COMP-3411 / 4704
Web Programming II

Lecture 12 - Continuing with Angular

Daniel Pittman, Ph.D., CISSP

02/10/2022

Advertisement: Courses Offered Next Quarter

- Since Spring registration is quickly approaching, I wanted to take a quick second to promote electives I'm teaching!
- I'll be teaching one class next quarter:
  - **Web Programming Projects (Web III): T / Th 2:00 pm - 3:50 pm**
    - Can count as advanced programming requirement!
    - CRN XXXX / COMP 3XXX - X
    - CRN XXXX / COMP 4XXX - X

Course Description: Web Programming Projects

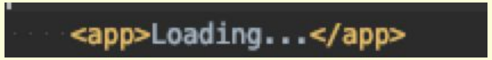
Many assignments that you work on in college are artificial problems created to demonstrate a concept or reinforce understanding of an idea. While this is valuable, and serves a very important purpose, it is important also to have a chance to work on a real problem that addresses an actual need. That is what this class is all about!

In this course you will learn how to develop, as a group, a full-stack web application that is capable of serving dynamic content from a database. We will use the MongoDB, ExpressJS, Angular, and Node.js (MEAN) software stack to work on a real-life problem presented to us by an external product owner. In the class we will use the Scrum framework for Agile development to work, as a software team, through several sprints of development. You will be peer reviewing each other throughout the course, and the product owner will also be reviewing your product through end-of-sprint demos as features are completed. My goal for this class is for it to be a fun, collaborative, and educational environment that demonstrates what it is like to work as a real software team!

How did Lab 10 work?

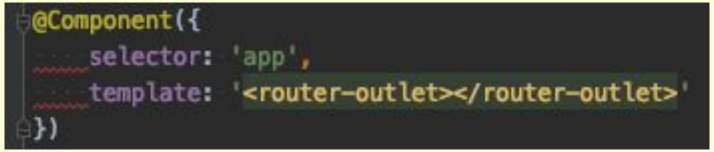
- Last class we went through the entire project structure of the client folder, and you completed a lab that demonstrated two-way data binding the ngFor directive
- Today we're going to take a step back and highlight a few key areas of the client application that made Lab 10 possible!

Key components of our Angular Application

- The **app** HTML tag
  - In the **client/app.template.html** file there is this line:
    - 

```
<app>Loading...</app>
```

  - This tag will be replaced by Angular with the contents of the HTML fragment for the Component whose **selector** is **app**
  - The file **client/app/main/app.component.ts** defines this **selector:**



```
@Component({
    selector: 'app',
    template: '<router-outlet></router-outlet>'
})
```

  - The **template** HTML for the **app** component shows one HTML tag: **router-outlet**
  - The **RouterOutlet** component is a placeholder
  - The contents of the currently active Component (determined via **Route** service) will be inserted into the router-outlet tag automatically by Angular

Bootstrapping an Angular Application

- Within this complex Angular application, where does everything start? How does the application **bootstrap** itself?
- In Angular, there are **three different ways** of bootstrapping the application:
  - Bootstrapping a **root module** for our application by calling **platformBrowserDynamic().bootstrapModule(AppModule);**
  - Manually bootstrapping a component using **ngDoBootstrap(...)** and **app.bootstrap(...)**
  - Use **Angular Elements** to turn a Component into a custom element, and then register it in the module's **entryComponents** list as the bootstrap Component
- In our application, we are bootstrapping a root module in **client/app/app.ts**

Key components of our Angular Application

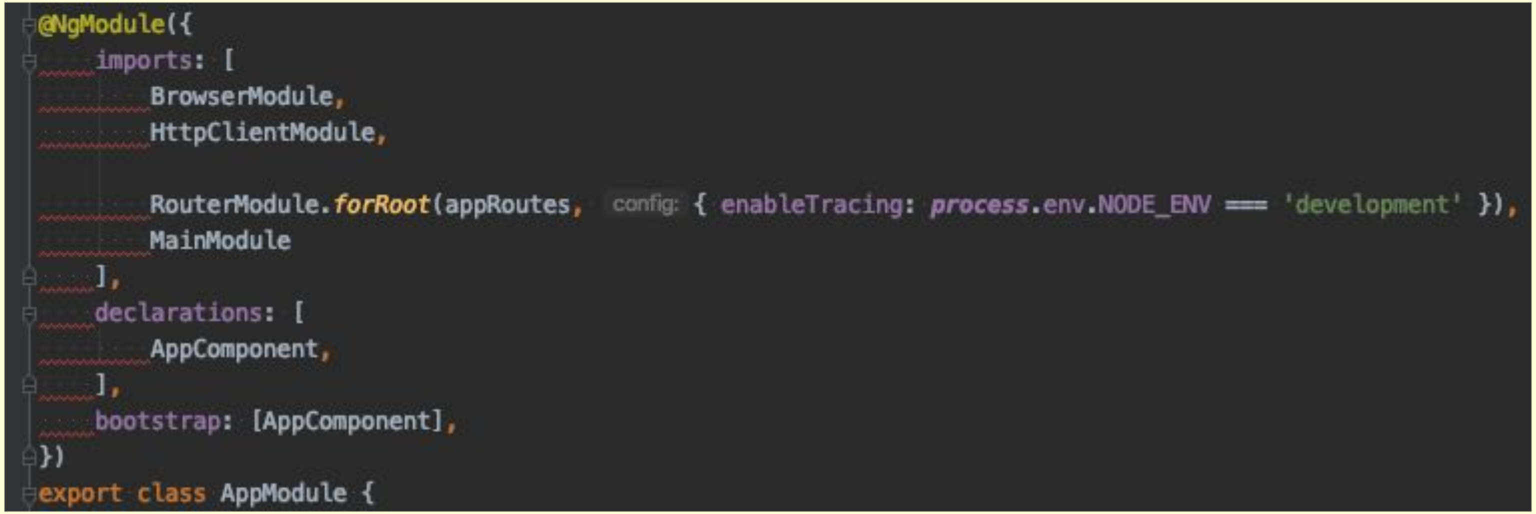
- What does the call to **platformBrowserDynamic().bootstrapModule(AppModule)** do?
- The **platformBrowserDynamic()** function call creates a **platform** for the application module
    - An Angular platform is an **entry point for Angular on a web page**
    - Each page has exactly one platform, and services which are common to every Angular application running on the page are bound to its scope
- Angular can be bootstrapped in other environments besides a browser (such as a server or inside a web worker), so this step is important to specify the environment in which the app will run!

Key components of our Angular Application

- The next call, **bootstrapModule()**, takes the **root module of our application** as its argument
- The module will get processed based on its declared metadata
- Any dependencies and declarations will be processed before anything is rendered onto the DOM
- NgModules help organize the application into related blocks of functionality
    - This is specified in the TypeScript file using the **@NgModule decorator**
- The most important configuration in a NgModule decorator is the **bootstrap** property, which specifies the component to bootstrap when the application starts!
- Once all of the dependencies and services in the provider have been resolved, Angular will instantiate the **bootstrap component** and insert it into the DOM in place of its component selector (custom HTML tag)

Key components of our Angular Application

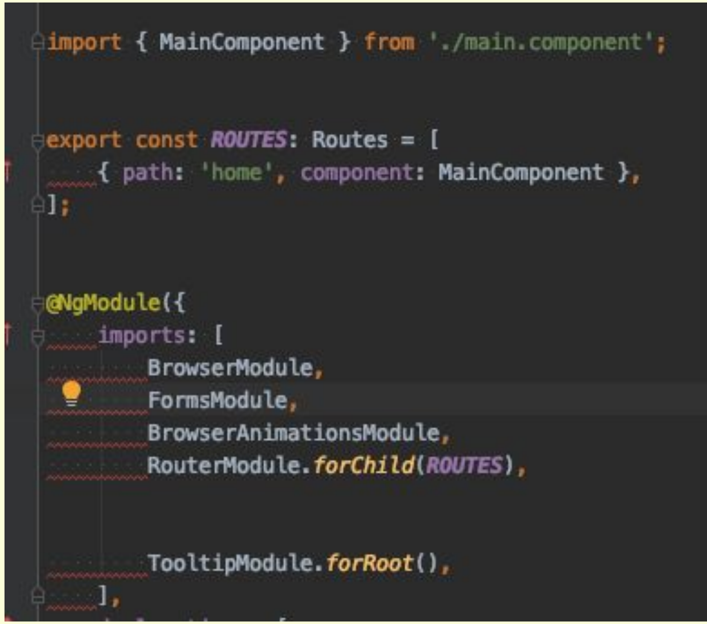
- Here's the NgModule decoration in **client/app/app.module.ts:**



```
@NgModule({
    imports: [
        BrowserModule,
        HttpClientModule,

        RouterModule.forRoot(appRoutes,  config: { enableTracing: process.env.NODE_ENV === 'development' }),
        MainModule
    ],
    declarations: [
        AppComponent,
    ],
    bootstrap: [AppComponent],
})
export class AppModule {
```

Key components of our Angular Application

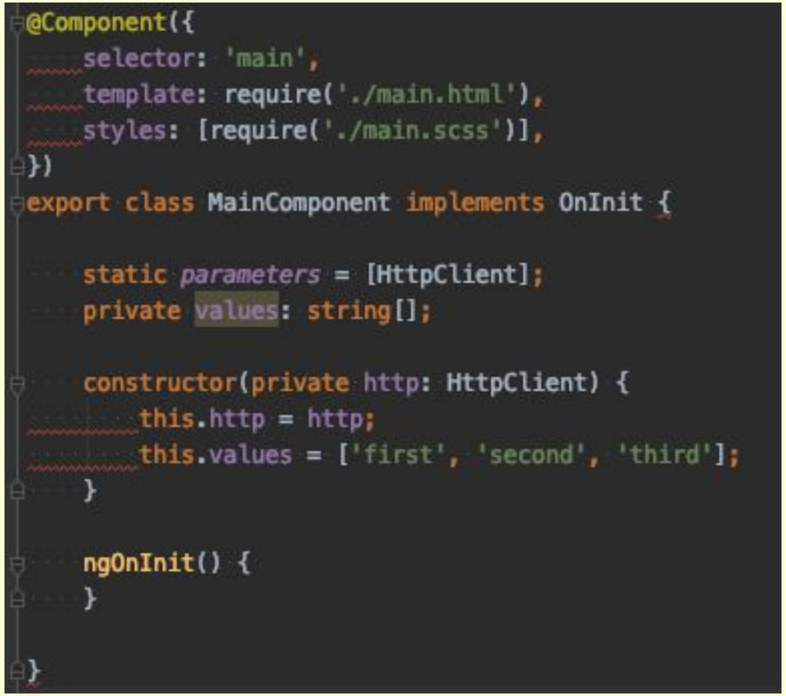
- As you may have noticed, in **client/app/app.ts** we are importing the **main** Component, the same one you modified in Lab 10!
    - The main Component is configured to be the default one to use for our router (more to come on routing next week!), which is why it shows up by default when you run the application
- So what about the main Component itself? How does that work?
    - The primary file for the main Component is **client/app/main/main.component.ts**
    - The module containing the main Component is declared in **client/app/main/main.module.ts**
- The **Module** declaration tells Angular what **Components** are contained within it, how to **route requests** to the Components, and what **dependencies** are required by the Module



```
import { MainComponent } from './main.component';


export const ROUTES: Routes = [
    { path: 'home', component: MainComponent },
];


@NgModule({
    imports: [
        BrowserModule,
        FormsModule,
        BrowserAnimationsModule,
        RouterModule.forChild(ROUTES),


        TooltipModule.forRoot(),
    ],
```

Key components of our Angular Application

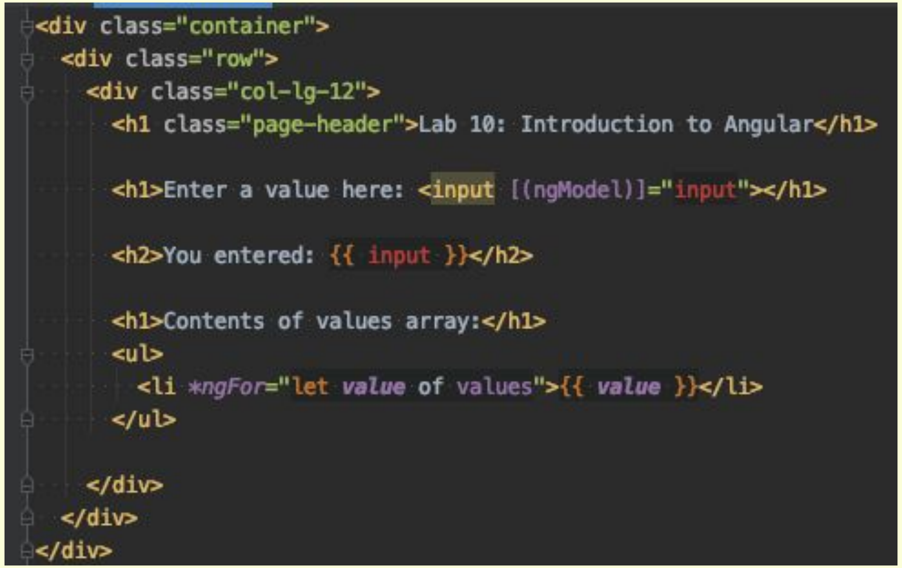
- The **Component** class has a **constructor** in which all automatically injected dependencies will be referenced
- Any custom logic or functions you wish to implement as part of the Component can be added inside the class, and will be available within its scope



```
@Component({
    selector: 'main',
    template: require('./main.html'),
    styles: [require('./main.scss')],
})
export class MainComponent implements OnInit {

    static parameters = [HttpClient];
    private values: string[];

    constructor(private http: HttpClient) {
        this.http = http;
        this.values = ['first', 'second', 'third'];
    }

    ngOnInit() {
    }

}
```

Key components of our Angular Application

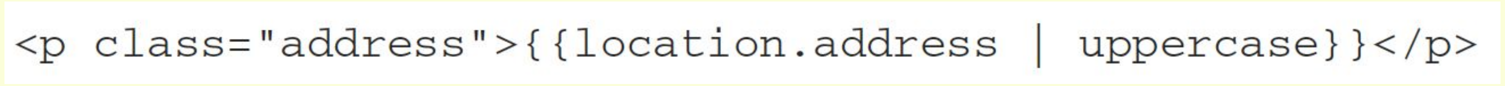
- The main Component declaration also references an **HTML template file and CSS style file**
    - The HTML template file is an HTML **fragment** that will be injected in place of the component's selector at runtime
    - The CSS styles will be injected as well, when the component is selected at runtime
- The HTML within the template is compiled within the context of Angular, and assigned the same scope as the Component its associated to
    - Therefore, two-way data binding is available!



```
<div class="container">
  <div class="row">
    <div class="col-lg-12">
      <h1 class="page-header">Lab 10: Introduction to Angular</h1>

      <h1>Enter a value here: <input [(ngModel)]="input"></h1>

      <h2>You entered: {{ input }}</h2>

      <h1>Contents of values array:</h1>
      <ul>
        <li *ngFor="let value of values">{{ value }}</li>
      </ul>

    </div>
  </div>
</div>
```

Formatting Data Using Angular Pipes

- In Lab 10 we showed how to bind data to our HTML
- If we want to **format** that data, Angular gives us a way to do that using **pipes**
- There are many built-in pipes in Angular for performing functions such as
  - **Formatting currency**
  - **Formatting dates**
  - **Transforming text**
- Pipes can be applied either programmatically inside your Component, or directly within your HTML
- To use a pipe in HTML, you insert the pipe symbol '**|**' after the value you wish to transform, followed by the name of the transformation you are applying



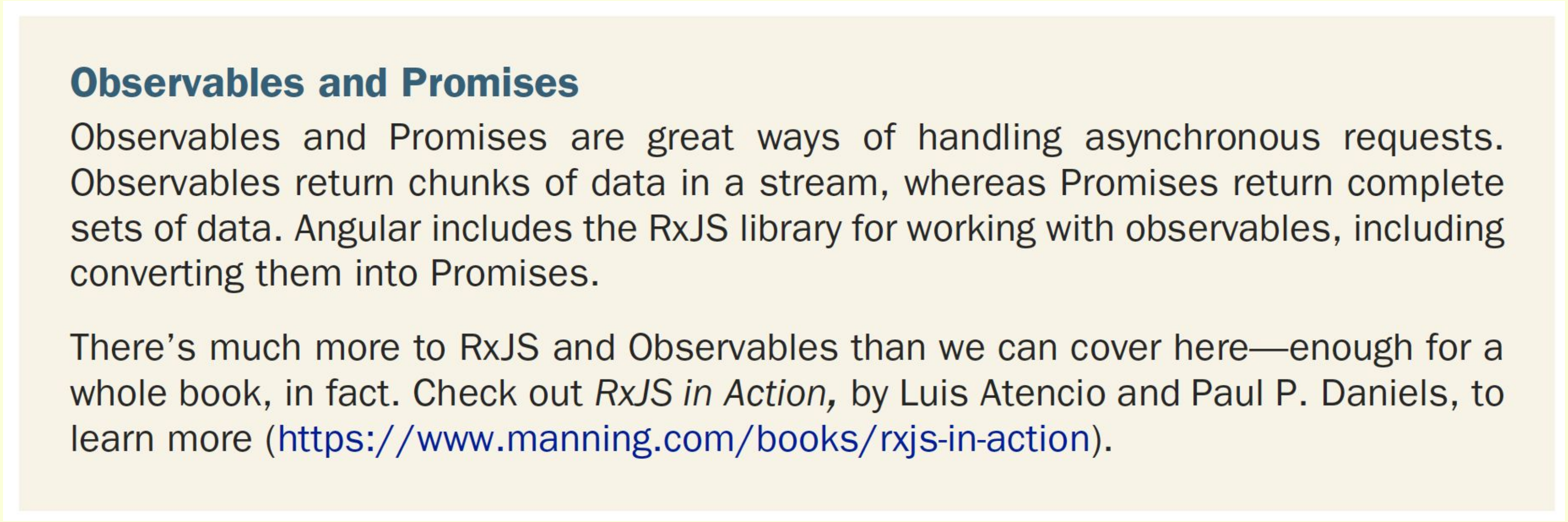
```
<p class="address">{{location.address | uppercase}}</p>
```

Angular Capability - Services

- Ideally, your Component's job is to **enable the user experience**, and nothing more!
- A Component can present properties and methods for data binding, in order to **act as go-between** for the **view** (rendered by the HTML template), and the **application logic / model**
- For other operations, a Component can delegate tasks to a service:
    - Fetching data from the server
    - Validating user input
    - Logging to the console
- **Services** are standalone, injectable objects that allow you to reuse code and share functionality within your application
- By isolating functionality into services, you can also **unit test pieces of your application independently**

Angular Capability - XHR

- Angular provides a **HttpClient** service that makes it easy to retrieve data over the Web
- **HttpClient** methods return an **Observable** or **Promise** since they are asynchronous in nature!
    - We will be using Promises in the class exclusively, but I suggest you read about Observables!

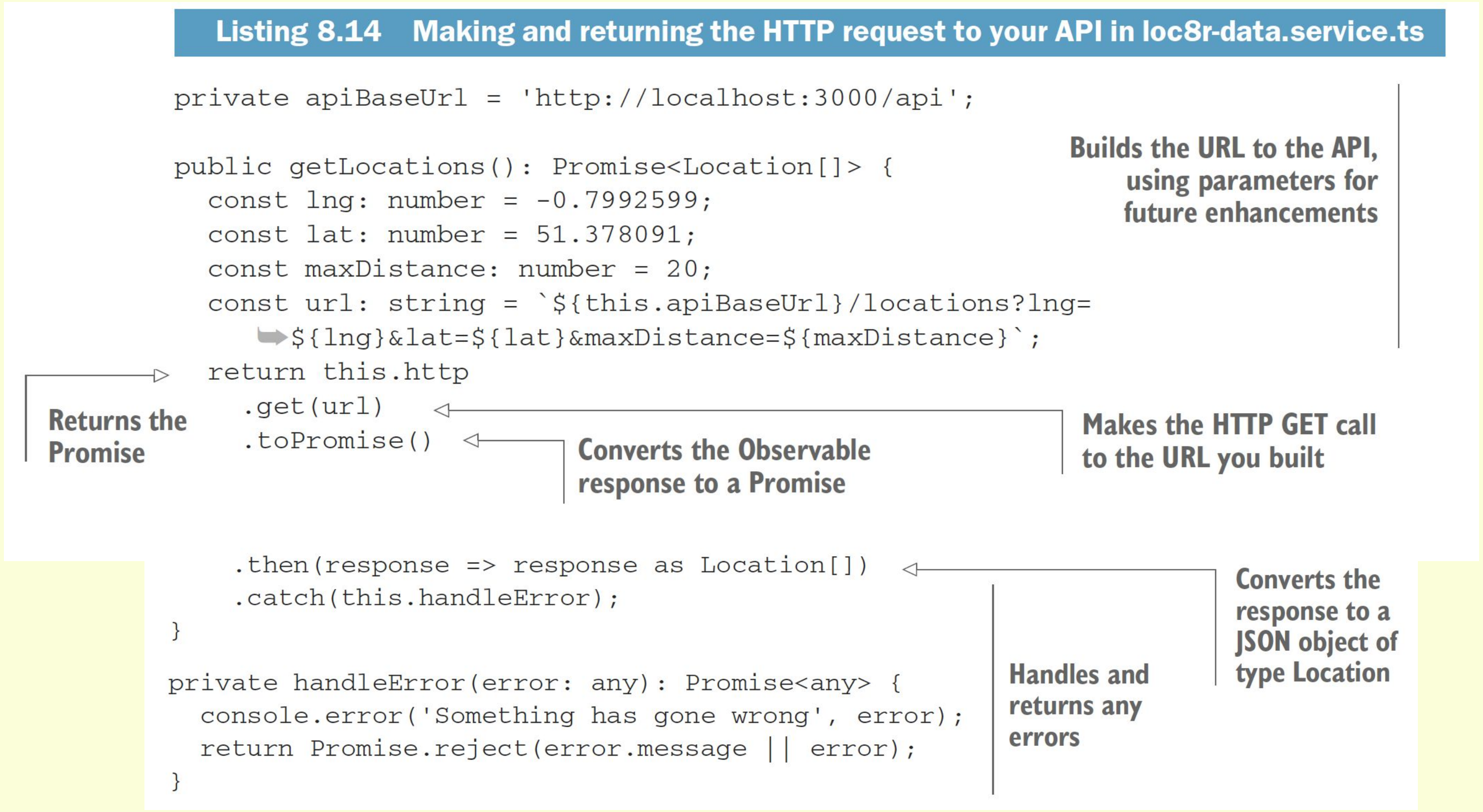


Observables and Promises

Observables and Promises are great ways of handling asynchronous requests. Observables return chunks of data in a stream, whereas Promises return complete sets of data. Angular includes the RxJS library for working with observables, including converting them into Promises.

There's much more to RxJS and Observables than we can cover here—enough for a whole book, in fact. Check out *RxJS in Action*, by Luis Atencio and Paul P. Daniels, to learn more (https://www.manning.com/books/rxjs-in-action).

Angular Capability - XHR



Listing 8.14 Making and returning the HTTP request to your API in loc8r-data.service.ts

```
private apiBaseUrl = 'http://localhost:3000/api';

public getLocations(): Promise<Location[]> {
  const lng: number = -0.7992599;
  const lat: number = 51.378091;
  const maxDistance: number = 20;
  const url: string = `${this.apiBaseUrl}/locations?lng=
    ➥${lng}&lat=${lat}&maxDistance=${maxDistance}`;
  return this.http
    .get(url)
    .toPromise()
    .then(response => response as Location[])
    .catch(this.handleError);
}

private handleError(error: any): Promise<any> {
  console.error('Something has gone wrong', error);
  return Promise.reject(error.message || error);
}
```

Builds the URL to the API, using parameters for future enhancements

Returns the Promise

Makes the HTTP GET call to the URL you built

Converts the Observable response to a Promise

Converts the response to a JSON object of type Location

Handles and returns any errors

Let’s get started with today’s lab!

- Today’s lab: Pipes and HttpClient within Angular
    - https://canvas.du.edu/courses/135260/assignments/1095021
    - In this lab we will be creating a custom Pipe in Angular that squares a number, and creating a custom Service that will return all the users we have stored in MongoDB using the HttpClient Service!