

With these quick steps, we have created a component that handles the rendering of a single trip and refactored our existing code to simplify the application. Additionally, the component can be used anywhere in the application.

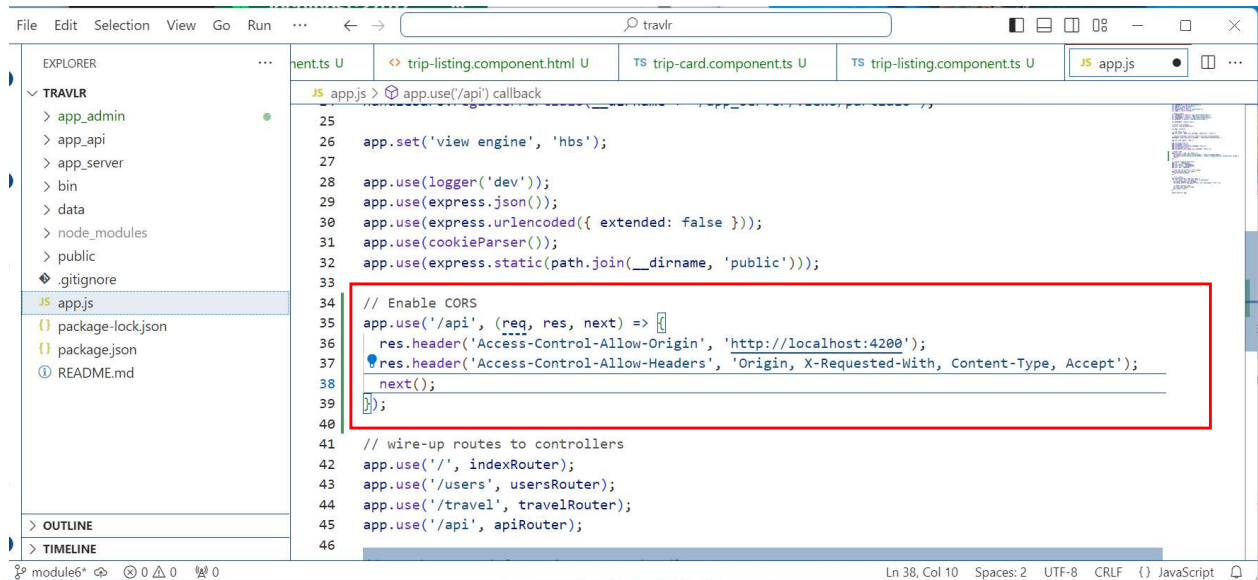
Create Trip Data Service

Now that we have seen what it takes to create the necessary components to render our trips, we need to start thinking about where the data is going to come from. In a single-page application (SPA), it is increasingly common to obtain information from another part of the application, particularly with distributed (or multi-tier) architectures. In a SPA, this generally means calling a REST endpoint to obtain data. Angular supports Separation of Concerns (SoC) by defining a “service” to contain functions or objects that are used by components to perform actions.

In Module 5 we created an “/api” REST endpoint on our backend application using Express. This was designed to handle data requests, and we were successfully able to test that with our application. Now we will create an Angular service to handle access to the backend endpoint for trip information.

1. The first step in allowing the Angular admin site to make calls against the Express backend API is to adjust the Express application to allow for the external calls. This change is made in

the **app.js** file at the root of our application tree and it enables what is generally referred to as CORS (Cross Origin Resource Sharing).



The screenshot shows a code editor with the file explorer on the left displaying the project structure. The main editor shows the `app.js` file. A red box highlights the CORS configuration code between lines 34 and 39:

```

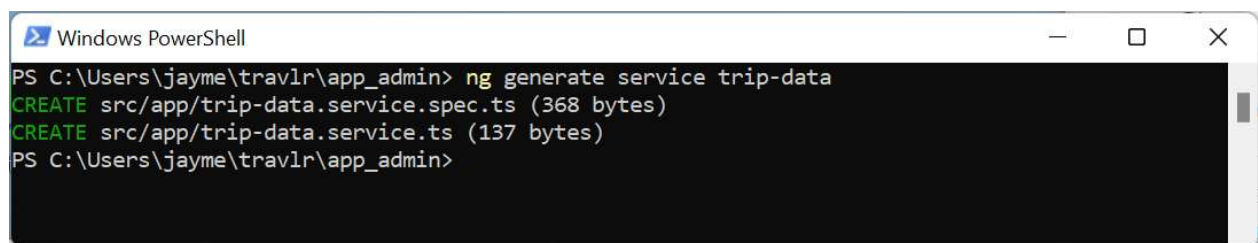
34 // Enable CORS
35 app.use('/api', (req, res, next) => {
36   res.header('Access-Control-Allow-Origin', 'http://localhost:4200');
37   res.header('Access-Control-Allow-Headers', 'Origin, X-Requested-With, Content-Type, Accept');
38   next();
39 });

```

You can refer to the section “Allowing CORS requests in Express” in Chapter 9 of the textbook for additional information.

- Now we will need to create a new Angular service called `trip-data` to allow us to configure the appropriate data paths. The command we need to use to accomplish this is:

```
ng generate service trip-data
```



The screenshot shows a Windows PowerShell terminal window with the following output:

```

PS C:\Users\jayne\travlr\app_admin> ng generate service trip-data
CREATE src/app/trip-data.service.spec.ts (368 bytes)
CREATE src/app/trip-data.service.ts (137 bytes)
PS C:\Users\jayne\travlr\app_admin>

```

- At this point we need to do some house-keeping in order to prepare the application structure for additional services later on. We will want to create a new folder under **app_admin/src/app** called **services** and move the **trip-data.service.*** files from the **app_admin/src/app** folder into the new **app_admin/src/app/services** folder. When

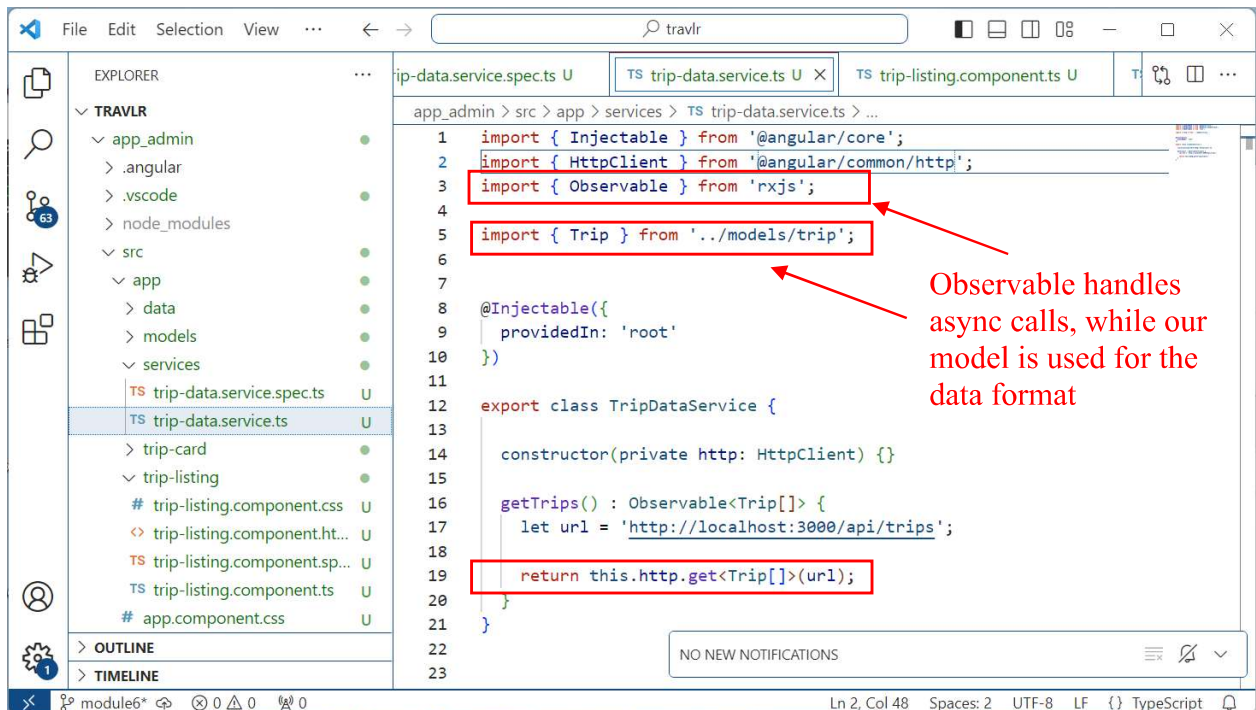
complete, your file structure should resemble this:

4. In order for our new basic service to function properly, we need some way to communicate what type of data it is going to handle. For this we need to create a **models** folder. We are going to put this parallel to the **data** and **service** directories under **app_admin/src/app**. This is where we will create an interface to define the data for a single Trip that will be received from the API endpoint as a JSON object. We will define the model in a file named **trip.ts**.

Instances of this interface will be used to transfer HTML form data to the component for rendering as well as between components and the REST endpoint. Angular will automatically marshal the data back and forth between JSON and JavaScript objects.

- Now that we have the interface defined, we need to introduce it to the service. In order to accomplish this, we need to edit the **trip-data.service.ts** file and add some code. This is going to be very straight-forward because we aren't going to introduce error handling on the connection at this time, but we are going to introduce another module for import because the HTTP connection in the service is *asynchronous*.

You should recall that we had to treat the internal API calls a bit differently in the Express application because they were issued asynchronously in Node JS. The same item applies here with Angular, but the implementation is slightly different. The module that we are going to introduce is '*Observable*' from the *rxjs* package.



The screenshot shows the Visual Studio Code editor with the file explorer on the left and the code editor on the right. The file explorer shows the project structure with the following files and folders:

- TRAVLR
 - app_admin
 - .angular
 - .vscode
 - node_modules
 - src
 - app
 - data
 - models
 - services
 - trip-data.service.spec.ts
 - trip-data.service.ts
 - trip-card
 - trip-listing
 - trip-listing.component.css
 - trip-listing.component.ht...
 - trip-listing.component.sp...
 - trip-listing.component.ts
 - app.component.css

The code editor shows the **trip-data.service.ts** file with the following code:

```

1 import { Injectable } from '@angular/core';
2 import { HttpClient } from '@angular/common/http';
3 import { Observable } from 'rxjs';
4
5 import { Trip } from '../models/trip';
6
7
8 @Injectable({
9   providedIn: 'root'
10 })
11
12 export class TripDataService {
13
14   constructor(private http: HttpClient) {}
15
16   getTrips() : Observable<Trip[]> {
17     let url = 'http://localhost:3000/api/trips';
18
19     return this.http.get<Trip[]>(url);
20   }
21 }
22
23

```

Annotations in the image:

- Red boxes highlight the following lines of code:
 - Line 3: `import { Observable } from 'rxjs';`
 - Line 5: `import { Trip } from '../models/trip';`
 - Line 19: `return this.http.get<Trip[]>(url);`
- Red arrows point from the text "Observable handles async calls, while our model is used for the data format" to the `Observable` import and the `getTrips` method.

Because we are returning an Observable object, our component can attach to that object and get notification when the async call has been completed and the associated promise fulfilled.

- Before we move forward with making the change to the **trip-listing.component.ts** file we need to make one small change to our application so that our new data service will be functional. We have to enable the HTTP services at the application level. To do this, we will add two lines to the **app.config.ts** file in our **app_admin/src/app** folder.



The two lines we added to the file allow the Angular environment to communicate via HTTP. Without these lines, our new service would not be able to function.

7. Now we need to make some changes to the **trip-listing.components.ts** file so that we can take advantage of our new service, and pull our data from the database instead of our local data file. The changes include:

- a. Importing our Trip model.

```
import { Trip } from '../models/trip';
```

- b. Importing our TripDataService

```
import { TripDataService } from '../services/trip-data.service';
```

- c. Registering TripDataService as a provider

```
providers: [TripDataService]
```

- d. Creating a constructor to initialize the TripDataService

```

constructor(private tripDataService: TripDataService) {
  console.log('trip-listing constructor');
}

```

- e. Creating a method that will call the getTrips() method in TripDataService

```

private getStuff(): void {
  this.tripDataService.getTrips()
    .subscribe({

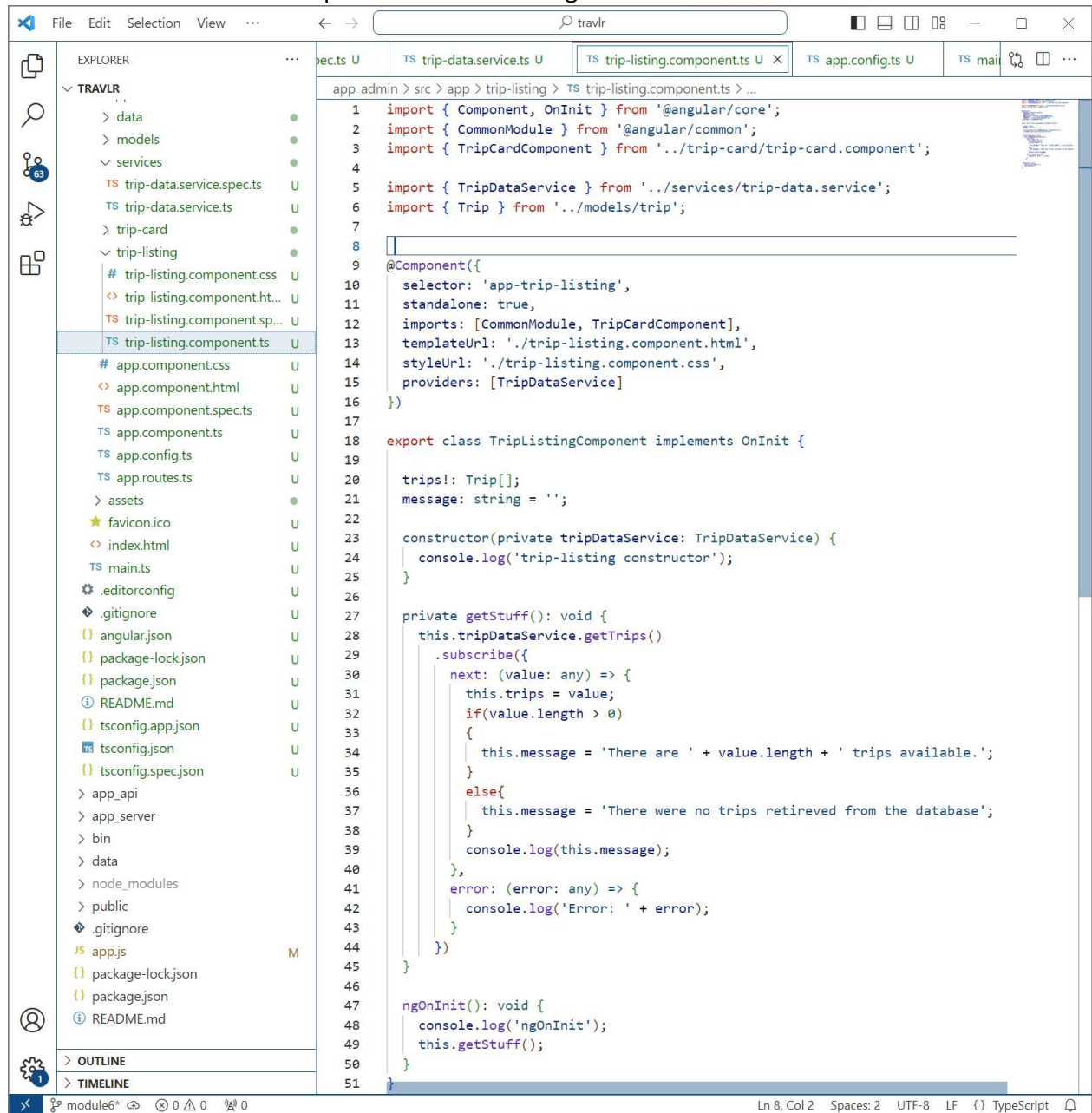
```

```
next: (value: any) => {
  this.trips = value;
  if(value.length > 0)
  {
    this.message = 'There are ' + value.length + ' trips
available.';
  }
  else{
    this.message = 'There were no trips retrieved from the
database';
  }
  console.log(this.message);
},
error: (error: any) => {
  console.log('Error: ' + error);
}
})
}
```

- f. And creating an `ngOnInit` method that will call our private method when the component is initialized.

```
ngOnInit(): void {
  console.log('ngOnInit');
  this.getStuff();
}
```

You can see how this is completed in the following screenshot:



The screenshot shows the Visual Studio Code editor with the following structure:

- EXPLORER:**
 - TRAVLR
 - data
 - models
 - services
 - trip-data.service.spec.ts
 - trip-data.service.ts
 - trip-card
 - trip-listing
 - trip-listing.component.css
 - trip-listing.component.html
 - trip-listing.component.spec.ts
 - trip-listing.component.ts (selected)
 - app.component.css
 - app.component.html
 - app.component.spec.ts
 - app.component.ts
 - app.config.ts
 - app.routes.ts
 - assets
 - favicon.ico
 - index.html
 - main.ts
 - .editorconfig
 - .gitignore
 - angular.json
 - package-lock.json
 - package.json
 - README.md
 - tsconfig.app.json
 - tsconfig.json
 - tsconfig.spec.json

- FILE:** app_admin > src > app > trip-listing > TS trip-listing.component.ts > ...
- EDITOR:**

```

1  import { Component, OnInit } from '@angular/core';
2  import { CommonModule } from '@angular/common';
3  import { TripCardComponent } from '../trip-card/trip-card.component';
4
5  import { TripDataService } from '../services/trip-data.service';
6  import { Trip } from '../models/trip';
7
8
9  @Component({
10     selector: 'app-trip-listing',
11     standalone: true,
12     imports: [CommonModule, TripCardComponent],
13     templateUrl: './trip-listing.component.html',
14     styleUrls: ['./trip-listing.component.css'],
15     providers: [TripDataService]
16 })
17
18 export class TripListingComponent implements OnInit {
19
20     trips!: Trip[];
21     message: string = '';
22
23     constructor(private tripDataService: TripDataService) {
24         console.log('trip-listing constructor');
25     }
26
27     private getStuff(): void {
28         this.tripDataService.getTrips()
29             .subscribe({
30                 next: (value: any) => {
31                     this.trips = value;
32                     if(value.length > 0)
33                     {
34                         this.message = 'There are ' + value.length + ' trips available.';
35                     }
36                     else{
37                         this.message = 'There were no trips retrieved from the database';
38                     }
39                     console.log(this.message);
40                 },
41                 error: (error: any) => {
42                     console.log('Error: ' + error);
43                 }
44             })
45     }
46
47     ngOnInit(): void {
48         console.log('ngOnInit');
49         this.getStuff();
50     }
51 }

```
- STATUS BAR:** module6* | 0 | 0 | 0 | 0 | Ln 8, Col 2 | Spaces: 2 | UTF-8 | LF | {} | TypeScript

If you haven't already done so, make sure that you have your Express app running in a PowerShell Window. From this console, you can see each time an API call is made to the

/api/trips endpoint – this is very helpful with debugging.

```

npm start
PS C:\Users\jayne\travlr> npm start

> travlr@0.0.0 start
> node ./bin/www

Mongoose connected to mongodb://127.0.0.1/travlr
GET /api/trips 304 48.745 ms - -
GET /api/trips 304 4.098 ms - -
GET /api/trips 304 5.158 ms - -
GET /api/trips 304 4.736 ms - -
GET /api/trips 304 3.330 ms - -
GET /api/trips 304 4.318 ms - -
GET /api/trips 304 12.372 ms - -
GET /api/trips 304 2.901 ms - -
GET /api/trips 304 4.122 ms - -
GET /api/trips 304 4.575 ms - -
GET /api/trips 304 4.137 ms - -
GET /api/trips 304 15.885 ms - -
GET /api/trips 304 6.632 ms - -
GET /api/trips 304 5.444 ms - -
GET /api/trips 304 21.408 ms - -
GET /api/trips 304 5.037 ms - -
GET /api/trips 304 5.185 ms - -
GET /api/trips 304 14.349 ms - -
GET /api/trips 304 13.155 ms - -
GET /api/trips 304 20.935 ms - -
GET /api/trips 304 5.402 ms - -
GET /api/trips 304 4.054 ms - -

```

Additionally, if you haven't already, you should open the developer tools panel for your web-browser. In the Microsoft-Edge Browser, you can achieve this with CTRL-Shift-I. This is a very, very useful method to assist with debugging Angular applications – as you can watch the local console and output *console.log* messages to provide debugging information for your development process. Here you can see the results of the message that we emplaced in our 'getStuff' method.

