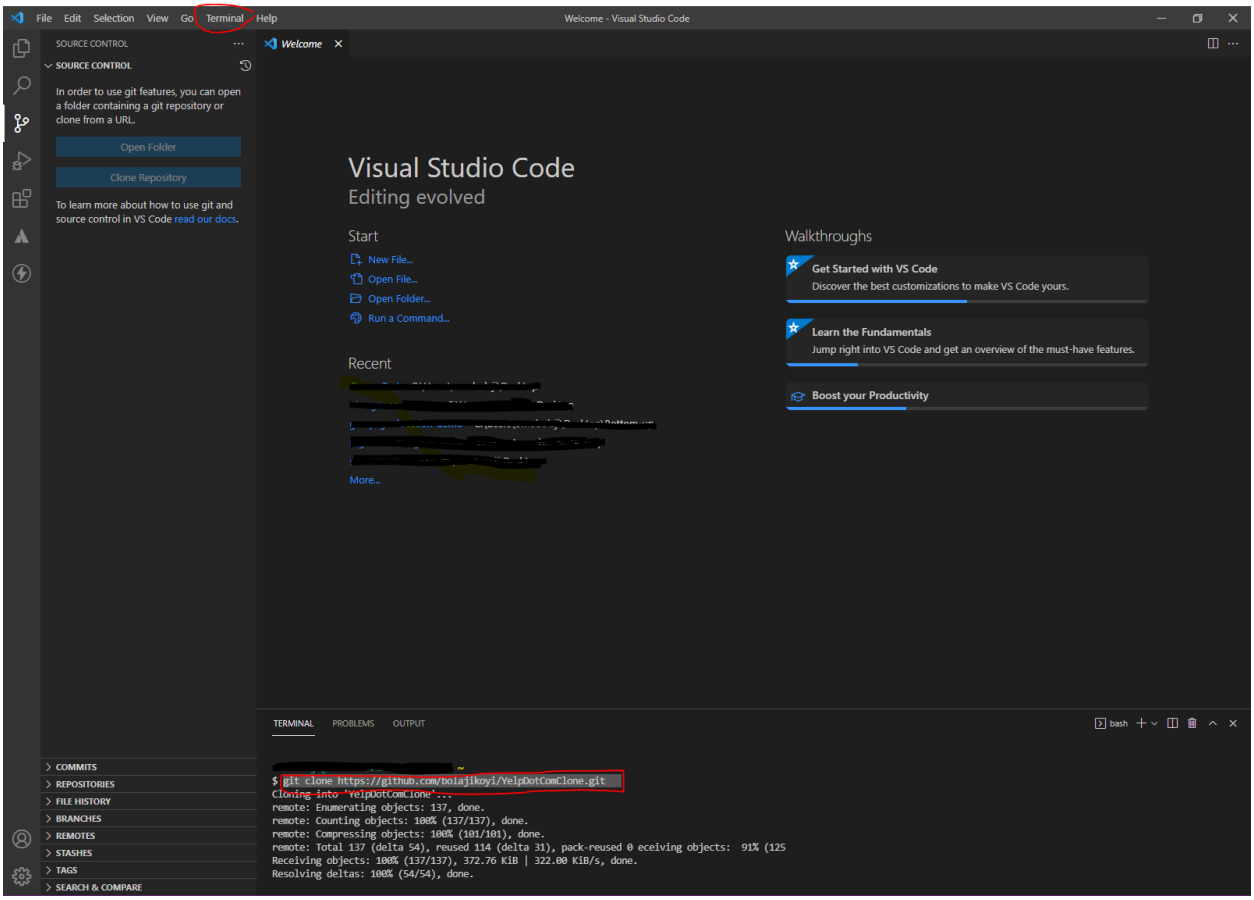
YelpClone Documentation

In this report, the steps for creating a yelp clone will the discussed. This project was done with Angular. It is best to first clone the project to have the complete project files so that it can be much easier to follow along. All the necessary details needed to understand the project and to create a similar project will be discussed.

1. Cloning the project
2. Installing Angular on your PC
3. Creating an Angular project
4. Overview of the project files
5. Creating components
    5.1. Filter component
    5.2. Footer component
    5.3. Header component
    5.4. Map component
    5.5. Search result component

Cloning the project

The Github repository link for this application is https://github.com/bolajikoyi/YelpDotComClone.git and the recommended text editor is VSCode. Copy the above link and head to your VSCode editor. In VSCode, open a new terminal and enter the command below to save the project on your local computer. git clone https://github.com/bolajikoyi/YelpDotComClone.git



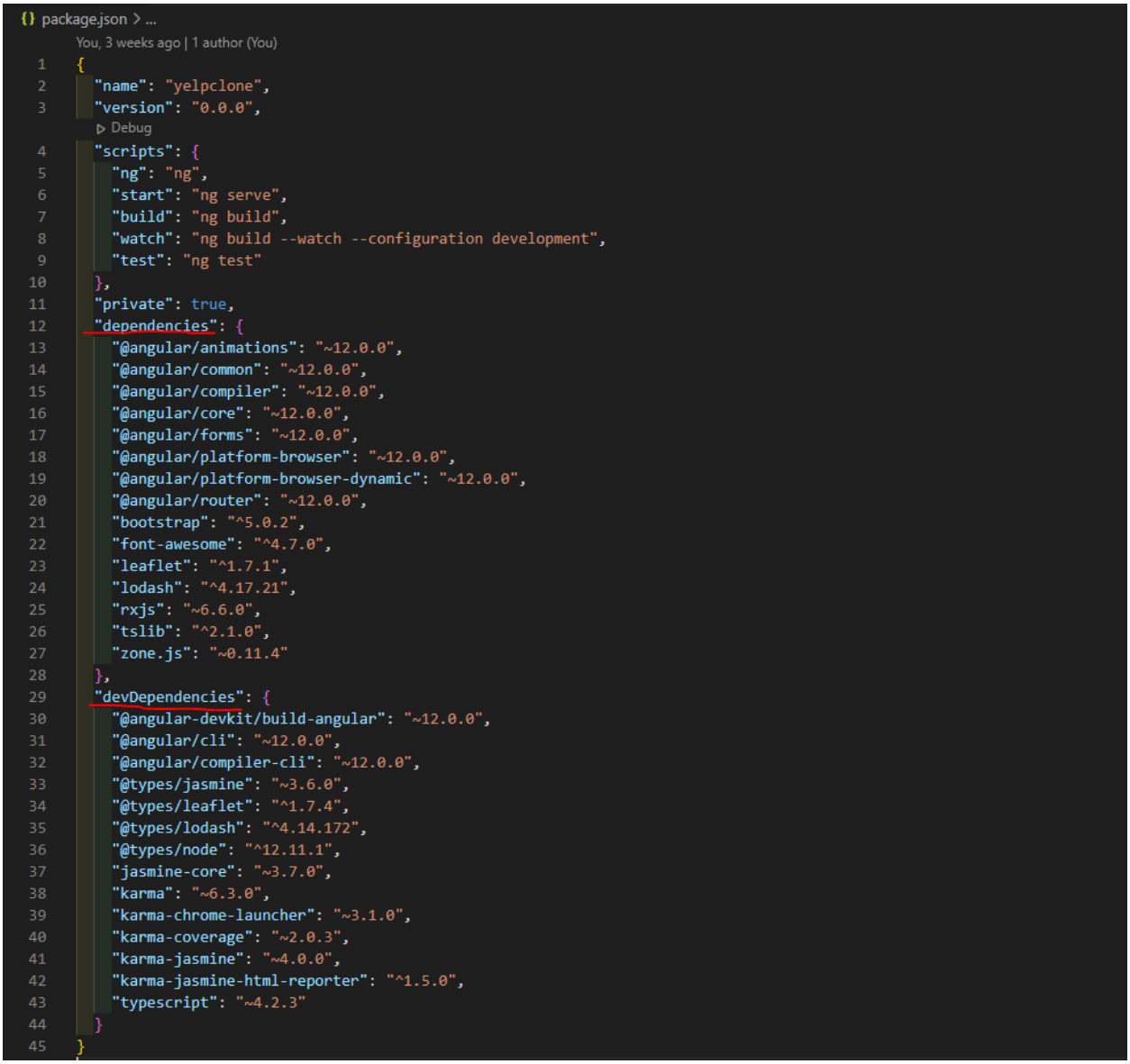
If the cloning was correctly done, then, you can have access to all the files and folders that the project contains. Having all the project files on your local computer will make it easier for you to follow the steps in this report, and you can also run the project on localhost to interact with it live on your computer. After cloning the project, navigate to the folder where it is saved, and open the project with VSCode.

Install Angular on your PC

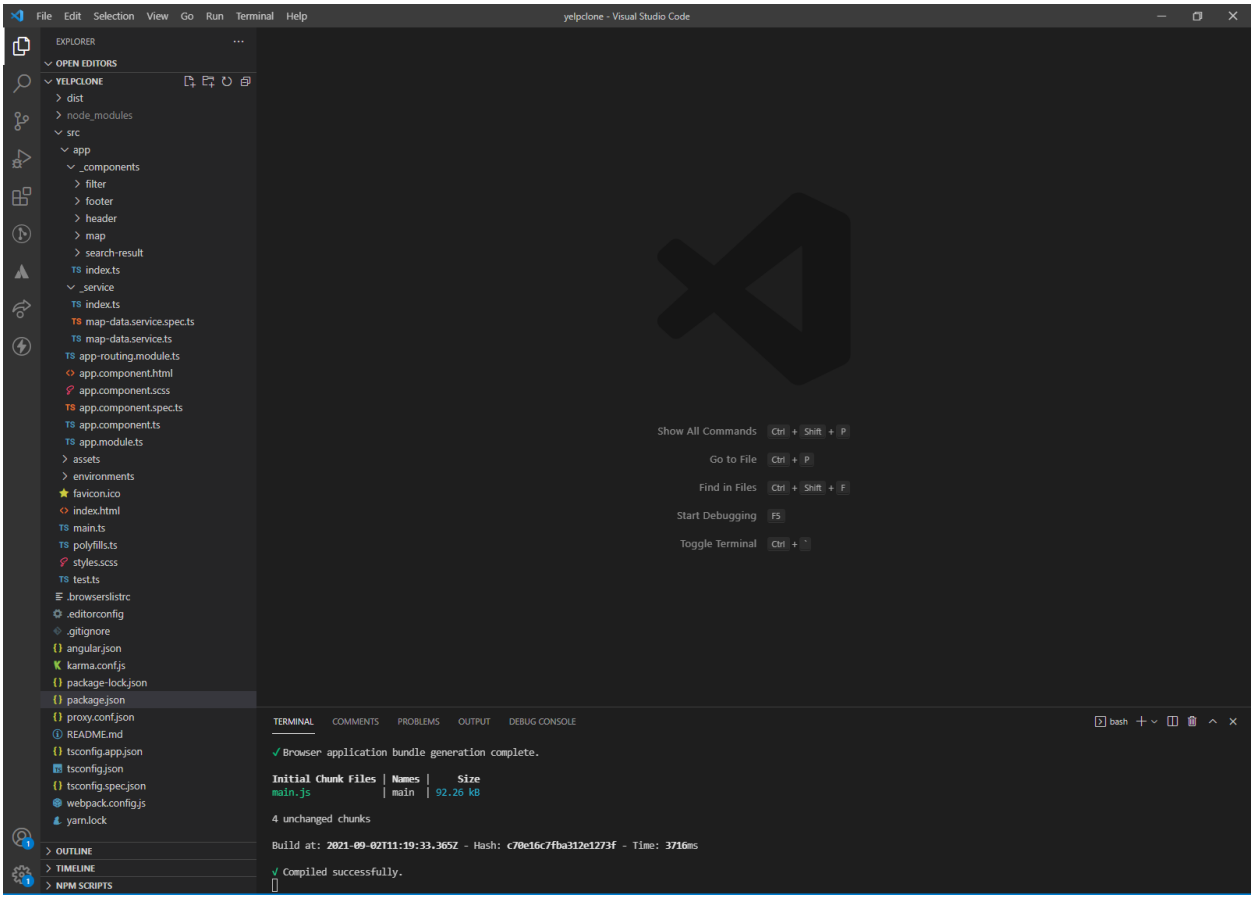
To install Angular on your local computer, you need to have node.js running on your system. If you do not already have node.js running on your computer, you can download it from https://nodejs.org/en/. After the installation, you can check the version of node.js you have running on your computer by entering the command “node -v” in the command prompt. Installing node.js allows you to use the node package manager (NPM). Different packages that are needed to develop your projects are downloadable using the node package manager. You can find numerous NPM packages on https://www.npmjs.com/. An alternative to NPM is yarn. It is beyond the scope of this report to compare NPM and yarn. Yarn packages can be found on https://yarnpkg.com/.

Angular documentation can be found at https://angular.io/. All the information necessary to use Angular can be found in the documentation. To install Angular globally on your local PC, run the following command “npm install -g @angular/cli” in the command prompt.

Now that you have Angular installed, you can open the cloned project in VSCode and open a new terminal. In the terminal, enter “npm install” to install the dependencies. Dependencies are the packages installed during the development of the project that is necessary for the project to work. The dependencies for this project can be found in the package.json file. The package.json file shown below contains the project name (line 2), the command to run the project (line 6), and the package (i.e. dependency) names and the installed versions.



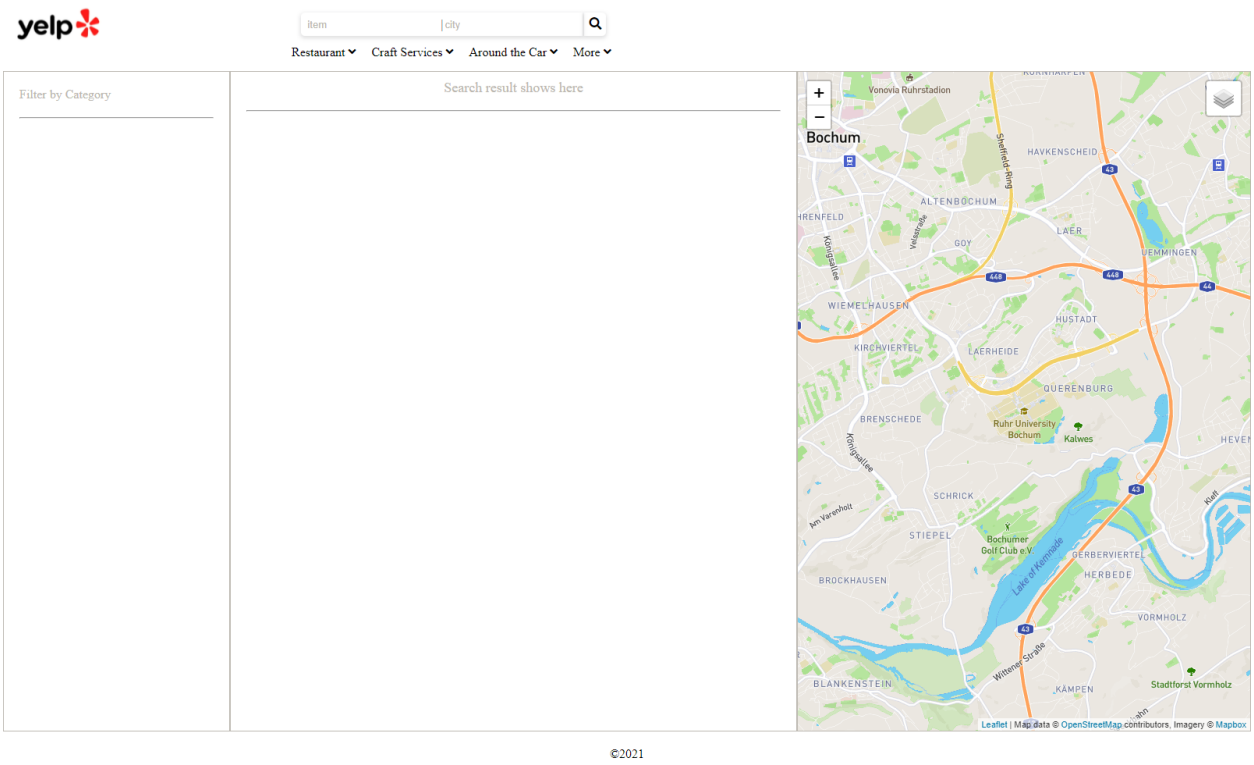
The folder structure of the project can be seen in the image below. Some of the most important files are package.json, and angular.json. Most of what we will be doing will be in the “src” (i.e. source) folder.



The src folder contains the app folder, assets folder, environments folder, favicon.ico, index.html, main.ts, ploy fills.ts, styles.scss and test.ts. The components that we will be working with will be created in the app folder. Depending on the environment we are working in, some variables might only be needed in the development and not needed when working in the production environment. These variables that are specific to the different environments and the variables that we do not want to be exposed to the public can be saved in the environments files. Some files and folders that are used across the application such as icons, images etc., are saved in the assets folder. In case you are new to ".scss" and ".ts" files, they are supersets of ".css" and ".js" files respectively. TS stands for TypeScript and JS stands for JavaScript. You can write ".css" syntax in ".scss" files and also, you can write ".js" syntax in ".ts" files. Style.scss stands as the global style sheet, and the stylings that operate at the global level of the project can be written here. The index.html file serves as the entry point for the application. You do not need to do much in the index.html file. If you want to add CDN links to the application such as fontawesome or bootstrap, it can be done in the index.html file. favicon.ico is the icon that shows in the tab of the web browser. The icon can be replaced with a preferred icon.

Now that we have an idea about what some of the project files and folders are used for, the next step is to run the application and serve it on localhost. In the VSCode terminal, you can use either “ng serve” or “npm start” commands to run the application.

So far, we have cloned the application, installed Angular globally, installed the dependencies, and the last thing we did was to serve the application on localhost. If everything works fine, we should have the image below. The application is served on port 4200. In case you want to serve the application on a different port, you can include the port number in the ng serve command i.e. “ng serve --port 4300” if we want to serve the application on port 4300. The choice of the port number to use is up to us but 4200 is the default port number.



Creating an Angular project

To create an Angular project, open a new window in VSCode, click on Terminal and select New Terminal from the drop-down menu. In the terminal, navigate to your desired directory. For example, if you want to create the project on the desktop, you can navigate to the desktop by typing “cd desktop” in the terminal as shown below.

Now, you can create an Angular project with the name yelpclone by typing “ng new yelpclone” in the terminal.

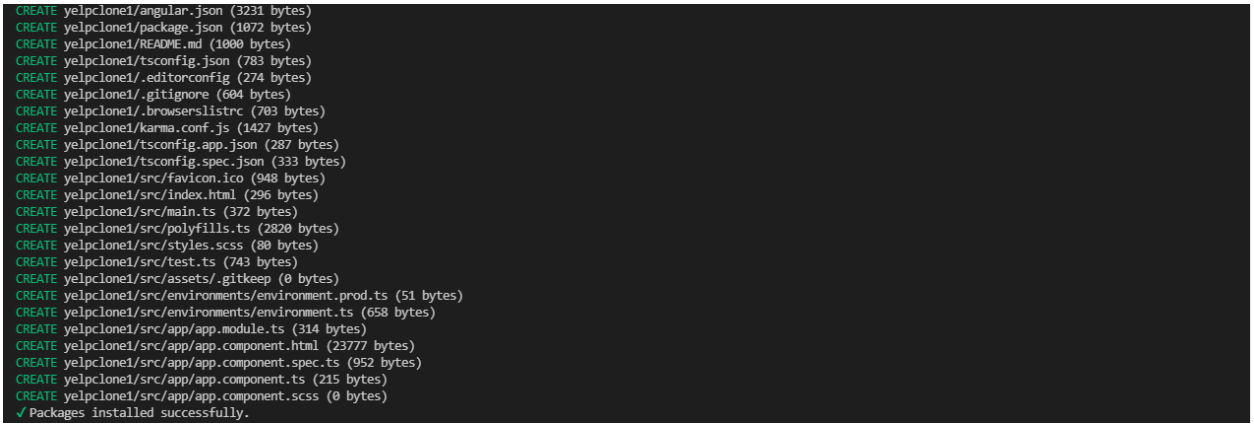
You will be prompted to answer some questions and make some choices as below:

- ? Would you like to add Angular routing? (Y/N)
- ? Which stylesheet format would you like to use? (Use arrow keys)
  > CSS
  SCSS [https://sass-lang.com/documentation/syntax#scss]
  Sass [https://sass-lang.com/documentation/syntax#the-indented-syntax]
  Less [http://lesscss.org]

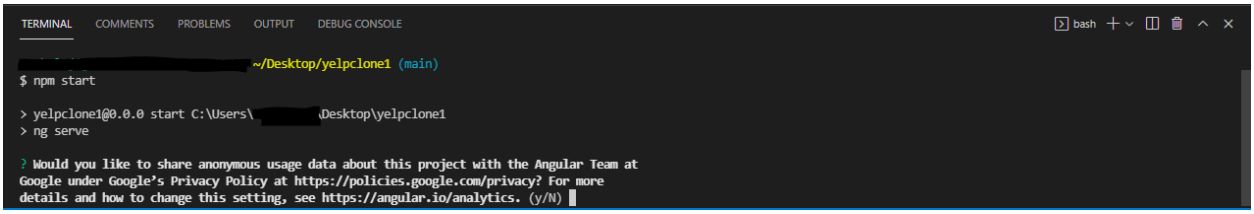
Depending on if you want to route or not, you will need to answer either yes or no, but for this project, we are not routing, so, it is enough to answer no. For the stylesheet format, you can use the arrow keys to choose any of the formats in the list and press “Enter”. We are choosing SCSS for this project.

After answering the above questions, the creation of the project will start, and you will see something similar to what is in the image below.

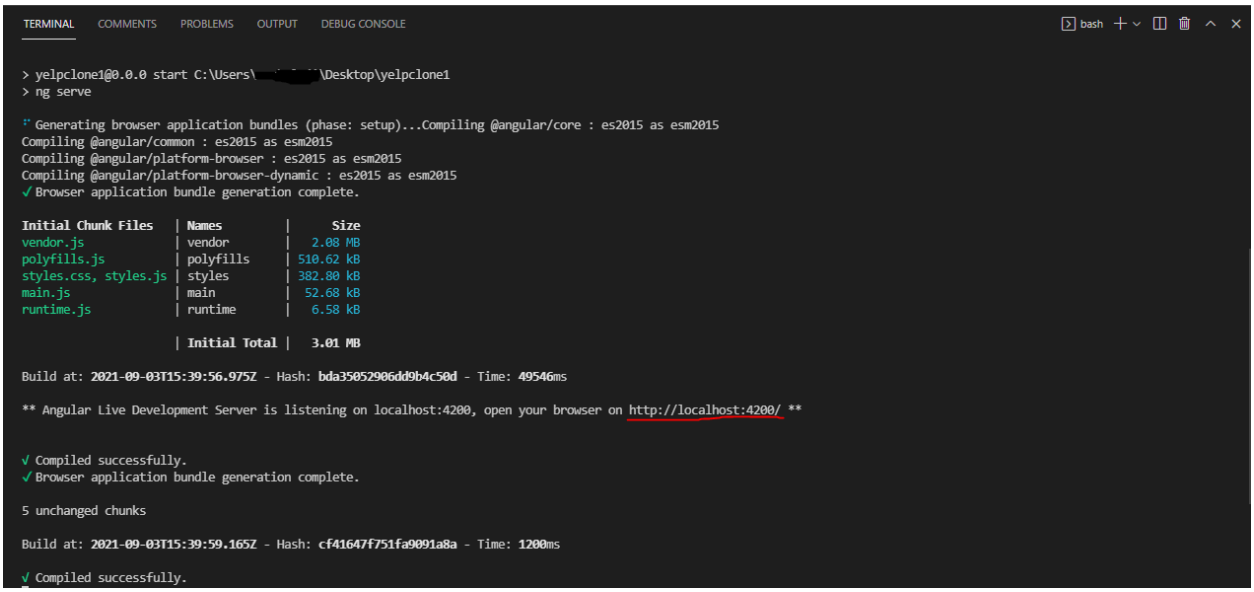
When the project creation is completed, the terminal looks similar to the below image.



Now our project is created. To open the newly created project, from VSCode, click “Open Folder” and navigate to the directory where the project is saved, it is on the desktop in our case. After opening the project in VSCode, go to terminal, then new terminal. In the new terminal, enter “ng serve” or “npm start” to serve the project on localhost. You will be asked if you want to share anonymous usage data about the project with the Angular team at Google. The response to the question is up to you. You can answer either yes or no.

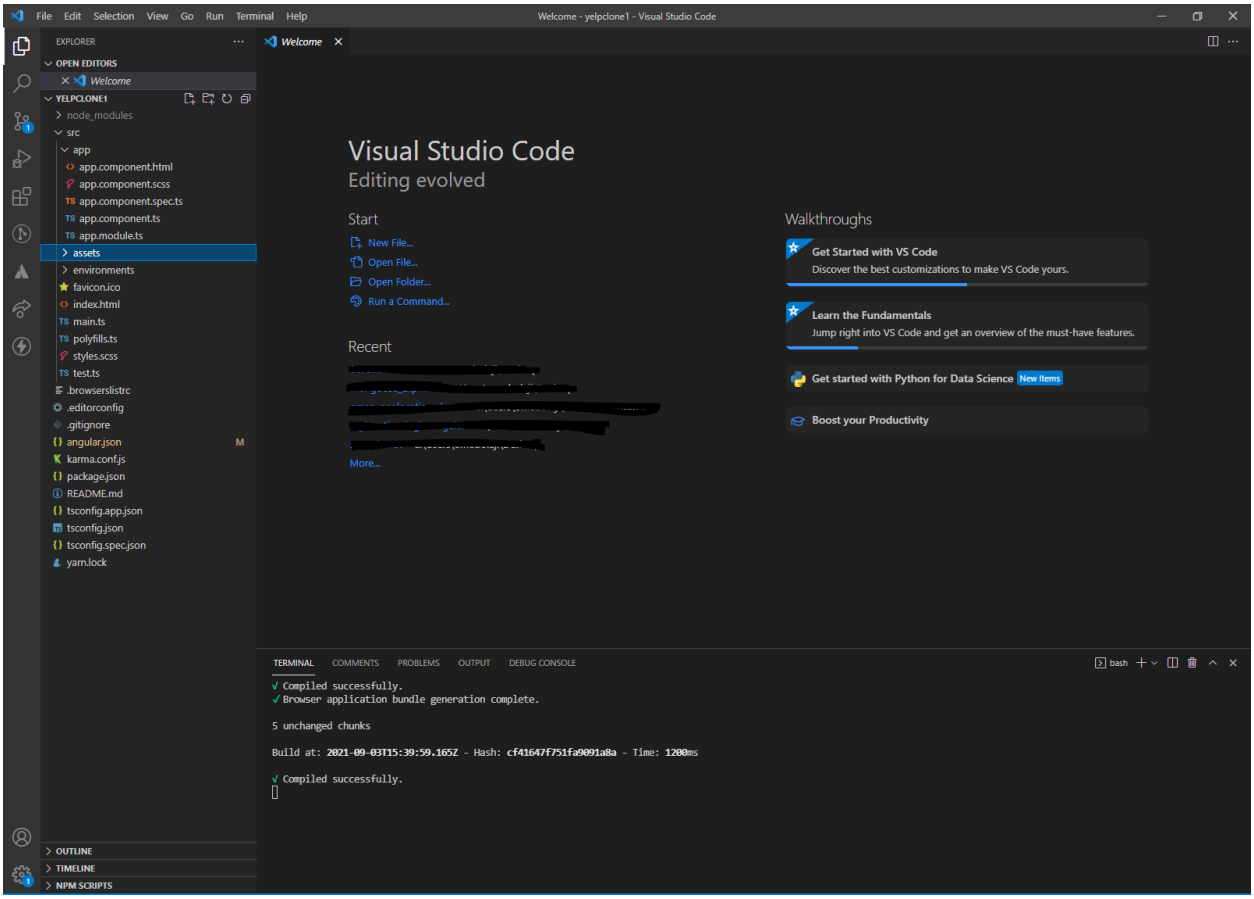


After compiling, you will find the localhost URL to access the application in the browser like in the image below. Here, the application runs on http://localhost:4200/.



Overview of the project files

We have discussed earlier some of the important files that are generated when an Angular application is created. From the below image, these files and folders can be seen. Now, the focus is going to be more on the files in the “app” folder. The “app” folder is one of the folders in the “src” folder.

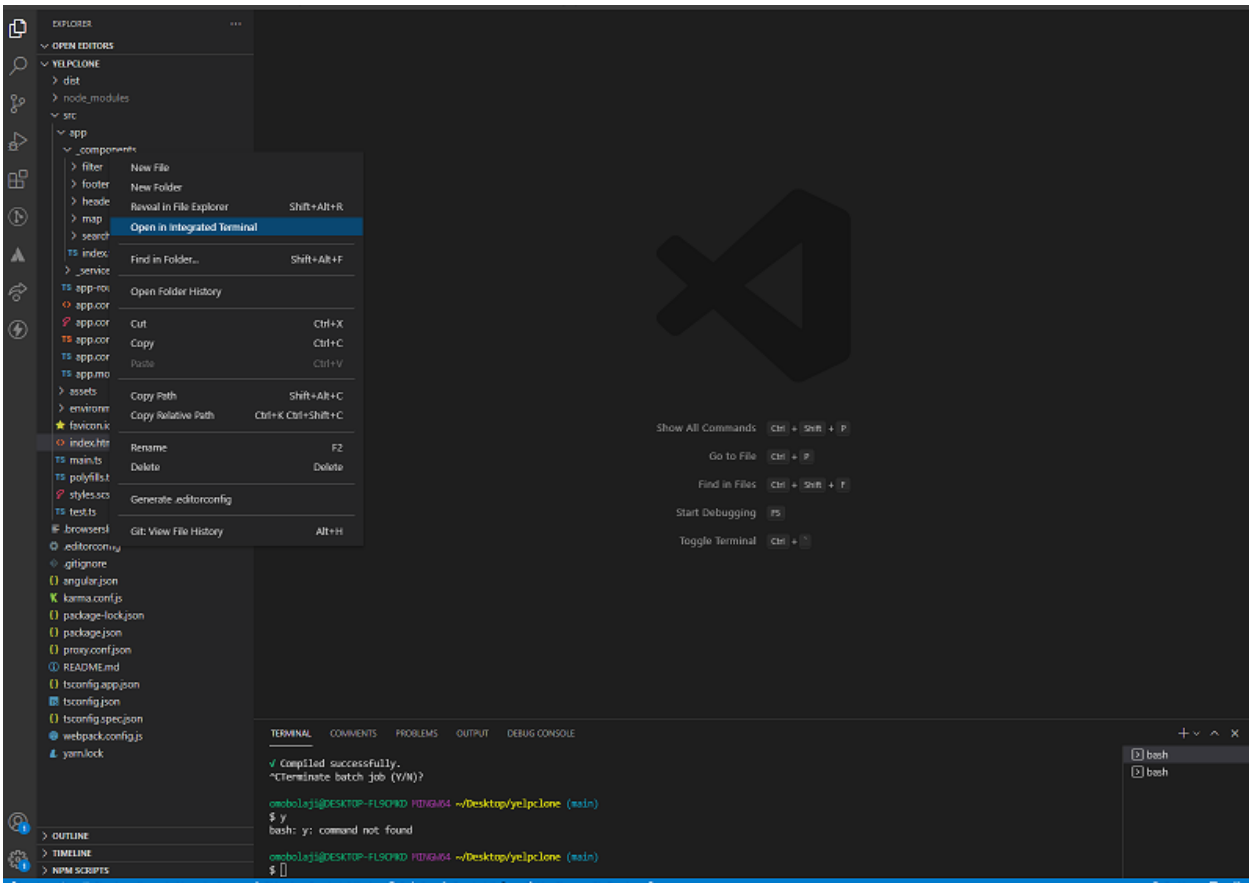


The app folder contains the app component, the app module, and the app-routing module (optional). A component by default contains four files; the HTML file, the scss file, the spec file, and the TypeScript file. The app component serves as the root component of the application. The app module serves to bind all the components in the application together. When a component is created, it will be registered in the app module of the application so that the components can communicate with each other and with the root component (i.e. app component).

Creating components

To create a component in Angular, you enter the following command in the VSCode terminal “ng generate component <component name>”. We will be creating 5 components in this application, and they are filter component, footer component, header component, map component, and search-result component. For the organization of the component folders, we will keep all the components in a folder named _components. The name of the folder is arbitrary and is up to you to choose a suitable name for the folder.

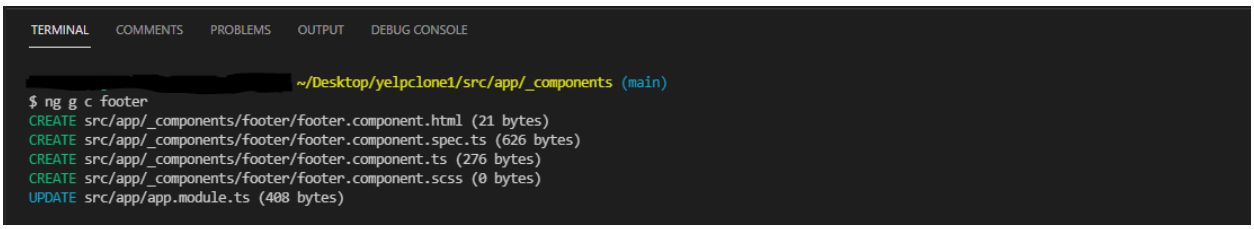
Firstly, we will create the _components folder in the app folder. To create the components in the _components folder, right-click on the _components folder and choose “open in integrated terminal” as shown in the image below.



In the terminal, you can create each of the components by entering the commands one after the other.

- ng generate component filter
- ng generate component footer
- ng generate component header
- ng generate component map
- ng generate component search-result

If for example, we try to create a footer component in the _components folder, after running the command in the terminal, we will get a response in the terminal like what is in the image below.



According to the above image, we can see that the HTML, spec, ts, and scss files have been created and that the app.module.ts file has been updated. For all the other components, we will get a similar response in the terminal after they have been created. Now we will go through all the files in each of the components one after the other. That app.module.ts is updated means that the newly generated component has been imported in the app.module.ts file.

Filter Component

A knowledge of some of the important Angular concepts such as input decorator, output decorator, observables, lifecycle hooks, ngIf, and ngFor directives, etc will make it easier to follow along with the creation of this project. If you are just seeing some of these concepts for the first time, do not panic, I will do my best to explain them basically and in the simplest form. See filter.component.scss for the styling. In this application, variable names preceded by a dollar sign "$" are used to represent observables.

Filter.component.html

In the HTML file, we have 2 divs, one for the title, and the other for outputting the filter texts. type$ is an observable that contains an array of the filter texts. type$ is declared in the TypeScript file. *ngFor is like a "for loop" that can be used to iterate over an array in the HTML file. In line 4, we are iterating over type$, and saving each of the values as types. To execute an observable, you have to subscribe to it. Subscription can be done either in the TypeScript file or in the HTML file. To subscribe to an observable in the HTML file, you add "| async" pronounced as "pipe async" after the observable. We also have "index as i" in the *ngFor directive to get the index of the elements of the type$ array and save them as i. In line 6, we are concatenating the value of i with the string "filterbutton" and passing it as an argument in the click event. The purpose of this is to create a unique selector for each of the filter items. In line 7, we assigned an id selector "#filterButton" which gives us access to all the properties of the div element. Because of the id selector, we can have access to the id by calling "filterButton.id". In line 8, we added a click event to the div element which calls the categoryFilter() method in the filter.component.ts file. The categoryFilter method accepts two arguments; the elements of the type$ array saved as "types", and "filterButton.id". In line 9, we are using a conditional (ternary) operator to assign a class to the div element. If selected is equal to filterButton.id, assign the class "buttonActive", otherwise, do not assign any class. In line 11, we have double curly braces (otherwise known as text interpolation) that contain the text "types". This is going to output the list of strings in the type$ array. Text interpolation can be used to evaluate an expression in the template, for example {{ 1 + 1 }} will give 2.

Filter.component.ts

In lines 1 and 2, we are importing different classes and functionalities into this component that are needed for our implementation.

In line 5 is the selector for this component. We may take this entire component and place it inside another component. In this application, we have taken this component and placed it in the app component. To do that, we will need to make use of the component's selector. If we are placing this component in the app component for example, we will need to place its selector "<app-filter></app-filter>" in the HTML file of the app component (i.e. app.component.html), and by doing this, an instance of the filter component will be created in the app component. If you find anything confusing at the moment, as we go on in the implementation, I believe they will become clearer.

In lines 6 and 7, the templateUrl and styleUrls are the references to the component's HTML file and stylesheet respectively.

The input decorator is used to send data from the parent component to the child component and the output decorator is used to send data from the child component to the parent component. The parent component in this case is the app component, and the child component is the filter component.

In line 10, we are using the input decorator to receive the category variable from the parent component (app component), and the value of category is set in "val" which has the type "any". In line 11, the value that is received from the parent component is stored in the observable type$ of type BehaviorSubject, using the "next" method. "this" is used to refer to the class where type$ is declared. Therefore, this.type$.next(val) means that we are referring to types$ declared inside the FilterComponent class, and we are saving "val" in type$ using the "next" method.

In line 15, we are creating a new instance of ReplaySubject of an array of strings and giving it the name type$. In line 16, variable “selected” is declared and initialized as an empty string. In line 13, we are using the output decorator to emit a value from the child component to the parent component. A new instance of an event emitter is created and is named filterText.

From line 22 to 30 is the categoryFilter method that accepts two arguments (event and id). In line 23, we are checking if “selected” is not the same as id. In line 24, if selected is not the same as id, we are assigning id to selected, and in line 25, we are sending “event” through the output decorator to the parent component using the emit method. In lines 27 and 28, selected and filterText are set to empty strings (“ “).

Footer Component

Footer.component.html

```
1  <div class="footer"><span>&copy;</span>{{ fullYear$ | async }}</div>
```

“©” is the HTML copyright symbol. fullYear$ is an observable, and we are subscribing to it in the template by using “pipe async”.

Footer.component.ts

```
1  import { Component, OnInit } from '@angular/core';
2  import { ReplaySubject } from 'rxjs';
3
4  @Component({
5    selector: 'app-footer',
6    templateUrl: './footer.component.html',
7    styleUrls: ['./footer.component.scss'],
8  })
9  export class FooterComponent implements OnInit {
10   fullYear$ = new ReplaySubject<number>();
11
12   constructor() {}
13
14   ngOnInit(): void {
15     this.fullYear$.next(new Date().getFullYear());
16   }
17 }
```

Imports are done in lines 1 and 2. In line 5 is the component selector, and lines 6 and 7 are the template and stylesheet reference respectively. In line 10, we created an observable fullYear$ of type ReplaySubject. In line 14 is one of the Angular lifecycle hooks “ngOnInit”. ngOnInit is called once after the component has been initialized. Operations that are needed to be executed after the component class is initialized can be placed inside ngOnInit. In line 15, we are obtaining the current year and saving the value in the observable fullYear$.

Header Component

Header.component.html

```
1  <div class="header-items">
2    <img class="yelp-logo" src="assets/image/yelp-logo.jpg" alt="" />
3
4    <div class="search-box">
5      <div>
6        <input
7          class=""
8          type="text"
9          name="search-item"
10         id="search-item"
11         placeholder="item"
12         [(ngModel)]="searchItem"
13         (keyup)="searchType()"
14       />
15     </div>
16     |
17     <div>
18       <input
19         class=""
20         type="text"
21         name="search-place"
22         id="search-place"
23         placeholder="city"
24         [(ngModel)]="searchCity"
25         (keyup)="searchPlace()"
26       />
27     </div>
28   </div>
29   <div class="search-button" (click)="searchButton()">
30     <i class="fas fa-search"></i>
31   </div>
32 </div>
```

In line 2, we are displaying the yelp logo using an “img” tag. The “src” attribute contains the directory where the image is saved. After the yelp logo, we have two input fields of type “text” that are used for search-items and search-place respectively. The item to be searched for is entered into the input field with the placeholder “item” while the location of what is searched for is entered into the input field with placeholder “city”. In line 12, we used “[(ngModel)]” and we set its value as “searchItem”. ngModel allows us to set the value of searchItem both in the template and in the TypeScript file. If the value of searchItem is modified in the template(i.e. HTML file), it is automatically updated in the TypeScript file, alternatively, if the value is set or modified in the TypeScript file, it is also updated in the template. In line 13, we are using a keyup event to call the searchType() method in the TypeScript file. With the keyup event, every time a key is pressed on the keyboard the searchType() method is executed. In lines 24 and 25, what we have is similar to what we have in lines 12 and 13. In line 29, we have a div that functions as a button, and it listens for a click event to execute the searchButton() method in the TypeScript file. In line 30, we placed a fontawesome search icon in the button.

Header.component.ts

```typescript
import {
  Component,
  OnInit,
  Output,
  EventEmitter,
  ViewChild,
  Input,
} from '@angular/core';
import { ReplaySubject } from 'rxjs';
import { debounceTime, map } from 'rxjs/operators';

You, a month ago | 1 author (You)
@Component({
  selector: 'app-header',
  templateUrl: './header.component.html',
  styleUrls: ['./header.component.scss'],
})
export class HeaderComponent implements OnInit {
  @Output() itemType: EventEmitter<string> = new EventEmitter();
  @Output() place: EventEmitter<string> = new EventEmitter();
  @Input() set menuList(val: any) {
    this.searchItem = val;
  }

  itemType$ = new ReplaySubject();
  place$ = new ReplaySubject();

  searchItem = '';
  searchCity = '';

  constructor() {}

  ngOnInit(): void {
    //we want to subscribe to the observable only once and delay for 1 second when triggering keyup.
    this.itemType$.pipe(debounceTime(1000)).subscribe((res: any) => {
      return this.itemType.emit(res);
    });
    this.place$.pipe(debounceTime(1000)).subscribe((res: any) => {
      return this.place.emit(res);
    });
  }

  searchType() {
    this.itemType$.next(this.searchItem);
  }
  searchPlace() {
    this.place$.next(this.searchCity);
  }
  searchButton() {
    this.itemType$.next(this.searchItem);
    this.place$.next(this.searchCity);
  }
}
```

We have output decorators in lines 18 and 19. Like it was previously explained, an output decorator is used to send data from the child component to the parent component, while an input decorator is used to send data from the parent component to the child component. We have an input decorator in line 20 that accepts “menuList” from the parent component. The value of menuList is saved in the variable “val”, and in line 21, val is saved in searchItem. itemType$ and place$ are observables of type ReplaySubject. In lines 27 and 28 searchItem and searchCity are initialized as empty strings. From line 32 to 40 is ngOninit lifecycle. In lines 34 to 36, and 37 to 39, we subscribe to the observables itemType$ and place$ once and delay for one second using “debounceTime” when the keyup event is called. debounceTime delays the values emitted by a source for a given due time, in this case, one second. In line 43, when searchType method is called, the value of searchItem is saved in itemType$ observable. In line 46, when the searchPlace() method is called, the value of searchCity is saved in place$ observable.

Map Component

Map.component.html

```
<div id="map"></div>
```

In the HTML, we only have one div with a unique identifier (id = “map”). The div element is selected in the stylesheet, and its height is set to 100% as shown in the image below.

Map.component.scss

```
#map {
  height: 100%;
}
```

Angular.json

After installing leaflet, and setting its height to 100% in the stylesheet, we need to modify the angular.json file as shown in the image below.

```
"assets": [
  "src/favicon.ico",
  "src/assets",
  {
    "glob": "**/*",
    "input": "node_modules/leaflet/dist/images/",
    "output": "./assets/"
  }
],
"styles": [
  "./node_modules/leaflet/dist/leaflet.css",
  "src/styles.scss"
],
"scripts": []
```

Map.component.ts

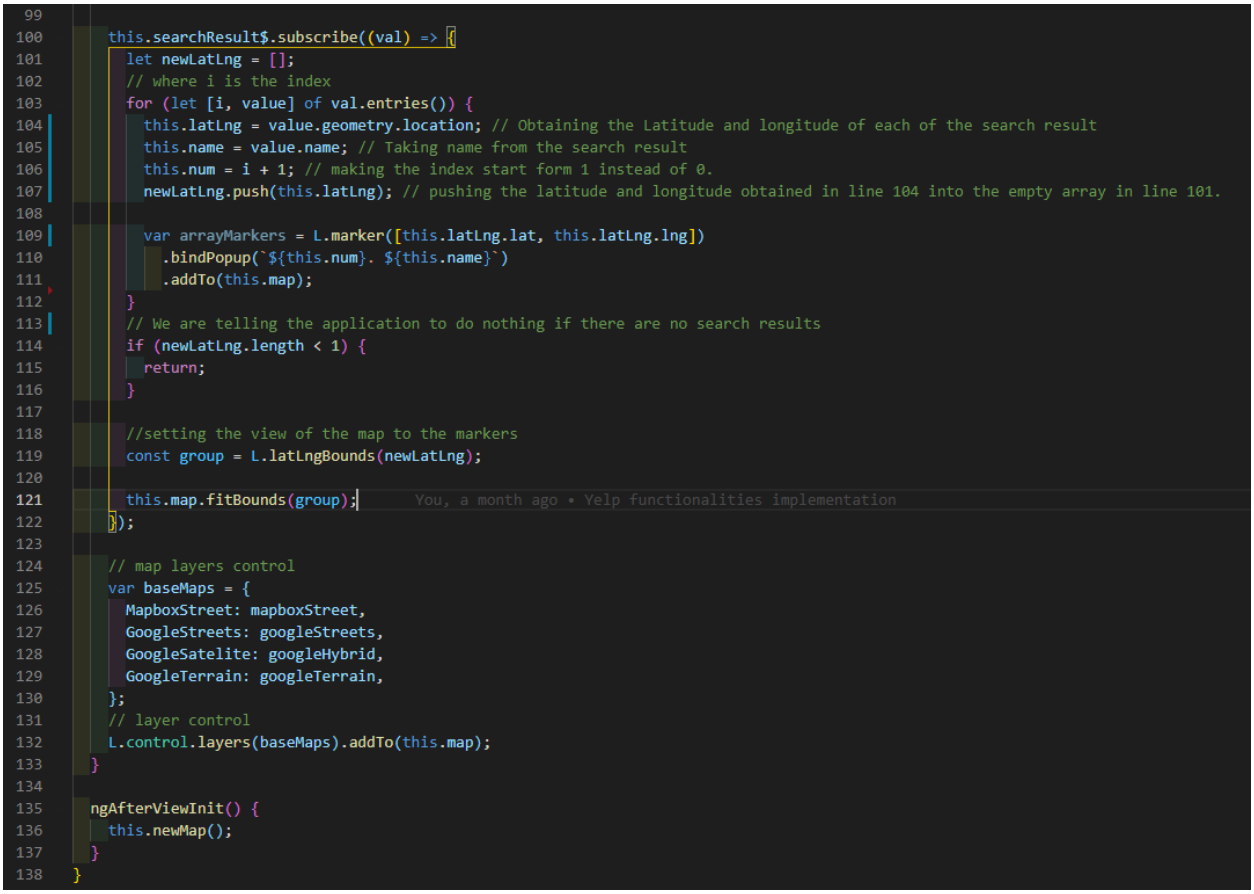
```
1  import { Component, OnInit, AfterViewInit, Input } from '@angular/core';
2  import * as L from 'leaflet';
3  import { icon, Marker } from 'leaflet';
4  import { ReplaySubject } from 'rxjs';
5  import { environment } from 'src/environments/environment';
   You, a minute ago | 1 author (You)
6  @Component({
7    selector: 'app-map',
8    templateUrl: './map.component.html',
9    styleUrls: ['./map.component.scss'],
10 })
11 export class MapComponent implements OnInit, AfterViewInit {
12   @Input() set searchResult(val: any) {
13     this.searchResult$.next(val ? val.results : []);
14   }
15   map!: L.Map;
16   latitude: number = 51.4457;
17   longitude: number = 7.2616;
18   latLng: any;
19   num!: number;
20   name!: string;
21   search!: any;
22   items: any;
23   searchResult$ = new ReplaySubject<any[]>();
24   constructor() {}
25
26   ngOnInit(): void {}
27
28   newMap() {
29     this.map = L.map('map', {
30       center: [this.latitude, this.longitude],
31       zoom: 13,
32     });
33
34     // Setting for using the default marker icon
35     const iconRetinaUrl = 'assets/marker-icon-2x.png';
36     const iconUrl = 'assets/marker-icon.png';
37     const shadowUrl = 'assets/marker-shadow.png';
38     const iconDefault = icon({
39       iconRetinaUrl,
40       iconUrl,
41       shadowUrl,
42       iconSize: [25, 41],
43       iconAnchor: [12, 41],
44       popupAnchor: [1, -34],
45       tooltipAnchor: [16, -28],
46       shadowSize: [41, 41],
47     });
48     Marker.prototype.options.icon = iconDefault; // setting iconDefault as the default marker icon
```

Leaflet is used to display the map in the application. Leaflet is the leading open-source JavaScript library for mobile-friendly interactive maps. To use leaflet in our Angular application, we need to install it in the VSCode terminal. The command for installing leaflet is “npm install leaflet”. After the installation of leaflet, it was imported in line 2. In line 3, we imported icon and Marker from leaflet into the TypeScript file. In line 12, we have an input decorator which allows us to receive “searchResult” from the parent component (i.e app component). The value of searchResult is saved in “val”. The searchResult$ in line 13 is an observable of type ReplaySubject. In line 13, we are checking if “val” is truthy (i.e. if it is true), and if it is, we save “val.results” in the observable, otherwise, we save an empty array in the observable. In line 15, we have a variable map of type L.Map. Since the variable map is not initialized, the “!” that precedes the variable name allows us to declare the variable without initializing it. In lines 16 and 17, latitude and longitude are of type number, and their initial values are 51.4457 and 7.2616 respectively. The same pattern follows for the other variables. In line 22, we create a method newMap() and we call it in the body of ngAfterViewInit lifecycle in line 139. In lines 29 to 32, the map was initialized, and its view was set to the initial latitude and longitude values. From lines 34 to 47, we have the settings for the marker icon, and

in line 48, we assigned the created icon to be used as the default marker icon. The marker icon images are saved in the asset folder.

```
49
50    // Using click event to get Latitude and Longitude values from the map
51    const popup = L.popup();
52    const onMapClick = (e: any) => {
53      popup.setLatLng(e.latlng).setContent(`${e.latlng}`).openOn(this.map);
54    };
55    this.map.on('click', onMapClick); // leaflet click event
56
57    // tileLayers
58    // mapbox street tile
59    var mapboxStreet = L.tileLayer(
60      'https://api.mapbox.com/styles/v1/{id}/tiles/{z}/{x}/{y}?access_token={accessToken}',
61      {
62        attribution:
63          'Map data &copy; <a href="https://www.openstreetmap.org/copyright">OpenStreetMap</a> contributors, Imagery © <a href="https://www
64        maxZoom: 18,
65        id: 'mapbox/streets-v11',
66        tileSize: 512,
67        zoomOffset: -1,
68        accessToken: environment.accessToken,
69      }      You, a month ago • Yelp functionalities implementation
70    );
71    mapboxStreet.addTo(this.map);
72
73    //street tile
74    var googleStreets = L.tileLayer(
75      'http://{s}.google.com/vt/lyrs=m&x={x}&y={y}&z={z}',
76      {
77        maxZoom: 20,
78        subdomains: ['mt0', 'mt1', 'mt2', 'mt3'],
79      }
80    );
81
82    //Hybrid tile
83    var googleHybrid = L.tileLayer(
84      'http://{s}.google.com/vt/lyrs=s,h&x={x}&y={y}&z={z}',
85      {
86        maxZoom: 20,
87        subdomains: ['mt0', 'mt1', 'mt2', 'mt3'],
88      }
89    );
90
91    //Terrain tile
92    var googleTerrain = L.tileLayer(
93      'http://{s}.google.com/vt/lyrs=p&x={x}&y={y}&z={z}',
94      {
95        maxZoom: 20,
96        subdomains: ['mt0', 'mt1', 'mt2', 'mt3'],
97      }
98    );
```

In line 51, we are instantiating a popup object. From lines 52 to 54, we have a method called onMapClick(). In line 53, we are setting the latitude and the longitude of the popup object that we created in line 51, and in line 55, we are binding the onMapClick method to a click event. So, when any position is clicked on the map, there is a popup that shows the latitude and the longitude of that position on the map. From lines 57 to 98, we are adding the tile layers. We made use of 4 tiles in this application; mapboxStreet, googleStreets, googleHybrid, and googleTerrain. To use a mapbox tile, you need to register on their website to receive an access token. The access token used in the mapbox tile was saved in the environments file and it is referenced in the TypeScript file. We imported the environments file in line 5. In line 71, we are adding the mapStreet tile to the leaflet map as the default tile.



When a search is performed in the parent component (app component), the search result is sent to the map component using an input decorator (@Input()) and is saved in searchResult$ observable. In line 100, we are taking the values saved in searcResult$ by subscribing to it. The content of the searchResult$ observable is saved in “val” (val is an arbitrary variable name, you can choose to name it whatever you want). In line 103, we are using a “for loop” to iterate over “val”. The individual content of val is saved in the variable “value”, while “i” is the index. We used “val.entries()” in the for loop so that we can get the index of the content of the array.

In line 109, we are setting the marker position for each of the search results, and we are binding a popup to them such that when a marker is clicked on the map, it shows the index of the search result and the name. In line 111, we are adding the popup to the map. In line 119, we are setting the map view to contain the group of markers so that we can see all the positions on the map at once using fitBounds() method. In line 125, we created an object baseMaps that contains all the tile layers on the map. In line 132, we are adding the tile layers to the map. In line 135, we have ngAfterViewInit lifecycle. In line 136, we are executing newMap() method inside the ngAfterViewInit lifecycle. ngAfterViewInit() is called after a component’s view, and its children’s are created.

Search Result Component

Search-result.component.html

```
1  <div class="search-result">{{ search }}</div>
2  <hr />
3  <div
4    class="search-items"
5    *ngFor="let results of searchResult$ | async; index as i"
6  >
7    <div>
8      <img
9        *ngIf="!!results.photos"
10       class="photo"
11       [src]="
12         'https://maps.googleapis.com/maps/api/place/photo?maxwidth=400&photoreference=' +
13         results.photos[0]?.photo_reference +
14         '&key=' +
15         apiKey
16       "
17     />
18   </div>
19   <div class="search-data">
20     <div class="name" *ngIf="!!results.name">
21       <span>{{ i + 1 }}. </span>{{ results.name }}<span></span>
22     </div>
23     <div *ngIf="!!results.formatted_address">
24       <span>{{ results.formatted_address }}</span>
25     </div>
26     <div *ngIf="results.rating">
27       Rating:
28       {{ results.rating }}
29     </div>
30     <div *ngIf="results.user_ratings_total">
31       {{ results.user_ratings_total }} users rated.
32     </div>
33   </div>
34 </div>
```

In line 1, “search” is a string with the value “Search result shows here”. In line 5, we are using ngFor directive to loop over searchResult$ observable. The “| async” pronounced as pipe async after searchResult$ is for subscribing to the observable inside the HTML file. Each of the contents of searchResult$ is saved in the variable “results” with index as “i”. In line 9, we are using the ngIf directive to check if there are images in the search results. If it returns true, i.e. there are images, then, the images are shown using the “img” tag, otherwise, the images are not displayed. ngIf functions like an “if” statement, but this time, it is used in the HTML file. In line 12, “src” references the directories of the images. In line 20, we are checking if there is “name” in the search result, and we are displaying the name with its index number in line 21. The rest of the code is also checking if some variables are present in the search results, and outputting those variables.

Search-result.component.ts

```
1  import { Component, Input, OnInit } from '@angular/core';
2  import { ReplaySubject } from 'rxjs';
3  import { environment } from 'src/environments/environment';
4
   You, an hour ago | 1 author (You)
5  @Component({
6    selector: 'app-search-result',
7    templateUrl: './search-result.component.html',
8    styleUrls: ['./search-result.component.scss'],
9  })
10 export class SearchResultComponent implements OnInit {
11   @Input() set searchResult(val: any) {
12     this.searchResult$.next(val ? val.results : []);
13   }
14   search: string = 'Search result shows here';
15   ngOnInit(): void {}
16   public searchResult$ = new ReplaySubject<any[]>();
17   apiKey = environment.googleApiKey;
18   constructor() {}
19 }
```

In line 11, we are receiving searchResult from the app component using the @Input() decorator. searchResult$ observable of type ReplaySubject is declared in line 16. In line 17 is the variable apiKey whose value is saved in the environments file. The environments file is imported in line 3. In line 12, we want to save the searchResult which we receive from the app component in the observable searchResult$ using the method next(). The argument of the next method is a ternary operator. In the ternary operator, we are checking if val is truthy. If val is truthy, we will save val.results in the observable, otherwise, we will save an empty array in the observable.

Before we move to the app component, there is a file called index.ts in the _coomponents folder which is used for exporting all the components in the _components folder.

```
1  export * from './filter/filter.component';
2  export * from './map/map.component';
3  export * from './search-result/search-result.component';
4  export * from './header/header.component';
5  export * from './footer/footer.component';
```

This method helps us to reduce the length of the import directories. Instead of referencing the components directly during imports, we will be referencing the components through the index.ts file. The naming of the index.ts file is arbitrary.

App component

App.component.html

App component is the parent component in this application. The selectors of filter component, footer component, header component, map component, and search-result component are placed in the app component. From lines 2 to 5, is the selector of the header component. The selector of the header component is “<app-header></app-header>”. The selectors of the other components follow the same pattern. Parenthesis together with the square bracket [( )] is used for two-way binding. Two-way binding gives components in your application a way to share data. Two-way binding is used to listen for events and update values simultaneously between parent and child components. The [( )] syntax combines the brackets of property binding, [ ], with the parentheses of event binding, ( ). To successfully use two-way binding in any of our components, we need to import FormsModule in the app.module.ts file.

```
1  <div class="header">
2    <app-header
3      (itemType)="yelpDataOutput($event)"
4      (place)="cityName($event)"
5      [menuList]="menuList$ | async"
6    ></app-header>
7    <ul class="items">
8      <li class="restaurant">
9        Restaurant <i class="fas fa-angle-down"></i>
10       <ul class="restaurant-items">
11         <div>
12           <li>
13             <i class="fas fa-biking"></i
14             ><span (click)="menuFxn('Delivery')">Delivery</span>
15           </li>
16           <li>
17             <i class="fas fa-hamburger"></i
18             ><span (click)="menuFxn('Citizens')">Citizens</span>
19           </li>
20           <li>
21             <i class="fas fa-cocktail"></i
22             ><span (click)="menuFxn('Chinese')">Chinese</span>
23           </li>
24           <li>
25             <i class="fas fa-pizza-slice"></i
26             ><span (click)="menuFxn('Italian')">Italian</span>
27           </li>
28         </div>
29         <div>
30           <li>
31             <i class="far fa-calendar-check"></i
32             ><span (click)="menuFxn('Reservations')">Reservations</span>
33           </li>
34           <li>
35             <i class="fas fa-bread-slice"></i
36             ><span (click)="menuFxn('Japanese')">Japanese</span>
37           </li>
38           <li>
39             <i class="fas fa-hotdog"></i
40             ><span (click)="menuFxn('Mexican')">Mexican</span>
41           </li>
42           <li>
43             <i class="fas fa-fish"></i
44             ><span (click)="menuFxn('Thai')">Thai</span>
45           </li>
46         </div>
47       </ul>
48     </li>
```

In line 3, we are using event binding “(itemType)” to send data from the header component to the app component. “@Ouput( ) itemType” is used in the header component to bind the event between the header component and the app component. The method yelpDataOutput($event) is bound to the

itemType event, and the data sent from the child component is passed to the parent component through the argument (i.e. $event) of the yelpDataOutput() method. What we have in line 4 is similar to what we have in line 3. In line 5, we are using property binding to send data (i.e. menuList$) from the app component to the header component. From lines 7 to 183, we are using ul (i.e. unordered list) and li (i.e. list item) to display the menu and sub-menu items. All the list items have a similar pattern. If we take the list item from lines 13 to 15 for example, in line 13, we have fontawesome icon. In line 14, we have a span whose htmlContent is Delivery and we are binding a click event to it. When a click occurs, a method menuFxn() with Delivery as its argument is executed.

```
49      <li class="craft-service">
50        Craft Services <i class="fas fa-angle-down"></i>
51        <ul class="craft-service-list">
52          <div>
53            <li>
54              <i class="fas fa-hammer"></i
55              ><span (click)="menuFxn('Construction Company')"
56                >Construction company</span
57              >
58            </li>
59            <li>
60              <i class="fas fa-plug"></i
61              ><span (click)="menuFxn('Electrician')">Electrician</span>
62            </li>
63            <li>
64              <i class="fas fa-hands-wash"></i
65              ><span (click)="menuFxn('Cleaning company')">Cleaning company</span>
66            </li>
67            <li>
68              <i class="fas fa-fire-alt"></i
69              ><span (click)="menuFxn('Heating and air conditioning')"
70                >Heating & air conditioning</span
71              >
72            </li>
73          </div>
74          <div>
75            <li>
76              <i class="fas fa-mountain"></i
77              ><span (click)="menuFxn('Landscaping')">Landscaping</span>
78            </li>
79            <li>
80              <i class="fas fa-user-lock"></i
81              ><span (click)="menuFxn('Locksmiths')">Locksmiths</span>
82            </li>
83            <li>
84              <i class="fas fa-truck-moving"></i
85              ><span (click)="menuFxn('Moving company')">Moving company</span>
86            </li>
87            <li>
88              <i class="fas fa-shower"></i
89              ><span (click)="menuFxn('Installers')">Installers</span>
90            </li>
91          </div>
92        </ul>
93      </li>
```

```
<li class="around-the-car">
  Around the Car <i class="fas fa-angle-down"></i>
  <ul class="around-the-car-list">
    <div>
      <li>
        <i class="fas fa-wrench"></i
        ><span (click)="menuFxn('Auto repair shop')">Auto repair shop</span>
      </li>
      <li>
        <i class="fas fa-tools"></i
        ><span (click)="menuFxn('Car preparation')">Car preparation</span>
      </li>
      <li>
        <i class="fas fa-leaf"></i
        ><span (click)="menuFxn('plant shop and body shop')"
          >Plant shop & body shop</span
        >
      </li>
      <li>
        <i class="fas fa-car"></i
        ><span (click)="menuFxn('Car wash')">Car wash</span>
      </li>
    </div>
    <div>
      <li>
        <i class="fas fa-key"></i
        ><span (click)="menuFxn('Car dealer')">Car dealer</span>
      </li>
      <li>
        <i class="fas fa-oil-can"></i
        ><span (click)="menuFxn('Oil change')">Oil change</span>
      </li>
      <li>
        <i class="fas fa-parking"></i
        ><span (click)="menuFxn('Parking spaces')">Parking spaces</span>
      </li>
      <li>
        <i class="fas fa-truck-monster"></i
        ><span (click)="menuFxn('Towing service')">Towing service</span>
      </li>
    </div>
  </ul>
</li>
```

```
137      <li class="more">
138        More <i class="fas fa-angle-down"></i>
139        <ul class="more-list">
140          <div>
141            <li>
142              <i class="fas fa-pump-soap"></i
143              ><span (click)="menuFxn('Chemical cleaning')"
144                >Chemical cleaning</span
145              >
146            </li>
147            <li>
148              <i class="fas fa-mobile-alt"></i
149              ><span (click)="menuFxn('Mobile phone repair')"
150                >Mobile phone repair</span
151              >
152            </li>
153            <li>
154              <i class="fas fa-glass-cheers"></i
155              ><span (click)="menuFxn('Bars')">Bars</span>
156            </li>
157            <li>
158              <i class="fas fa-beer"></i
159              ><span (click)="menuFxn('Nightlife')">Nightlife</span>
160            </li>
161          </div>
162          <div>
163            <li>
164              <i class="fas fa-soap"></i
165              ><span (click)="menuFxn('Hairdressers')">Hairdressers</span>
166            </li>
167            <li>
168              <i class="fas fa-dumbbell"></i
169              ><span (click)="menuFxn('Gyms')">Gyms</span>
170            </li>
171            <li>
172              <i class="fas fa-hot-tub"></i
173              ><span (click)="menuFxn('Massages')">Massages</span>
174            </li>
175            <li>
176              <i class="fas fa-shopping-cart"></i
177              ><span (click)="menuFxn('Shopping')">Shopping</span>
178            </li>
179          </div>
180        </ul>
181      </li>
182    </ul>
183  </div>
```

```
184
185  <div class="project">
186    <div class="filter">
187      <app-filter
188        [category]="category$ | async"
189        (filterText)="filterFunction($event)"
190      >
191      </app-filter>
192    </div>
193    <div class="search-result">
194      <app-search-result [searchResult]="data$ | async"></app-search-result>
195    </div>
196    <div class="map">
197      <app-map [searchResult]="data$ | async"></app-map>
198    </div>
199  </div>
200  <app-footer></app-footer>
```

From line 187 to 191 is the selector of the filter component. In line 188, we are using property binding to send data from the app component to the filter component. In line 189, we are using event binding to

send data from the filter component to the app component. In line 194 is the selector for the search-result component, and the selector for the map component is in line 197. In line 200 is the selector for the footer component.

App.component.ts

```
1   import { Component, OnInit } from '@angular/core';
2   import { ReplaySubject } from 'rxjs';
3   import { HttpClient } from '@angular/common/http';
4   import * as _ from 'lodash';
5   import { environment } from 'src/environments/environment';
6
    You, a minute ago | 1 author (You)
7   @Component({
8     selector: 'app-root',
9     templateUrl: './app.component.html',
10    styleUrls: ['./app.component.scss'],
11  })
12  export class AppComponent implements OnInit {
13    title = 'yelpclone';
14    searchItem = '';
15    searchCity = '';
16    filter = '';
17    data$ = new ReplaySubject();
18    category$ = new ReplaySubject<string[]>();
19    filter$ = new ReplaySubject<string>();
20    menuList$ = new ReplaySubject<string>();
21    year$ = new ReplaySubject<number>();
22
23    constructor(private http: HttpClient) {}
24    ngOnInit(): void {
25      this.filter$.subscribe((val) => {
26        this.filter = val ? val : '';
27      });
28      this.year$.next(new Date().getFullYear());
29    }
30
31    yelpDataOutput(event: any) {
32      this.searchItem = event.trim();
33      this.search();
34    }
35    cityName(event: any) {
36      this.searchCity = event.trim();
37      this.search();
38    }
39    filterFunction(filterText: string) {
40      this.filter$.next(filterText);
41      let url = `${environment.CORS}https://maps.googleapis.com/maps/api/place/textsearch/json?query=${this.searchItem}+in+${this.searchCity}
42      return this.http.get(url).subscribe((data) => {
43        this.data$.next(data);
44      });
45    }
```

We imported HttpClient in line 3 for making API requests, lodash is imported in line 4, and the environments file is imported in line 5. Lodash can be installed using the following command “npm install lodash”. Lodash makes JavaScript easier by taking the hassle out of working with arrays, numbers, objects, strings, etc. From line 13 to 21, we have variables declaration and assignment. The variables declared in lines 17 to 21 are observables of type ReplaySubject. In line 23, we have “private http: HttpClient” in the constructor. This is called dependency injection. Dependency injection is a technique in which an object receives other objects that it depends on, called dependencies. In this case, the service “HttpClient” is injected into the app component. In line 25, we are taking the data saved in the observable filter$, and in line 26, we are saving the data in the variable filter of type string. In line 28, we are getting the current year, and saving it in the observable year$. From line 31 to 34, we have yelpDataOutput(event: any) method which has the argument event of type any. The argument of yelpDataOutput() is the data that is received from the header component. In line 32, we are using the trim() method to remove whitespaces from both ends of the data received from the header component and saving it as a string in searchItem.

In line 33, we are calling search(). In lines 41 and 49, we are passing searchItem, searchCity and googleApi key as part of the query string. From line 35 to 38, we have cityName(event) where we are receiving the name of the city from the header component, using trim() to remove the whitespaces from both ends of the string, and calling search() in line 37. From line 39 to 45, we have filterFunction(filterText). Here, when a filter object is clicked, the value of the clicked object is passed from filter component to app component, and it forms part of the query string in line 41. In line 41, we are saving Google place API as "url", and we are making an http get request in line 42. In line 42, we are subscribing to this.http.get(url) to get the json data from the API and save it in data$ in line 43. The Google place API is https://maps.googleapis.com/maps/api/place/textsearch/json?query=${this.searchItem}+in+${this.searchCity}&key=${environment.googleApiKey} but if you used it this way, it will give the following error in the developer console http://localhost:4200' has been blocked by CORS policy: No 'Access-Control-Allow-Origin' header is present on the requested resource. Place holders which are indicated by the dollar sign and curly braces (${expression}) are used to add searchItem, searchCity, and filterText to Google place API string in lines 41 and 49. For us to be able to use the place holders in lines 41 and 49, we enclosed the Google place API by the backtick (` `) character instead of double or single quotes. To overcome the error due to CORS header, and be able to get the expected search result, we need to find a way to add CORS header to our request, and this was achieved by prefixing the Google place API with https://cor5allow.herokuapp.com/ which we saved in the environments file as CORS. You can read up Google place API for a good understanding of how to use the API. Also, to understand how we added CORS to the request header, you can do a further reading from the following urls https://stackoverflow.com/questions/29670703/how-to-use-cors-anywhere-to-reverse-proxy-and-add-cors-headers and https://github.com/Rob--W/cors-anywhere.

```
46
47  //https://cor5allow.herokuapp.com/ is used to grant cors access
48  search() {
49    let url = `${environment.CORS}https://maps.googleapis.com/maps/api/place/textsearch/json?query=${this.searchItem}+in+${this.searchCity}&
50    return this.http.get(url).subscribe((data: any) => {
51      this.data$.next(data);
52      //empty array to save the types
53      let types = [];
54      //for loop to iterate over the results from the api
55      for (let category of data.results) {
56        types.push(...category.types);
57        //using lodash to get the unique contents in type array
58        let uniqCategory = _.uniq(types);
59        this.category$.next(uniqCategory);
60      }
61    });
62  }
63  menuFxn(item: string) {
64    // the string value is taken from the function on click event, and it is saved in an observable ( i.e. replaysubject)
65    this.menuList$.next(item);
66  }
67  }      You, a month ago • initial commit
```

From line 48 to 62, we have search() method. This is the function that is called whenever we perform a search. Line 49 is similar to line 41, but the difference is that in line 49 type=${event} is excluded from Google place API because we are not performing filtering here. type={event} is used to filter the search result in line 41 based on the filter element that is clicked. From line 55 to 60, we are iterating over the search result from the API and saving it in the variable name category. In line 56, we are pushing category.types into the empty array we declared in line 53. category.types is an array of filter strings, and we are performing spread operation on it before pushing it into the empty array. Spread syntax (…) can be used when all elements from an object or array need to be included in a list of some kind, for example …[1, 2, 4] will give the result //1, 2, 3. In line 58, we are using lodash to compare the content of types

array, take the unique elements, and then save them in the variable uniqCategory. In line 59, we are saving uniqCategory in the observable category$ which will be sent to filter component. From line 63 to 66, we have “menuFxn(item: string)” where item is of type string, and it is the argument of the function which we are receiving from the html file. In line 65, we are saving item in the observable menuList$ which we are sending to header component.

App.module.ts

This is what our app.module.ts looks like. Here, we have all our components and modules imported. Our components are added to the declaration array from line 15 to 21, and modules are added to the imports array in line 23. You usually will not need to manually import the components in the app.module.ts file because it is automatically created for us when we create our components.

```
1  import { NgModule } from '@angular/core';
2  import { BrowserModule } from '@angular/platform-browser';
3
4  import { AppRoutingModule } from './app-routing.module';
5  import { AppComponent } from './app.component';
6  import { MapComponent } from './_components/map/map.component';
7  import { FilterComponent } from './_components/filter/filter.component';
8  import { SearchResultComponent } from './_components/search-result/search-result.component';
9  import { HeaderComponent } from './_components/header/header.component';
10 import { FormsModule } from '@angular/forms';
11 import { HttpClientModule } from '@angular/common/http';
12 import { FooterComponent } from './_components/footer/footer.component';
13
   You, a month ago | 1 author (You)
14 @NgModule({
15   declarations: [
16     AppComponent,
17     MapComponent,
18     FilterComponent,
19     SearchResultComponent,
20     HeaderComponent,
21     FooterComponent,
22   ],
23   imports: [BrowserModule, AppRoutingModule, FormsModule, HttpClientModule],
24   providers: [],
25   bootstrap: [AppComponent],
26 })
27 export class AppModule {}
28
```

You can check out the application live on http://yelp-clone.s3-website.us-east-2.amazonaws.com/