Contents

Day01

Introduction to NG19

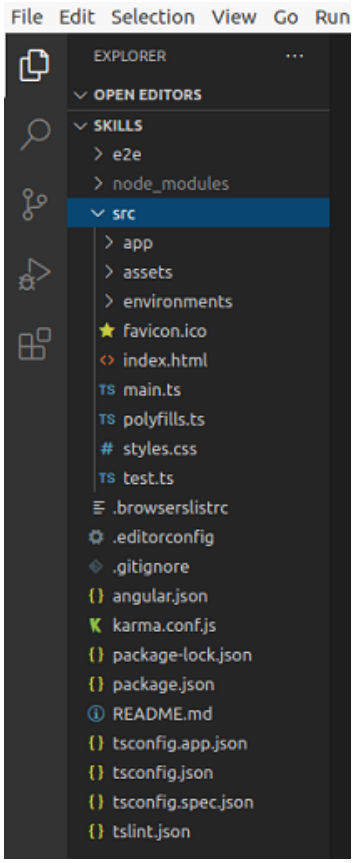
PART 01 – NG19 SETUP

This section assumes that you have already installed the latest Angular CLI. If you did not, please run the command **`npm install -g @angular/cli`** before proceeding. If you wanted to be sure you are installing V 19, try this line instead: **`npx install -g @angular/cli@19.0.0`**

1. From your root folder (Documents in my case), open a terminal window (or tab) to that folder and type the command **`ng new skills19 --skip-tests --skip-git`**
   (If you want your app to be NOT stand-alone pass the **`--no-standalone`** flag)

2. To choose CSS use the arrow keys on the keyboard, however CSS should be auto selected, just hit **Enter**. We will not be using Server-Side Rendering in this boot camp so enter **N** for any questions about this feature. Note, if you add the **`--style-css`** flag, you will NOT be asked about CSS again.

3. Open VS Code and open the skills folder that should have been created after #2 above. Open a terminal window inside of VS Code and type in **`ng serve`** or **`ng serve -o.`** If you browser does not open, open it manually and navigate to the URL: **`http://localhost:4200/`**



4. Open the file app.component.html in VS Code.

   This is the file that feeds the default page that shows up on the browser at port 4200. Remove everything except the **`<h1>`** tag which has the code **`Hello, {{title}}`** and the **`<router-outlet>`** tags. (around line 228)

   Leave these three lines in app.component.html

```
<h1>Hello, {{ title }}</h1>
<p>Congratulations! Your app is running.</p>
<router-outlet />
```

Should be around line 228 in the original boiler-plate app

Note, there is a starter file available, it's called skills19_starter.zip

5. There is already a property of the **`AppComponent`** class called **`title`**. Lets make it a signal and change its value from the template

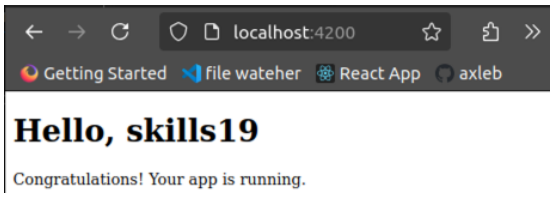
```
export class AppComponent {
  title = signal('skills19');
}
```

Note, you will need to import the *signal* function from @angular/core.

6. In the template, the interpolated title will have a yellow squiggly underline, just add a pair of parenthesis so that the signal value can be read properly:

```
<h1>Hello, {{ title() }}</h1>
<p>Congratulations! Your app is running.</p>
<router-outlet />
```

7. At this point, you should have a view on your browser similar to what is shown below:



At the end of most parts, there will be an accompanying zipped file. The zipped files will be different depending on what was changed. Since most of the changes are made in the app folder, this is the folder that will be zipped. This means that you must have a base application in order to use those unzipped files. For Part 1 however, you will get the entire app, so make sure you read the readme file for instructions on how to set it up. Usually the title of the zipped file will have a clue as to which folder is zipped up.

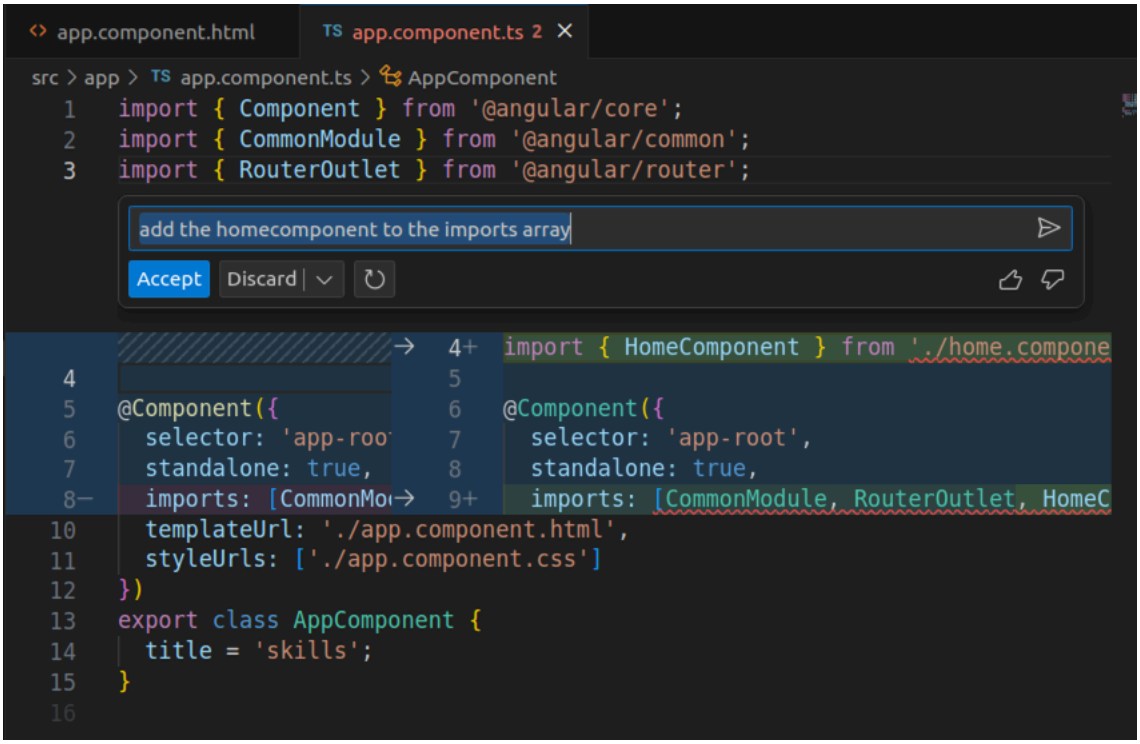
PART 02 – CONFIGURING NEW APP

1. Go back to the parent folder and look at the file app.component.ts there is a *selector* property with a value of **`app-root`**. Now open index.html under the original app folder and notice that between the **`<body>`** tags, this *name* appears between angle brackets. This means that where-ever Angular sees the directive **`<app-root>`** in the HTML, it will replace **`<app-root>`** with the template being pointed to, so app.component.html in this case. In other words, **`<app-root>`** will be replaced with new HTML content being constructed in the template. Following this path, we can build an entire application using just this one template.

2. Create a new *home* component by typing in the command in the terminal window: **ng g c home**
   If you did not want to have the CLI create a CSS file for you, add the flag -s. For example:
   ng g c home -s will not produce a new CSS file, only the .ts and .html files

3. Before we can use our new component, **home**, we need to import the component into our existing setup. We do this in app.component.ts file. In fact this **app** component now behaves like the parent for the entire application. Import the new **home** component there:

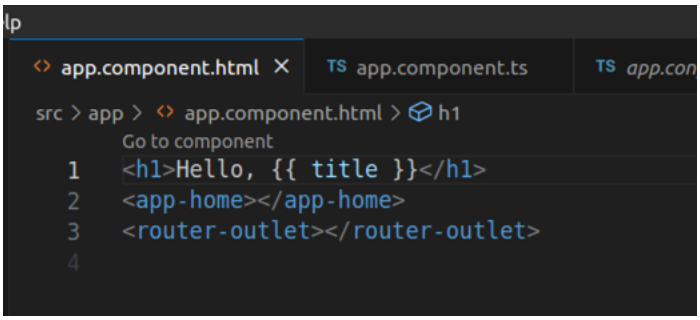
```
import { Component } from '@angular/core';
import { RouterOutlet } from '@angular/router';
import { HomeComponent } from "./home/home.component";
@Component({
```



4. Also update the imports array:

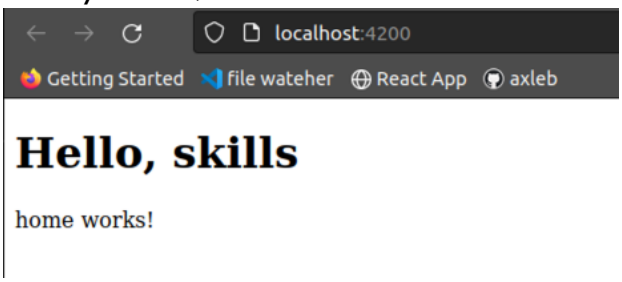
```
@Component({
  selector: 'app-root',
  standalone: true,
  imports: [RouterOutlet, HomeComponent],
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
```

5. Now go to the home folder and inside of home.component.ts file there is a selector with the name **app-home**, lets insert this name in the app.component.html file with angle brackets. So now, angular is building up components to deliver the *main* html page with the **<app-root>** directive.

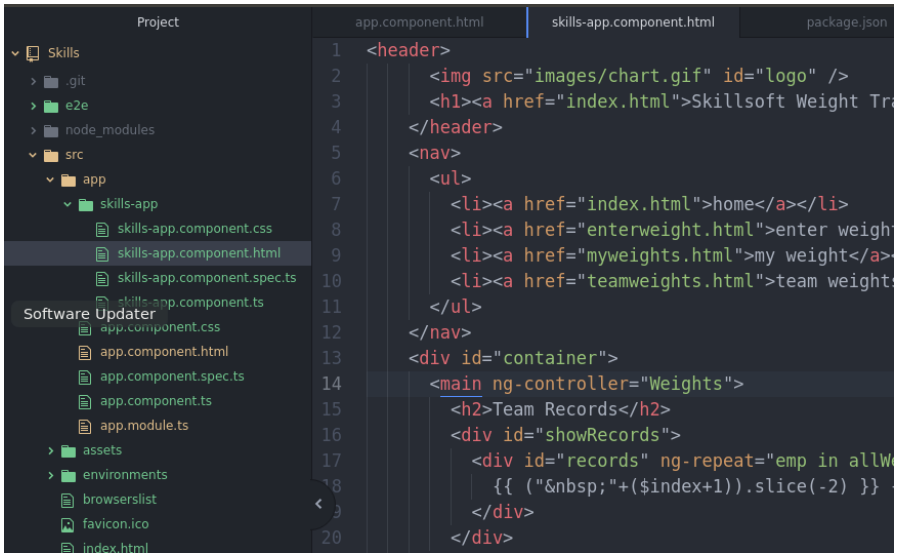


Note: **<router-outlet>** must appear last on the app.component.html page.

6. Once you save, the browser will reload the new page



We will use the original index.html file from a previous bootcamp as an example for the next step. Open that page in a **different** editor and copy everything between the **<body>** tags. Paste all this code in the home.component.html file replacing the original text that was there. You can find this index file in a folder called HTML.



During copy, do not copy the actual <body> tags and be careful to not copy any JavaScript tags.

7. Copy the image file from the original HTML folder into the new public folder of the new NG19 skills app. If you want you could create a new assets folder inside of the public folder. Use VS Code to help locate the image and get it ready for display on the browser when the app runs.
If you did put the chart.gif file directly into the public folder, then just use this type of path location:

```
<header>
  <img src="chart.gif" id="logo" />
  <h1><a href="index.html">Skillsoft Weight Tracker</a></h1>
</header>
<nav>
  <ul>
```

If you get an optimization error, you can try this: In the home component import the NgOptimizedImage module from @angular/common:

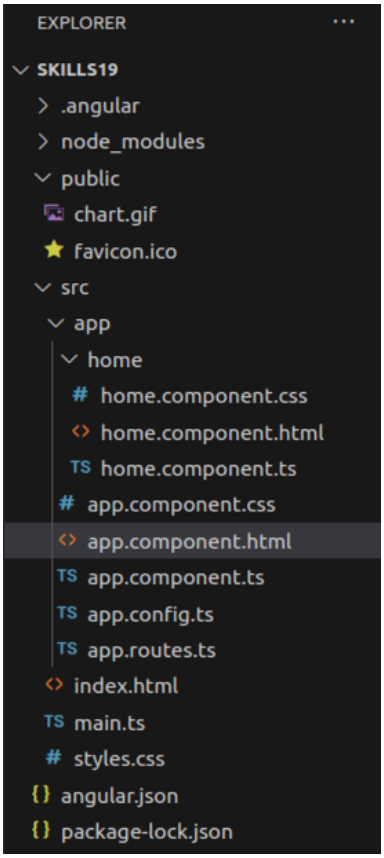
```
import { Component } from '@angular/core';
import { NgOptimizedImage } from "@angular/common";
@Component({
```

8. (Optional) Add that module to the imports array of the @Component() decorator function:

```
@Component({
  selector: 'app-home',
  imports: [NgOptimizedImage],
  templateUrl: './home.component.html',
```

Do this only if you get the optimization error.

9. Copy the original CSS file and replace the NG19 CSS file in the src folder. Spin the app to see what it looks like. You can remove the original line from the app.component.html file.



10. This is what my folder structure looks like:

The public folder now contains the chart.gif file and the styles.css is now the one we got from the HTML folder in the GitHub download. Also the home.component.html file now contains the HTML content from the original index.htm file, also from the HTML zipped file you got from GitHub.

You may now remove the the first two lines from the app.component.html file. Your app.component.html file should now contain just the **<app-home>** and **<router-outlet>** custom tags.

PART 03 – ROUTING

1. At this point we do not have true routing, we hard coded our home page to show up on the first hit to :**4200**, we will fix this by changing app-routes.ts. In previous versions you were asked if you wanted routing at the app creation point, so at the time where you used **ng new.** In version 19 we are provided with the routes file by default.

2. In the app.routes.ts file, enter the home route in the **Routes[]** array as a JavaScript object, but import the component first:

```
import { HomeComponent } from './home/home.component';

const routes: Routes = [
  { path: 'home', component: HomeComponent }
];
```

   You can allow VS Code to guide you to finding the HomeComponent or import it manually

3. Now we can remove the **<app-home>** element from app.component.html. So this file should only have the **<router-outlet>** element, nothing else. In NG19 when the Angular home page refreshes we do not see the home page we built using the old HTML. You must now navigate to localhost:4200/home

4. While we are on this topic lets add a default route, so if the user goes to just :**4200**, they should see the home page. So back in app-routes.ts add the default path as shown below:

```
import { HomeComponent } from './home/home.component';

const routes: Routes = [
  { path: '', redirectTo: '/home', pathMatch: 'full' },
  { path: 'home', component: HomeComponent }
];
```

   Notice the comma after the first path. Also, this new path is above the home path.

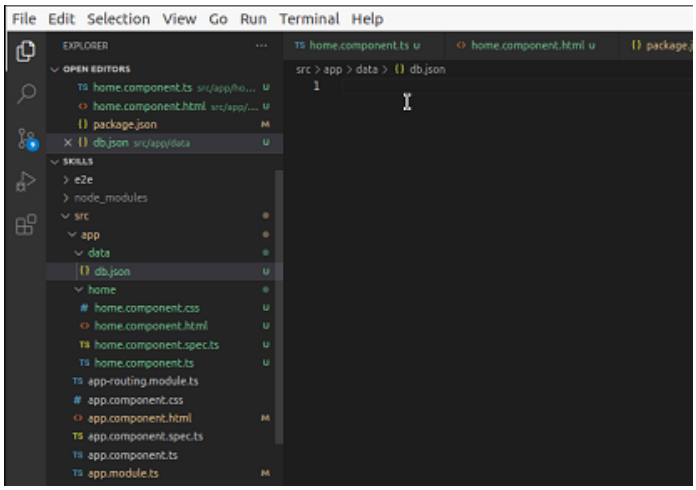
Note: Once you make this change, you can remove the HomeComponent from the app.component.ts file

PART 04 – JSON SERVER

We would need a mock server so that we can make API calls. Install the **JSON** Server using the command: **`npm install json-server --save-dev`**

Make sure that you are in the skills19 folder when you do this.

1. If you have VS Code, create a new folder under the app folder called data. Inside of the data folder create a new text file called db.json



Note: there is a db.json file in the code for today. It already has data in it.

2. There are several ways to create **`json`** files but lets follow the example below. First we name our mock database, **`employees`** in this case and then point that to an array of employees:

```
{
  "employees":[ ]
}
```

Note, you must have the enclosing curly braces to encapsulate the employees

3. Enter the first employee like this (use your own name but the password can be something simple):

```
{
    "employees":[
        {
            "username":"Axle",
            "password":"1234"
        }
    ]
}
```

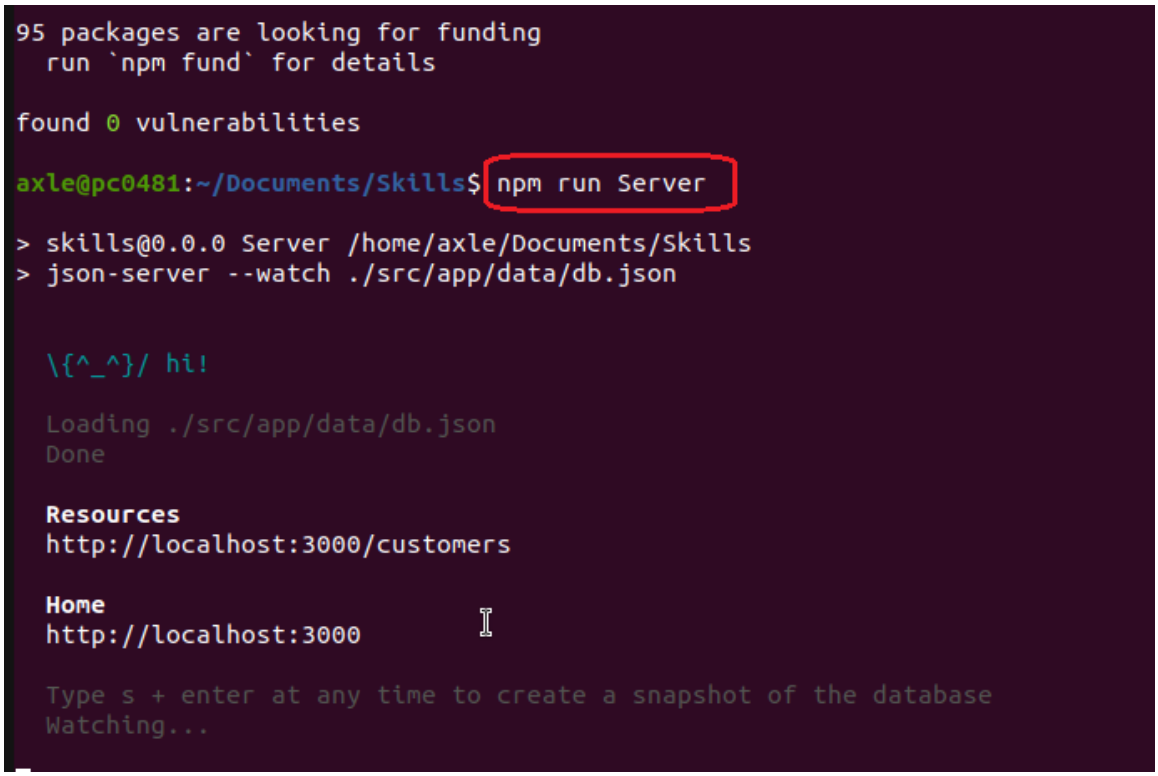
This is NOT a secure site, yet, so use simple passwords for now. You may use the db.json file provided for today.

4. Configure the package.json file to run our server. In package.json go to the scripts section and add a new script as shown below:

```
  "version": "0.0.0",
  "scripts": {
    "ng": "ng",
    "start": "ng serve",
    "build": "ng build",
    "test": "ng test",
    "lint": "ng lint",
    "e2e": "ng e2e",
    "server": "json-server --watch ./src/data/db.json"
  },
  "private": true,
```

Remember to insert a comma at the line above. Notice that the server is watching a file called db.json, we created this file in #3 above. **Note, if you are using json-server v. 1.0.0 you do not need the watch flag.**

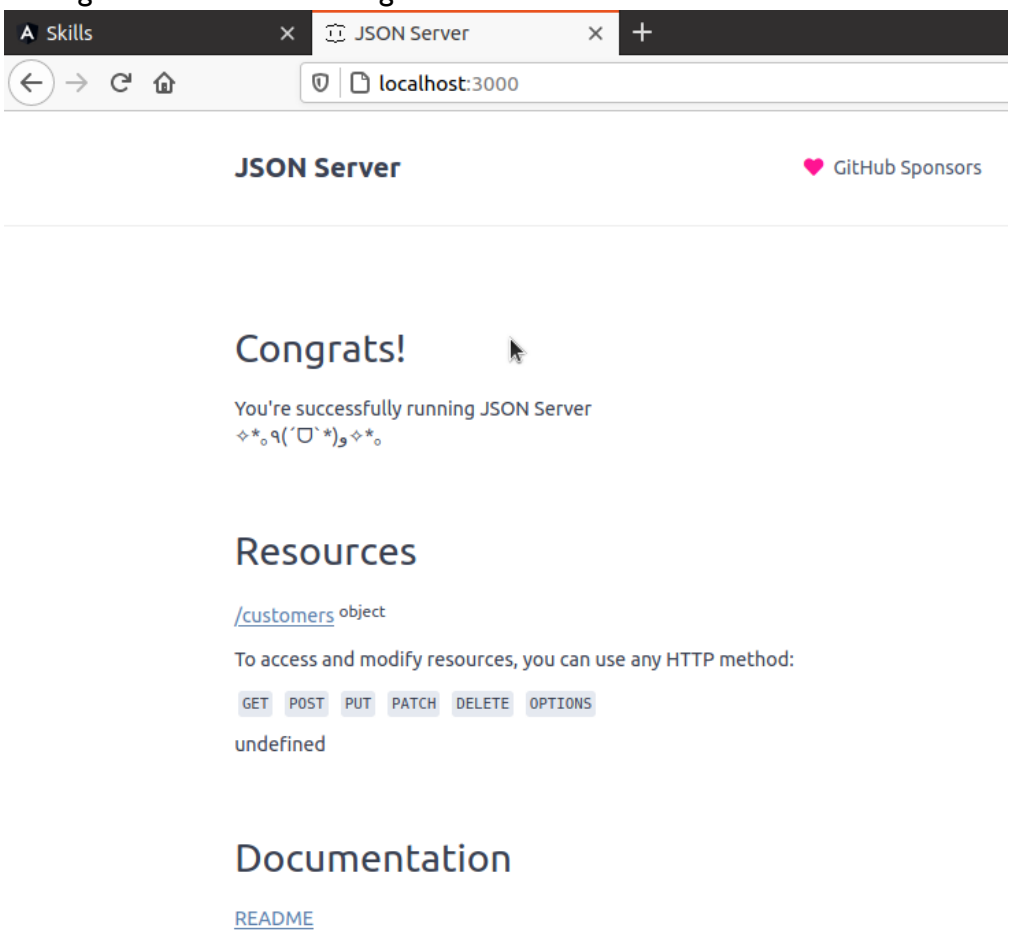
5. Start the server by going back to a terminal window and run the command: **`npm run Server`. Note, do this in a separate terminal window.**



The computer should respond with a message and a location of where our data can be accessed with a browser, localhost:3000 in this case. You can start this server from a second terminal window in VS Code. Once the service starts, leave it running. To stop it use **`CTRL-C`**

If you want, you can add this flag --4201 and the database will be on the port next to 4200

6. Now go to that location using a browser



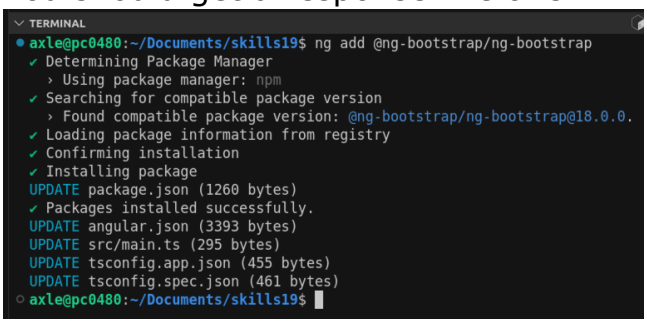
Notice that under **Resources** the server found our *employees* database

Note to stop any process in Linux just hold down the **`CTRL`** button and then hit the letter C on the keyboard.

PART 05 – ADD BOOTSTRAP

1. Run the following command in a terminal window to add Bootstrap, make sure your terminal window is pointing to your application folder, skills:
   `ng add @ng-bootstrap/ng-bootstrap or npm install bootstrap@latest`
   Respond with a "Y" if asked about installation.

   You should get a response like this:



2. Do not change the angular.json file, this line will be added for you automatically:

```
            "assets": [
              {
                "glob": "**/*",
                "input": "public"
              }
            ],
            "styles": [
              "node_modules/bootstrap/dist/css/bootstrap.min.css",
              "src/styles.css"
            ],
            "scripts": []
          },
          "configurations": {
```

   The home page will look a little different but no major changes. If the line highlighted in green is not added to your angular.json file, add it manually.

PART 06 – NG19 FORMS

We will create a simple *register* form and a *login* form to test our server and at the same time learn valuable skills in Angular 19

1. Create a new *register* component like you did for the *home* component, so from a terminal window:
   `ng g c register`

2. Copy all the code from home.component.html file into register.component.html. Remove all of the code between the **`<main>`** tags. Replace all the code you removed with the code highlighted below:

```
<form>
<div class="form-group">
  <label for="username">User name</label>
  <input type="text" class="form-control" id="username">

</div>
<div class="form-group">
  <label for="password">Password</label>
  <input type="password" class="form-control" id="password">
</div>
<button type="submit" class="btn btn-primary">Submit</button>
</form>
```

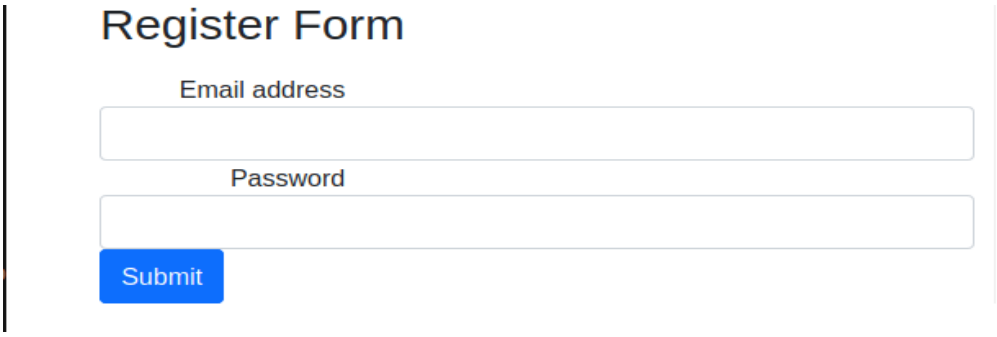
3. Follow the steps from Part03 to add a new route to your register component in the app.routes.ts file:

```
import { Routes } from '@angular/router';
import { HomeComponent } from './home/home.component';
import { RegisterComponent } from './register/register.component';
//
export const routes: Routes = [
  { path: '', redirectTo: 'home', pathMatch: 'full' },
  { path: 'home', component: HomeComponent },
  { path: 'register', component: RegisterComponent },
];
```

4. Wrap the **form** inside of a pair of ** ** tags and add the class of **container** to this new **div**, also add an **<h2>** tag with class of **pb-2** and give it a **title**:

```
</nav>
<div id="container">
  <main>
    <div class="container">
      <h2 class="pb-2">Register Form</h2>
      <form>
        <div class="form-group">
```

Note, that the name of the fields on this form match the database.



Test by going to: http://localhost:4200/register

7. Add the **FormGroup, FormControl** and **ReactiveFormsModule** module to the register component:

```
import { Component } from '@angular/core';
import { FormGroup,FormControl, ReactiveFormsModule } from '@angular/forms';
```

8. Add the **ReactiveFormsModule** module to the **@Component** decorator:

```
@Component({
  selector: 'app-register',
  standalone: true,
  imports: [ReactiveFormsModule],
  templateUrl: './register.component.html',
  styleUrl: './register.component.css'
})
```

9. Create a new **FormGroup** with two **FormControls** to match our template:

```
export class RegisterComponent {
  registerForm = new FormGroup({
    userName : new FormControl(),
    password : new FormControl()
  });
```

Note the use of colons, curly braces and commas.

10. Create a new function to handle form submit:

```
    });
  }
  onSubmit() {
  }
```

11. In the template, add the **formGroup** directive:

```
<h2 class="pb-2">Register Form</h2>
<form [formGroup]="registerForm">
    <div class="form-group">
```

Notice that the directive points to the same name we created in step 6. This is called property binding and it tracks the changes in the form controls.

12. Add each **FormControl** to their respective **<input>** tag:

```
 <div class="form-group">
   <label for="username">User name</label>
   <input type="text" class="form-control" id="username"
formControlName="userName">
    </div>
    <div class="form-group">
     <label for="password">Password</label>
      <input type="password" class="form-control" id="password"
formControlName="password">
     </div>
```

13. In the template we need to handle form submission, for now just point to a function and we will create this function later:

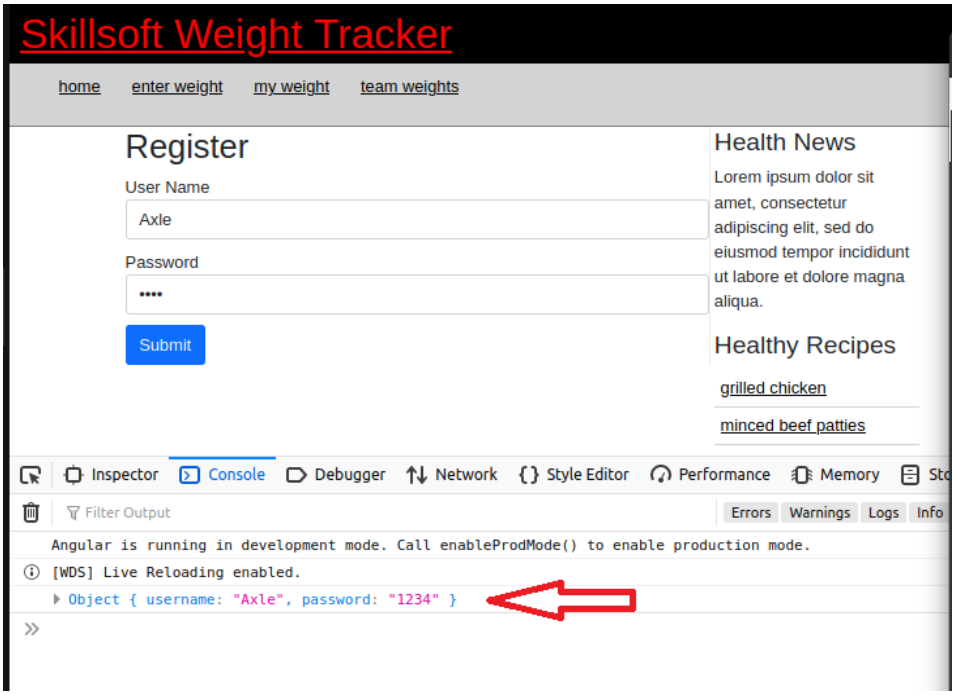
```
 <h2 class="pb-2">Register Form</h2>
   <form [formGroup]="registerForm" (ngSubmit)="onSubmit()"></form>
      <div class="form-group">
```

The **formGroup** directive is responsible for tracking the overall value of the form , which contains the values of all of its form fields. It will also keep track of the overall validity and state of the form, which is dependent on the validity state of its form fields. This is called event binding. Here **ngSubmit** is the event listener.

14. Complete the **`onSubmit()`** function in the **`RegisterComponent`** and for now simply output the field values in the console window:

```
onSubmit() {
   console.warn(
      `You entered : userName:
      ${this.registerForm.controls['username'].value},
      And password:
      ${this.registerForm.controls['password'].value}`,
   );
 }
```

Test the form, see results below



Note the image on the left is the result of printing the entire form using this line: **`console.log(this.registerForm.value);`**

Appendix A – Install Angular 19 on Linux Ubuntu 22

First install NodeJS if it is not already installed, but update the profile first, so:

1. sudo apt update
2. sudo apt install nodejs
3. sudo apt install npm

At this point you can install the Angular CLI

1. npm install -g @angular/cli

Verify that NodeJS, NPM and Angular was installed, run these commands:

```
nodejs -v
npm -v
ng --version
```

The entire process could take between 5 to 15 minutes depending on your system and internet connection

If you are using VS Code you may get a message to install **Angular Language Service**, please install it.

Appendix B – Angular Architectural Concepts

Angular uses the concept of modules (Ng Modules) into which components are placed. There are built-in modules that come with the installation of Angular. Some of these modules we will be using in the course include the HttpClientModule and the FormsModule. An Ng Module is just a TypeScript class with an @NgModule decorator. Most decorators add metadata to the class and in come cases functionality. By default we get the AppModule to help us kickstart our customized development.

Decorators may contain declarations, exports, imports, providers and bootstrap classes. Declarations handle views like component views and directive views. Export classes ensure that a class can be accessed by other classes. Imports exposes modules required by a class. Providers handle Services which are mostly logic required by some class. Bootstrap is in the root component and provides the initial view.

There are several JS modules used as libraries in an Agular application. Libraries such as @angular/core, @angular/router and Material are used to add functionality. These libraries are simply imported.

Components comprise of a TypeScript class, some kind of HTML template for display and a stylesheet. A component will have the @Component decorator to define it as a component.

A customized component will usually have a selector which is an instructor to Angular to insert this particular component where ever it finds the selector. The selector tag within the HTML is usually written as <app-root></app-root>.

The templateUrl will point to an html file which acts as the template for a component. styleUrls of course does the same for CSS files.

Directives:

Directives are instructions that instruct the DOM as to how to place your components and business logic in the Angular project. Directives are just JS class which are declared as @directive. There are 3 directives in Angular: Component Directives, Structural Directives and Attribute Directives.

Component Directives look like this @Component. They contain the detail of how the component should be processed, instantiated and used at runtime.

Structural directives start with a * sign. These directives are used to manipulate and change the structure of the DOM elements. For example, *ngIf and *ngFor.

Attribute directives are used to change the look and behavior of the DOM elements. For example: ngClass, ngStyle etc.

The main building blocks of Angular are:

- Modules
- Components
- Templates
- Services
- Metadata
- Directives
- Data binding
- Dependency injection

Here are a few Angular CLI commands that we will be using

add	Used to add support for an external library to your project.
build	Will compile an Angular app into an output directory named dist/ at the given output path.
generate	Generates and possibly modifie files based on a schematic.

new Creates a new workspace and a boilerplate Angular app.

run Runs an Architect target

serve Builds and serves your app via http, also re-compiles when it detects changes.

test Executes unit tests in a project

update Updates your application and its dependencies

Angular 19 File Explanation

- src folder: all the action takes place here
- app folder: all the files, that support app components.
- app.component.css: the cascading style sheets code for your app component.
- app.component.html: the template html file connected to app component and is used by angular to do any data binding.
- app.component.spec.ts: use the command ng test to see this file in action. It is a unit testing file related to app component. All files that have .spec in the middle is a test file
- app.component.ts: probably the most important typescript file which contains the view logic driving the component.
- app.module.ts: a file which includes all the dependencies for the entire website. This file defines any modules to be imported, components to be declared and the main component to start the app
- karma.config.js: This file specifies the config file for the Karma Test Runner, Karma has been developed by the AngularJS team which can run tests for both AngularJS and Angular 2+
- main.ts: As defined in angular.json file, this is the main ts file that will first run. This file bootstraps (starts) the AppModule from app.module.ts , and it can be used to define global configurations.
- polyfills.ts: This file is a set of code that can be used to provide compatibility support for older browsers. Angular 7 code is written mainly in ES6+ language specifications which is getting more adopted in front-end development, so since not all browsers support the full ES6+ specifications, pollyfills can be used to cover whatever feature missing from a given browser.
- styles.css:/ This is a global css file which is used by the angular application.
- tests.ts: This is the main test file that the Angular CLI command ng test will use to traverse all the unit tests within the application and run them.
- tsconfig.json: This is a typescript compiler configuration file.
- tsconfig.app.json: This is used to override the tsconfig.json file with app specific configurations.

tsconfig.spec.json: This overrides the tsconfig.json file with app specific unit test configurations

APPENDIX C – ANGULAR DIRECTIVES

Directives are functions used reinforce HTML, make it do much more than what it was designed for. These directives have names like **ngFor* and *ngStyles* but can be any name you make up and they are specific to an HTML element, an attribute or class

DOM manipulation directives are called attribute or structural directives.

Attribute directives manipulate the DOM by changing its behavior and appearance.

Using the Existing Angular Directives in an example:

```
<div [ngStyle]="myStyles">
  Content goes here
</div>
```

You can now define myStyles somewhere in your .ts file as a function.

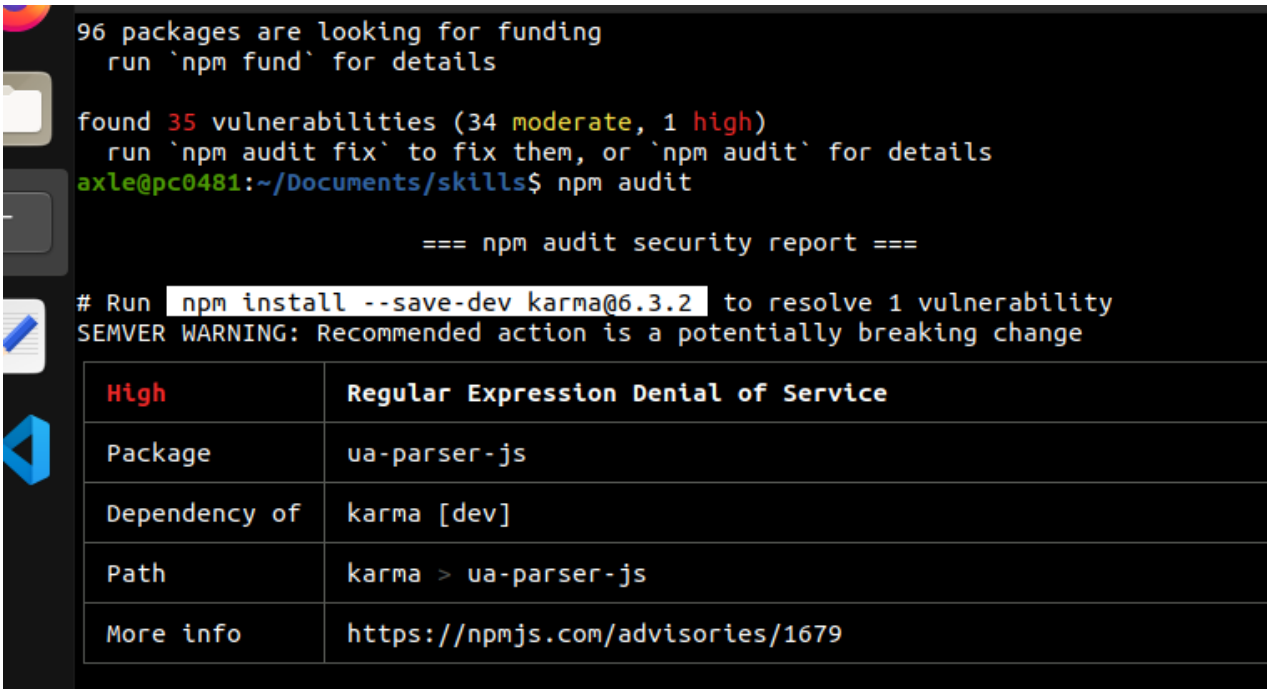
Structural directives are meant to create and destroy DOM elements and usually start with the * character such as **ngIf*

```
<div *ngIf="condition">Content to render when condition is true.</div>
```

Components are also special directives

Appendix D – Installation Issues

If you get issues while installing json-server run npm audit to see what might be stopping the installation and how you might be able to fix it:



So in this case, I installed karma@6.3.2

Appendix E – @NgModule

Angular 19 still uses modules if you want them.

Declarations are used to declare components, directives, pipes that belong to the current module. Think of a namespace, declarations create a namespace so all the components in this @NgModule are available to each other in a public but protected way.

Imports (and exports) work just like in other programming languages. They are used to import supporting modules like FormsModule, RouterModule and the CommonModule.

Providers are used by modules for accessing the services required by components, directives. The process is known as injecting services into the component.

The **bootstrap** property simply points to a component that will be used to start the application.