular **`@for()`** block that allows blocks of HTML content to be repeated for a given number of times. The same effect can be achieved in a more compact form by the **`*ngFor`** directive that iterates over a collection by the syntax

```
*ngFor = "let item of collection"
```

where the loop variable ***`item`*** takes the value of each element of ***`collection`*** in turn. For our "stars" use case, we can make use of this by defining an array containing as many items as stars that we want to display. This can be achieved by using the constructor of the array element and passing the number of elements required as a parameter. Hence the expression

```
[].constructor(business.rating)
```

will create an array with a number of elements dictated by the value of the business **`rating`** field.

By wrapping the display of a star image with a **`<span>`** element that deploys **`*ngFor`** to display that image a certain number of times, we can generate a number of stars equal to the rating of the business. This is shown by the code box below which we can deploy to the card footer instead of the current business rating message.

Note how the image file ***star.png*** is represente3d in the ***/images*** folder. This is a sub-folder of the ***public*** folder that will have been created at the top level of your Angular project structure. ***public*** can be used for those assets that need to be accessible directly by the browser, such as images, logos, sounds and other media elements.

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

```
...

 <div class="card-footer">
    <span *ngFor="let star of [].constructor(business.rating)">
          <img src="images/star.png"
               style="width:30px; height:30px">
    </span>
 </div>

...
```

To use *ngFor in the component it needs to be imported in the BusinessComponent TypeScript file and added to the list of component imports as below.

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

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

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

Making these changes and visiting the page for an individual business shows how the rating is displayed as illustrated in Figure 14.1 below.

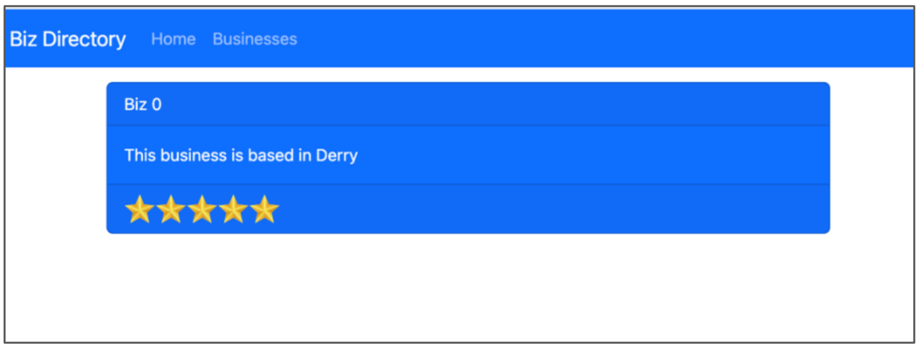


Figure 14.1 Graphical Representation of the Business Rating

Do it now!	Make the change shown above to ***business.component.html*** to change the way in which the rating of a business is displayed. Remember to create ***images*** as a sub-folder of ***public*** and to copy the ***stars.png*** file found in the **Practical BE13 Files** to this new folder.

14.2 Using Coordinate Data

One of the most interesting features of our dataset is the provision of a realistic coordinate position for each business through latitude and longitude values. An obvious way in which these could be used would be in the display of a map that provides a visual representation of the location of the business.

14.2.1 Using Google Maps

Note:	Using Google Maps within your applications requires that you have a **Google Cloud Platform developer account** and that you create a **Google Maps API key**. This will require that you register a payment card with Google, but the level of free credit provided is such that you should never be charged. However, if you are uncomfortable with registering a card, please feel free to skip this section and replace the Map that we will add to our **Business Component** template with a representative image from the internet.

To create a new Google Cloud Platform developer account and a Google Maps project, visit https://developers.google.com/maps/get-started and following the 3-step process described. Once you have created your project in Step 1, visit the **Maps API Library Page** in Step 2 and enable the **Maps JavaScript API**. Then, in Step 3, click the link for the **Google Maps Platform Credentials** page and follow the link to **Maps Platform API Key**. From here, you will be able to copy the API key that you will need to embed in your application.

Do it now!	Create a Google Cloud Platform developer account and create a new project. In that project, enable the Google Maps JavaScript API and obtain an API key.

To use Maps in our Angular applications, there is a very useful Google Maps module that we can install using **npm** by the command

```
npm install @angular/google-maps --save
```

Then, we need to import the **GoogleMapsModule** into the **BusinessComponent** in which we want to display a map. This is achieved by the modifications to ***business.component.ts*** shown in the following code box.

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

```
...
import { GoogleMapsModule } from '@angular/google-maps'

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

...
```

14.2.2 Generating a First Map

With the **GoogleMapsModule** installed, we can now add a Map object to our **BusinessComponent** HTML template. The key information for the Map is the coordinate position at which it is centered, so we retrieve the latitude and longitude values from the currently selected business and use them to initialize the **lat** and **lng** properties of the **MapOptions** object.

To accomplish this, we add new properties to the **BusinessComponent** to store the latitude value as **business_lat**, the longitude value as **business_lng**, and the

`MapOptions` object as **`map_options`**. We also initialise the **`map_locations`** property that will hold map markers as an empty list.

Then, we modify the component **`ngOnInit()`** method to initialise the latitude and longitude values to those associated with the current business object, as well as initializing the **`map_options`** with the default **`mapId`** and centering the map on the latitude and longitude values.

The modifications in ***business.component.ts*** are described in the code box that follows.

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

```
...

export class BusinessComponent {

    business_list: any;
    business_lat: any;
    business_lng: any;
    map_options: google.maps.MapOptions = {};
    map_locations: any[] = [ ]

    ...

    ngOnInit() {
        this.business_list =
             this.dataService.getBusiness(
                  this.route.snapshot.paramMap.get('id'))
        this.business_lat =
             this.business_list[0].location.coordinates[0];
        this.business_lng =
             this.business_list[0].location.coordinates[1];

        this.map_options = {
          mapId: "DEMO_MAP_ID",
          center: { lat: this.business_lat,
                    lng: this.business_lng },
          zoom: 13,
        };
      }

...
```

Now that the **`ngOnit()`** method has been set to initialize the map when the **`BusinessComponent`** loads, we can add the HTML tag that inserts the map onto the web page. In the business.component.html template, we add a **`<google-map>`** tag which specifies the dimensions of the map on the page and binds the **`options`** property to the **`map_options`** element specified above.

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

```
      ...

      <google-map height="400px" width="400px"
              [options]="map_options">
      </google-map>

      ...
```

Finally, we need to add the following script to the application's ***src/index.html*** file and paste the JavaScript content provided in the Google Maps Script.html file immediately before the closing **`</body>`** tag. Remember to replace the placeholder **YOUR_API_KEY** with your corresponding Google Maps developer API key.

File: bizFE/src/index.html

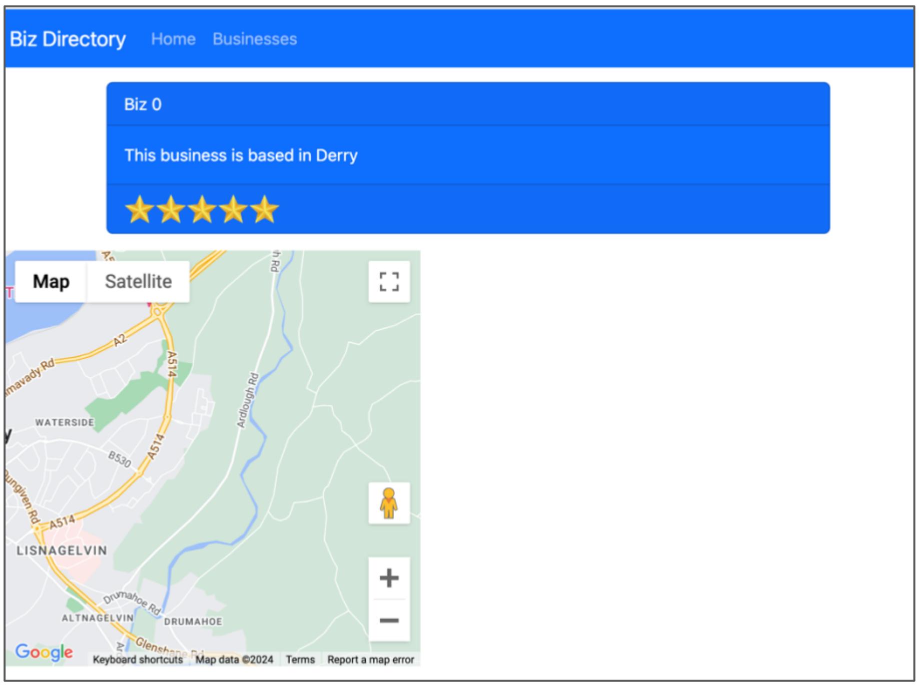
```
      ...

      <body>
        ...
      <!–- Paste the <script> element provided in the FE14 Practical
           Files here -->
      </body>

      ...
```

With the Typescript setup code and the HTML template in place, we can then run the application and select an individual business to see its location centered on a Google Map that is displayed on the page.

Do it now!	Make the modifications to the TypeScript and HTML files of the **`BusinessComponent`** as described above. Run the application and select an individual business and confirm that the map displayed is centered on the business location as shown in Figure 14.2, below. Try different businesses to see how the map moves from town to town with the business location.



Biz 0

This business is based in Derry

★★★★★

Figure 14.2 Creating a Google Map

With the Map displayed and centered on the business, we can now add a marker to the Map that indicates the precise location. Earlier, we defined **`map_locations`** in the **`BusinessComponent`** as an array so that the code the follows will support multiple markers, but here we will add a single marker indicating the business coordinate position.ng

For each marker to be displayed, we need to push an object containing **`lat`** and **`lng`** values to the **`map_locations`** property of the component. Here, we push a single location by using the **`business_lat`** and **`business_lng`** values that were assigned for the current business from the values retrieved from the **`DataService`**.

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

```
      ...

          ngOnInit() {
              this.business_list =
                   this.dataService.getBusiness(
                        this.route.snapshot.paramMap.get('id'))
              this.business_lat =
                   this.business_list[0].location.coordinates[0];
              this.business_lng =
                   this.business_list[0].location.coordinates[1];

              this.map_locations.push( {
                lat: this.business_lat, lng: this.business_lng
              });

              ...
```

Now, we can add a **`@for()`** block to the **`<google-map>`** element in the **`BusinessCompoent`** template to iterate over the contents of **`map_locations`** and add a **`<map-advanced-marker>`** element for each location specified.

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

```
      ...

      <google-map height="400px" width="400px"
              [options]="map_options">
           @for (location of map_locations; track location) {
              <map-advanced-marker
                #markerElem="mapAdvancedMarker"
                [position]="{ lat: location.lat,
                              lng: location.lng }">
              </map-advanced-marker>
           }
       </google-map>
      ...
```

Do it now!	Make the modifications to ***business.compoennt.ts*** and ***business.component.html*** described above and run the application. Verify that a marker has been added to the map at the location of the business being displayed as shown in Figure 14.3, below.

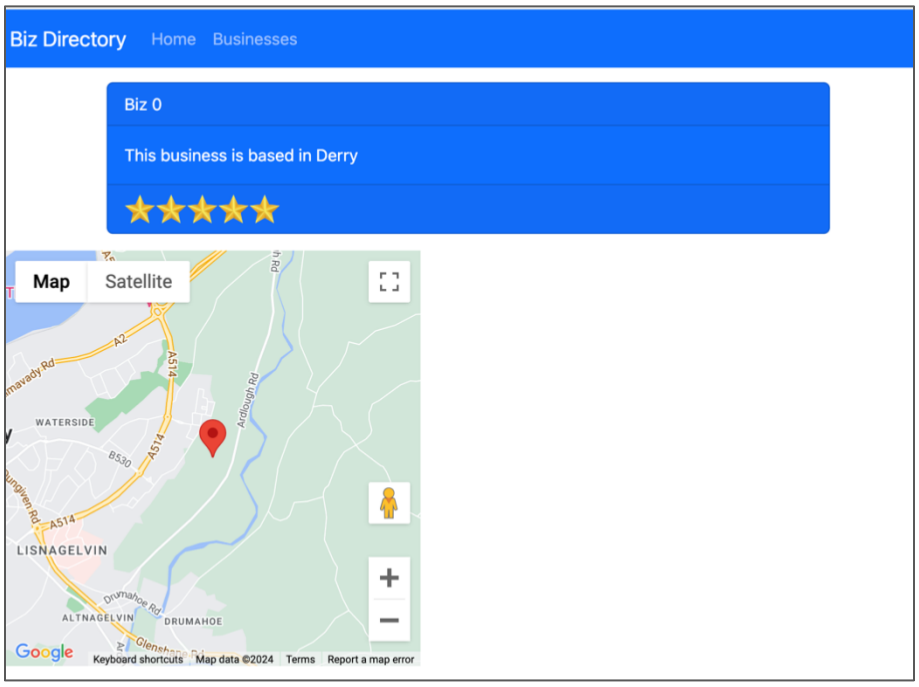


Figure 14.3 Google Map with a Marker

14.3 Generating Placeholder Content

Sometimes, our application is a demonstration of some concept or idea that either has no live data or where the data being used is incomplete. In such cases, we can use external sources of dummy data to populate the front-end and there are many potential sources for this. In this section, we make use of an external API from api-ninjas.com to generate dummy text that will represent a description of a business in the absence of a corresponding field in our randomly generated dataset.

14.3.1 A Lorem Ipsum Generator

In publishing and graphic design, **Lorem Ipsum** is meaningless placeholder text that occupies the space where the real information would be displayed. It is often used in industry when layout is to be presented but actual content is not yet available.

api-ninjas.com makes available a very useful range of APIs which can be used to generate a wide range of types of sample data. Among their catalogue is an API that supplies Lorem Ipsum text which we will use to populate our business description.

Do it now!	Sign up for a free account at https://www.api-ninjas.com and obtain an API key for the Lorem Ipsum API.

With the API key available, we can now add a new method to the DataService to return a specified number of random Lorem Ipsum paragraphs from the API.

The call to the external API requires that we make an **HTTP GET** request to the API endpoint https://api.api-ninjas.com/v1/loremipsum?paragraphs=2 where the query string parameter '2' is the number of paragraphs being requested. We also pass our API key as an **X-Api-Key** request header. Note in the code box below how the HTTP request returns an Observable object.We will cover Observables in more detail later when we connect our front end to the API built earlier in the module, but for now we can describe an Observable as a stream of data that happens over time. Rather than read a single value, we subscribe to the stream and accept and process data as it arrives. The new **`getLoremIpsum()`** method is shown in the code box below.

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

```
    ...

        getLoremIpsum(paragraphs: number): Observable<any> {

            let API_key = 'YOUR-API-KEY';
            return this.http.get<any>(
                        'https://api.api-ninjas.com/v1/' +
                        'loremipsum?paragraphs=' + paragraphs,
                        { headers: { 'X-Api-Key': API_key } }
            );

    ...
```

In order to support the new method, we also need to import new modules **`HttpClient`**, **`Injectable`** and **`Observable`** into the **`DataService`** as described below. We also need to make the component injectable by adding the **`@Injectable`** decorator to the component class and make the **`HttpModule`** available for use by injecting it to the component by providing it as a parameter to the component constructor

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

```
import jsonData from '../assets/businesses.json';
import { HttpClient } from '@angular/common/http';
import { Injectable } from '@angular/core';
import { Observable } from 'rxjs';

@Injectable( {
    providedIn: "root",
})
export class DataService {
    pageSize: number = 3;

    constructor(private http: HttpClient) {}

...
```

The final stage in enabling the **`HttpClient`** is to **`import`** the **`provideHttpClient`** module into ***app.config.ts*** and add it to the list of **`providers`** specified.

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

```
import { ApplicationConfig,
         provideZoneChangeDetection } from '@angular/core';
import { provideRouter } from '@angular/router';
import { routes } from './app.routes';
import { provideHttpClient } from '@angular/common/http';

export const appConfig: ApplicationConfig = {
  providers: [
      provideZoneChangeDetection({ eventCoalescing: true }),
      provideRouter(routes),
      provideHttpClient()
  ]
};
```

With the **`DataService`** updated and the **`HttpClient`** registered with the application, we can now update the **`BusinessComponent`** to call the Lorem Ipsum generator from the **`ngOnInit()`** method and log the text received to the console.

Note in the code box below how, as the call to **`getLoremIpsum()`** returns an Observable, we need to subscribe to it by chaining the **`subscribe()`** method to the request. The **`subscribe()`** method takes as a parameter a function that is written in the “fat arrow” style. Fat arrow functions (sometimes just called "arrow functions" are a concise notation for short (often single line) functions that do not have a function name and do not return a value. The parameter to the function is on the left-hand side of the **=>** symbol, while the code to be executed is on the right-hand side. Here, the parameter is the data returned from the Observable, while the response is simply to log that value to the browser console.

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

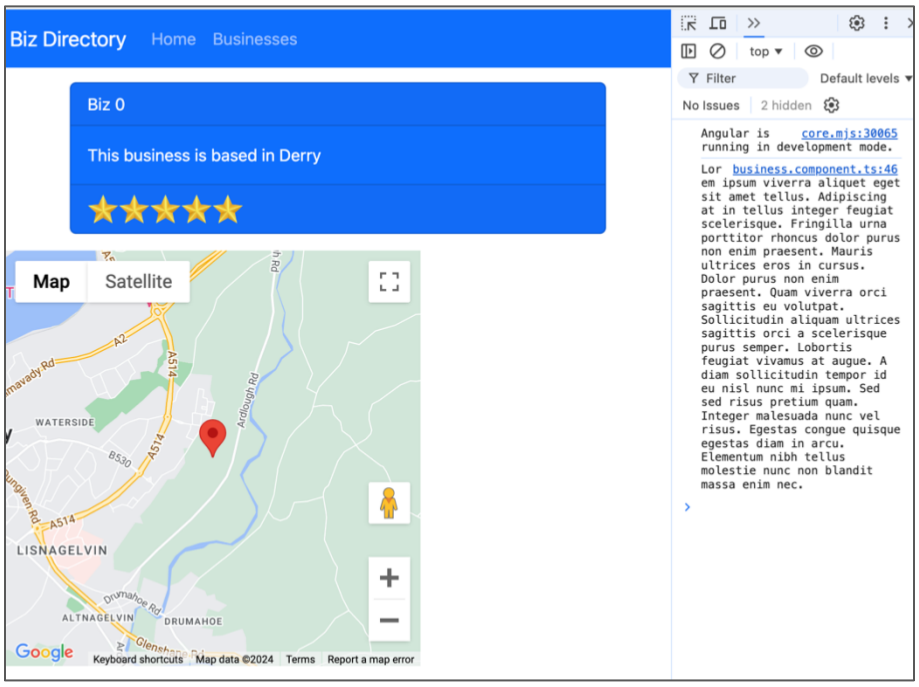
```
...

    ngOnInit() {

        ...

        this.dataService.getLoremIpsum(1)
          .subscribe((response: any) => {
              console.log(response.text);
        })};
    }
...
```

Do it now! Update the files ***data.service.ts***, ***app.service.ts*** and ***business.component.ts*** with the code presented above and run the application with the browser console open. Verify that displaying the page for an individual business results in a stream of Lorem Ipsum text being retrieved from the API and displayed in the browser console as shown in Figure 14.4, below.



```
Angular is        core.mjs:30065
running in development mode.

Lor  business.component.ts:46
em ipsum viverra aliquet eget
sit amet tellus. Adipiscing
at in tellus integer feugiat
scelerisque. Fringilla urna
porttitor rhoncus dolor purus
non enim praesent. Mauris
ultrices eros in cursus.
Dolor purus non enim
praesent. Quam viverra orci
sagittis eu volutpat.
Sollicitudin aliquam ultrices
sagittis orci a scelerisque
purus semper. Lobortis
feugiat vivamus at augue. A
diam sollicitudin tempor id
eu nisl nunc mi ipsum. Sed
sed risus pretium quam.
Integer malesuada nunc vel
risus. Egestas congue quisque
egestas diam in arcu.
Elementum nibh tellus
molestie nunc non blandit
massa enim nec.
```

Figure 14.4 Lorem Ipsum Retrieved as a Placeholder

Now, we can add the Lorem Ipsum retrieved from the API as placeholder text for the description of the business. First, we define a new component property **`loremIpsum`** in which to store the text, and then we update the **`ngOnInit()`** method, replacing the previous **`console.log()`** with code to return the first 400 characters of the retrieved text of the new **`loremIpsum`** property.

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

```
...

export class BusinessComponent {
    ...

    loremIpsum: any;
    ...

    ngOnInit() {

        ...

        this.dataService.getLoremIpsum(1)
          .subscribe((response: any) => {
              this.loremIpsum = response.text.slice(0,400);
        })};
    }
...
```

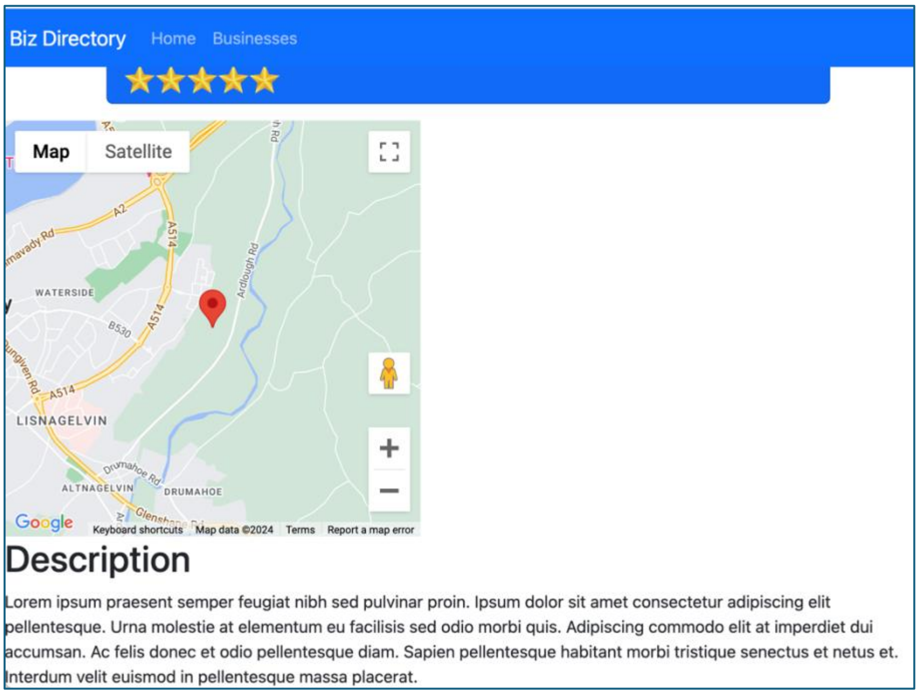
Finally, we add new content to the `BusinessCompoent` HTML template to display the Lorem Ipsum under a "Description" heading.

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

```
        . . .

        <h1>Description</h1>
        {{ loremIpsum }}
```

Do it now! Update the ***business.component.ts*** and ***business.component.html*** files with the code presented and navigate to the page for an individual business. Verify that the Lorem Ipsum description of the business is displayed as shown in Figure 14.5 below, and that the description text is randomly generated for each business selected.



Description

Lorem ipsum praesent semper feugiat nibh sed pulvinar proin. Ipsum dolor sit amet consectetur adipiscing elit pellentesque. Urna molestie at elementum eu facilisis sed odio morbi quis. Adipiscing commodo elit at imperdiet dui accumsan. Ac felis donec et odio pellentesque diam. Sapien pellentesque habitant morbi tristique senectus et netus et. Interdum velit euismod in pellentesque massa placerat.

Figure 14.5 Lorem Ipsum in the Browser

14.4 Using Application Data from External APIs

The previous example made use of an external API to obtain random data that can enhance the presentation of our application, but there are also opportunities to use information from our dataset as parameters to an external API call and retrieve real-world live data that can be integrated into our front end.

14.4.1 Connecting to OpenWeatherMap API

In a previous practical, we have already seen the **OpenWeatherMap** API as an example of a commercial API that provides live data. Since each business in our dataset is associated with a location and has a realistic latitude and longitude coordinate position, we can use that data to obtain an accurate weather report for the location and display it on the page of business information.

First, we add a new method **`getCurrentWeather()`** to the Data Service, that takes the latitude and longitude values of the business as a parameter, and returns the packet of data representing the current weather conditions at that location. The call to the API endpoint is built as before, with the latitude and longitude values specified as query string parameters as well as specifying "metric" for the "units" parameter and adding our individual API key. The code for **`getCurrentWeather()`** is shown in the code box below.

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

```
      ...

          getCurrentWeather(lat: number, lon: number) {

              let API_key = "YOUR-API-KEY";
              return this.http.get<any>(
                     'https://api.openweathermap.org/data/2.5/' +
                     'weather?lat=' + lat +
                     '&lon=' + lon + '&units=metric&appid=' +
                     API_key);
          }
      ...
```

Next, we update the business component to call the new **`DataService`** method and extract the current temperature and weather description from the data retrieved from the API.

First, we add two new properties to the **`BusinessComponent`** to store the current **`temperature`** and **`weather`** description. Then, in the **`subscribe()`** method following the call to **`getCurrentWeather()`**, we extract the values for each and assign them to the respective properties. Finally, we format the weather description for display by capitalizing the first letter while leaving the rest of the description in lower case. The new code added to ***business.component.ts*** is highlighted in the code box below.

Do it now!	Update ***data.service.ts*** with the new **`getCurrentWeather()`** method. Remember to add your personal API key at the place indicated in the first line of the method body.

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

```
...

export class BusinessComponent {

    ...

    temperature: any;
    weather: any;

    ...

    ngOnInit() {

        ...

        this.dataService.getCurrentWeather(
            this.business_lat, this.business_lng)
          .subscribe((response: any) => {
              let weatherResponse =
                  response['weather'][0]['description'];
              this.temperature =
                  Math.round((response['main']['temp']));
              this.weather =
                  weatherResponse[0].toUpperCase() +
                  weatherResponse.slice(1);
          });
...
```

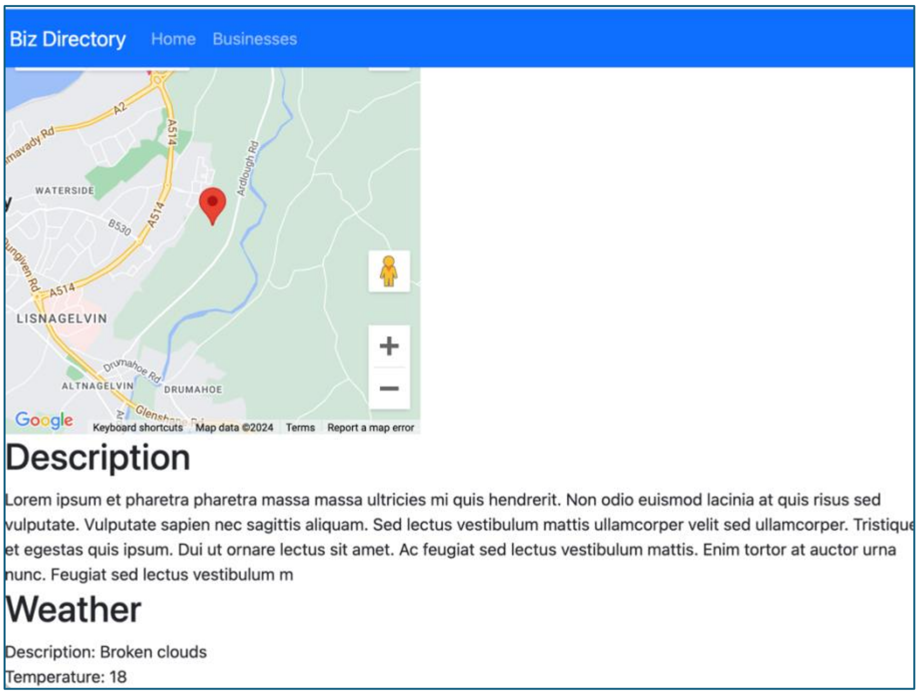
With the changes to ngOnInit() described above, loading the BusinessComponent for any individual business will now populate the properties temperature and weather with the values retrieved from OpenWeatherAPI for that specific location. We can now add these to the HTML template of the component so that they are made available to the user.

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

```
        ...

        <h1>Weather</h1>
        Description: {{ weather }} <br>
        Temperature: {{ temperature }} <br>
```

Do it now! Update the ***business.component.ts*** and ***business.component.html*** files with the code presented and navigate to the page for an individual business. Verify that the description of the current weather and the temperature value are displayed as shown in Figure 14.6 below.



Description

Lorem ipsum et pharetra pharetra massa massa ultricies mi quis hendrerit. Non odio euismod lacinia at quis risus sed vulputate. Vulputate sapien nec sagittis aliquam. Sed lectus vestibulum mattis ullamcorper velit sed ullamcorper. Tristique et egestas quis ipsum. Dui ut ornare lectus sit amet. Ac feugiat sed lectus vestibulum mattis. Enim tortor at auctor urna nunc. Feugiat sed lectus vestibulum m

Weather

Description: Broken clouds
Temperature: 18

Figure 14.6 Current Weather Information Displayed

14.4.2 Graphical Representation of the Weather Data

The previous section has successfully retrieved and displayed the live weather information for the location of the selected business, but it would be more informatively and attractively displayed in graphical form. One of the data items retrieved from the OpenWeatherMap API is the URL of a symbol describing the current weather conditons and we can use this as additional graphical information.

The weather symbols are in the form https://openweathermap.org/img/wn/10d@2x.png where **10d** is the symbol identifier retrieved from the `weather[0][‘icon’]` element of the data returned from the API and **2x** is the magnification factor at which the symbol is to be rendered. We add this to our BusinessComponent by creating a new Component property weatherIcon to store the URL of the image and weatherIconURL to store the URL from which the icon can be retrieved. We populate these by an additional retrieval from the API data as shown in the code box below.

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

```
...

export class BusinessComponent {

    ...

    temperature: any;
    weather: any;
    weatherIcon: any;
    weatherIconURL: any;

    ...

    ngOnInit() {

        ...

        this.dataService.getCurrentWeather(
             this.business_lat, this.business_lng)
          .subscribe((response: any) => {

              ...

              this.weatherIcon =
                   response['weather'][0]['icon'];
              this.weatherIconURL =
                  "https://openweathermap.org/img/wn/" +
                  this.weatherIcon + "@4x.png";
          });
...
```

Do it now!	Update the ***business.component.ts*** file with the code shown above to retrieve the weather icon code and construct its URL

Additionally, we can display the current temperature in a graphical symbol of the kind frequently shown in weather maps in broadcast and online media where the temperature is displayed inside a solid disc where the colour of the disc represents the band in which the temperature falls.

First, we add an additional method **`getTemperatureColour()`** to the **`DataService`** which returns a colour according to the temperature value. Following convention, temperatures lower than 5^{o}C are represented in blue, from 5-12 ^{o}C in green, from 13-17 ^{o}C in yellow, from 18-25 ^{o}C in orange and temperatures of 26 ^{o}C or higher are in red. The new method is presented in the code box below.

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

```
      ...

          getTemperatureColour(temp: number) {
              if (temp <= 5) return "#0000ff";
              else if (temp <= 12) return "#00ff00";
              else if (temp <= 17) return "#ffff00";
              else if (temp <= 25) return "#ff7f00";
              else return "#ff0000"
          }

      ...
```

Now, we return to the **`BusinessComponent`** **`ngOnInit()`** method and add a new **`temperatureColour`** property which we populate by a call to **`getTemperatureColour()`** passing the current temperature value as a parameter.

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

```
...

export class BusinessComponent {

    ...

    temperatureColour: any
    ...

    ngOnInit() {

        ...

        this.dataService.getCurrentWeather(
            this.business_lat, this.business_lng)
          .subscribe((response: any) => {

            ...

            this.temperatureColour =
                this.dataService.getTemperatureColour(
                    this.temperature)
          });
...
```

Finally, we update the **`BusinessComponent`** HTML template and add an **SVG** object in which we specify a circle of radius 40 pixels, filled with the calculated **`temperatureColour`**, and with the current temperature added in text, centred within the circle.

Note:	SVG (Scalable Vector Graphics) is an XML format for the description of 2-dimensional vector-based graphics that is supported by all major browsers. It is outside of the scope of this module, but if you would like to explore it further, links to tutorials and examples are provided in Section 14.6 of this practical.

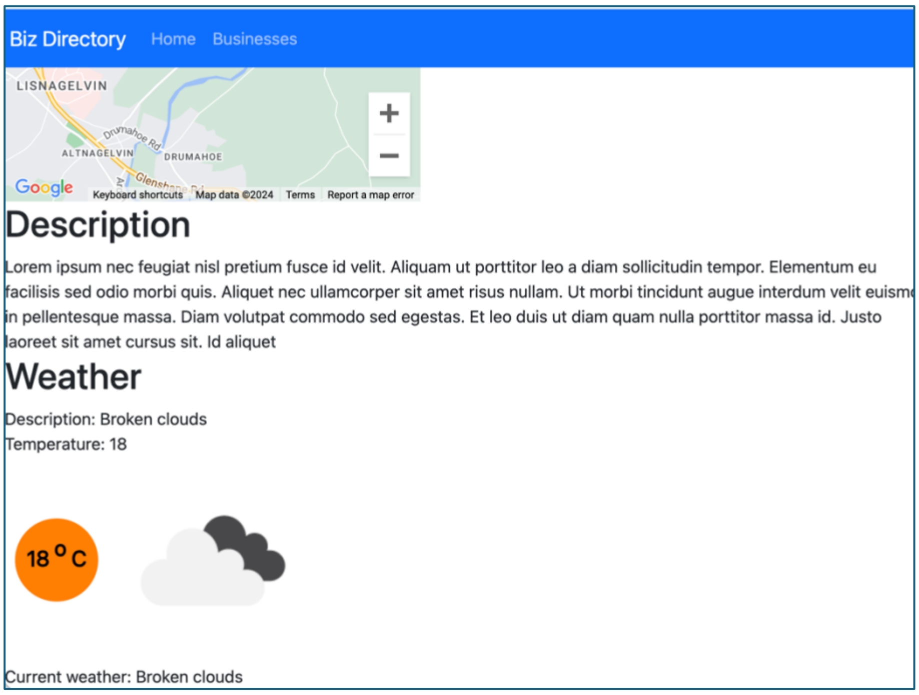
File: bizFE/src/app/business.component.html

```
...

<svg height="100" width="100">
   <g>
      <circle [style.fill]="temperatureColour"
              cx="50" cy="50" r="40">
      </circle>
      <text x="50%" y="50%" font-size="1.3em"
            text-anchor="middle" stroke="#000"
            stroke-width="1px" dy=".3em">
          {{ temperature}}
          <tspan dy="-10">o</tspan>
          <tspan dy="10"> C</tspan>
       </text>
   </g>
</svg>

<img src="{{ weatherIconURL }}"><br>
Current weather: {{ weather }}
```

Do it now! Add the new method `getTemperatureColour` to the `DataService` and update the `BusinessComponent` TypeScript file to call the method and obtain the `temperatureColour` property for the business. Then add the code shown above to the `BusinessComponent` HTML template and verify that the current temperature for the selected business is displayed on a coloured disc along with the weather icon and descriptive text as shown in Figure 14.7, below.



Description

Lorem ipsum nec feugiat nisl pretium fusce id velit. Aliquam ut porttitor leo a diam sollicitudin tempor. Elementum eu facilisis sed odio morbi quis. Aliquet nec ullamcorper sit amet risus nullam. Ut morbi tincidunt augue interdum velit euism in pellentesque massa. Diam volutpat commodo sed egestas. Et leo duis ut diam quam nulla porttitor massa id. Justo laoreet sit amet cursus sit. Id aliquet

Weather

Description: Broken clouds
Temperature: 18

Current weather: Broken clouds

Figure 14.7 Weather Information Displayed Graphically

14.5 Building a Layout

Finally, we can organise the display of the Business Component to provide an attractive layout. To differentiate from the Bootstrap Card display of the list of businesses, we remove the card from the Business Component and instead generate a layout as shown below, where we have a Bootstrap row containing a single column with the business name and star rating above a row split into two equal columns, once containing the Lorem Ipsum description and the weather report and another containing the Google Map (or a representative image if you prefer not to sign up for a Google Cloud Platform developer account).

Business name and rating	
Business information as Lorem Ipsum text Current weather report	Google Map centered on the business location

The code structure for the layout is described in the code box below, with the place to add the specific items of content indicated by comments.

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

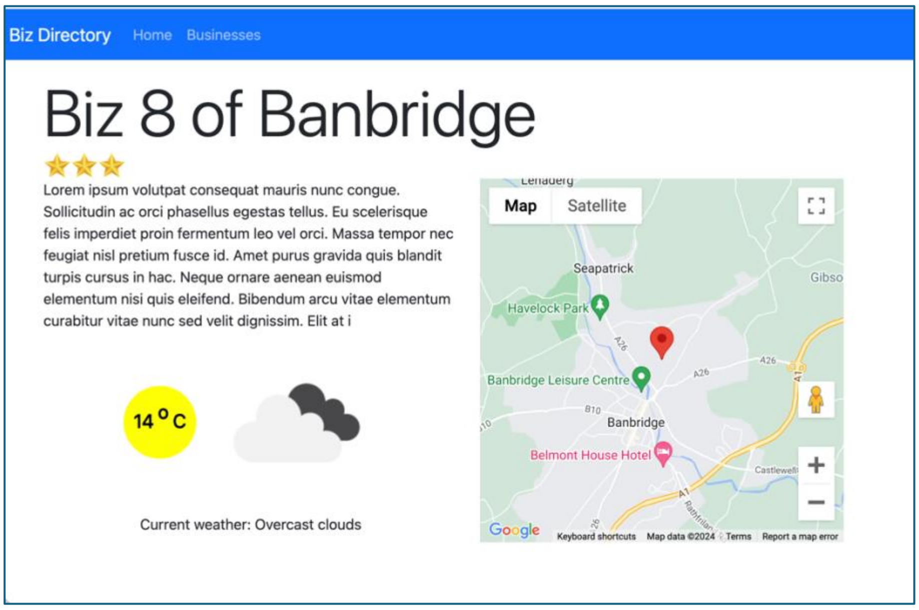
```
<div class="container"  style="margin-top: 70px">
    @for ( business of business_list; track business.name ) {

        <div class="row">
            <div class="col-sm-12">
                <div class="display-1">
                    {{ business.name }} of {{ business.town }}
                </div>
                <!-- Star rating here -->
            </div>
        </div>

        <div class="row">
            <div class="col-sm-6">
                <!-- Lorem Ipsum description here -->

                <div>
                    <p align="center">
                        <!—- Weather information here -->
                    </p>
                </div>
            </div>
            <div class="col sm-6 text-center">
                <!-- Google Map here -->
            </div>
        </div>
    }
</div>
```

Do it now!	Replace the current contents of ***business.component.html*** with the code structure shown above, inserting the code for the star rating, Lorem Ipsum descripton, weather information, and Google Map (or a representative image) at the locations indicated. Verify that your display of a single business now resembles that shown in Figure 14.8, below and that the information is updated correctly as each new business is selected from the directory.

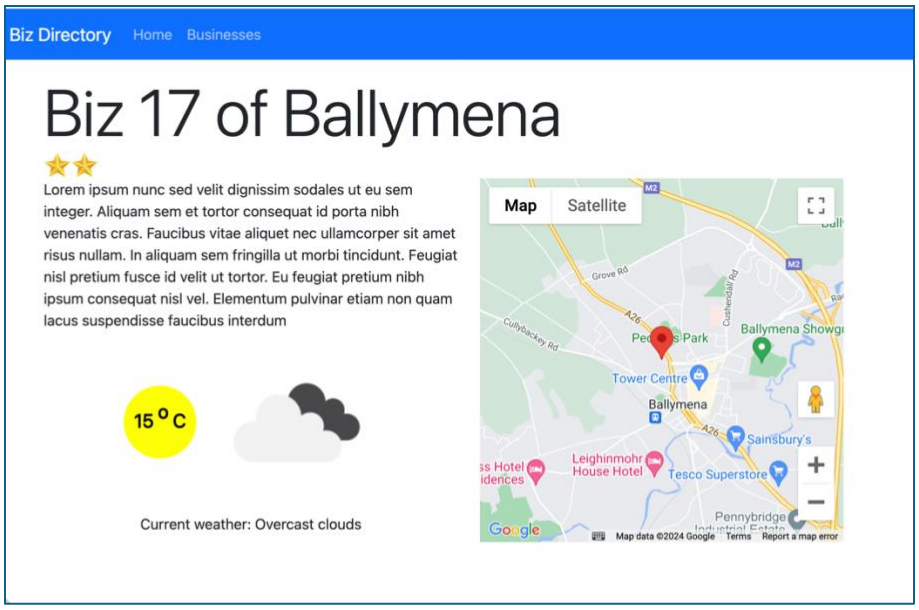

Biz Directory Home Businesses

Biz 8 of Banbridge

Lorem ipsum volutpat consequat mauris nunc congue. Sollicitudin ac orci phasellus egestas tellus. Eu scelerisque felis imperdiet proin fermentum leo vel orci. Massa tempor nec feugiat nisl pretium fusce id. Amet purus gravida quis blandit turpis cursus in hac. Neque ornare aenean euismod elementum nisi quis eleifend. Bibendum arcu vitae elementum curabitur vitae nunc sed velit dignissim. Elit at i

14 °C

Current weather: Overcast clouds


Biz Directory Home Businesses

Biz 17 of Ballymena

Lorem ipsum nunc sed velit dignissim sodales ut eu sem integer. Aliquam sem et tortor consequat id porta nibh venenatis cras. Faucibus vitae aliquet nec ullamcorper sit amet risus nullam. In aliquam sem fringilla ut morbi tincidunt. Feugiat nisl pretium fusce id velit ut tortor. Eu feugiat pretium nibh ipsum consequat nisl vel. Elementum pulvinar etiam non quam lacus suspendisse faucibus interdum

15 °C

Current weather: Overcast clouds

Figure 14.8 The Improved Business Component Template

14.6 Further Information

- https://www.techiediaries.com/angular-18-assets-folder/
  The Angular 18 Assets folder

- https://www.programiz.com/javascript/library/array/constructor
  JavaScript Array Constructor

- https://cloud.google.com/
  Google Cloud Platform

- https://developers.google.com/maps/get-started
  Getting Started with Google Maps Platform

- https://medium.com/@selsa-pingardi/integrating-google-maps-in-angular-17-66487ed2238c
  Integrating Google Maps in Angular

- https://timdeschryver.dev/blog/google-maps-as-an-angular-component
  Google Maps Component for Angular

- https://developers.google.com/maps/documentation/javascript/advanced-markers/migration
  Google Maps Advanced Markers

- https://api-ninjas.com/
  API Ninjas – Build Real Projects with Real Data

- https://api-ninjas.com/api/loremipsum
  Lorem Ipsum API

- https://openweathermap.org/current
  OpenWeatherMap Current weather API

- https://www.w3.org/TR/SVG/
  SVG Specification

- https://www.tutorialscampus.com/html5/svg-draw-circle-with-text.htm - google_vignette
  SVG Text in a Circle

- https://getbootstrap.com/docs/4.1/layout/grid/
  Bootstrap Grid System