HTTP 5302 Lab 5
June 7/8 2018

Objective

Practice the MVC development model with Angular

Description

This lab will reinforce the MVC structure of Angular 2+ by constructing the models, services, and components of the framework.

Instructions

Start a new Angular project and set new repository origin for this lab called “Web Trends Lab 5”. As a reminder, you do not need to initialize a git repository when you make an Angular project with the cli.

When finished with each part of these instructions, commit that part to git. By the end of the lab, you should be able to see the differences of code and files on your git repository when adding each part of an Angular 2+ project.

Part 1: Model

Construct a model for a person with the following properties:

- id: number
- firstName: string
- lastName: string
- dateOfBirth: date

Add a method to this model called getAge() which will return an accurate age based on the dateOfBirth.

```
ng generate class person
```

Notice that a single file is produced. No modifications have been made to the other files. Is this an advantage or disadvantage when working with git?

Part 2: Service

Construct a service that will manage a list of people based on the Person class made in Part 1. Make three people in this service by setting up an array with three Person objects.

```
ng generate service people
```

Like models, services are made with new files without altering existing files.

The service will need direction on how to output its contents. Define a method that will return the array containing the Person objects.

HTTP 5302 Lab 5
June 7/8 2018

Part 3: Component

When making new items that will alter existing files, it's good practice to make a branch in git:

```
git branch guestlist_component
```

Making components will alter app.module.ts.

Generate a guestlist component that will list the people in people.service.ts:

```
ng generate component guestlist
```

Now checkout app.module.ts. New components are registered in app.module.ts and you should see a new `import` link underneath `import { AppComponent }`.

Clear app.component.html and replace it with the selector for the new guestlist component.

Injecting a service into a component

"Inject" your service, and therefore its contents, by importing it as if you would import a class. Angular makes it easy to identify services you've defined by appending 'Service' to your services:

```
import { PeopleService } from [path to service]
```

Angular suggests as best practice to declare a service inside a component using the component's constructor.

```
contructor(private peopleService: PeopleService){

}
```

Many services can be used in a component, just simply import them and declare an instance of them inside the constructor.

Now let's use the service. Set up an undefined property typed to an array of Person. Components have a built-in function called 'ngOnInit()' that runs when the component loads. Use the declared instance of the service and its methods to retrieve the contents of the service as defined in the service's definition:

```
this.peopleService.getPeople().subscribe(data => this.guests = data)
```

Consuming services *accurately* requires the use of the subscribe method. In other words, data will be retrieved *when its ready* while the rest of your application continues its operations. This is asynchronous loading and Angular makes it easy to handle asynchronous actions of your application.

Manipulating a component's view

Using Angular's HTML templates, loop over the guests array using *ngFor. You can choose the format in which you want to display them whether that be inside a list or a series of divs. This will create multiple elements for each of the items in the for loop.

```
<element *ngFor="loop expression">
```

HTTP 5302 Lab 5
June 7/8 2018

Part 2: User interactivity

Using input

Using text input from the user requires multiple layers from Angular and it has many methods to implement forms. To use the forms here, another *module* must be included in the project, the FormsModule. In apps.module.ts import the FormsModule from @angular/forms, then add it to the 'imports' array below.

```
import { FormsModule } from '@angular/forms';
```

You may now use the following to *bind* user input to a model

```
[(ngModel)]="model.property"
```

The way ngModel works is the value of the element using the ngModel attribute *sets* the value of the model that it is associated with but can also be set by the model that it is being used with:

guestlist.component.ts

```
guestSearch: Person = new Person();

guestSearch.firstName = "Lee"
```

guestlist.component.html

```
<input [(ngModel)]="guestSearch.firstName" placeholder="firstName">

<div>{{ guestSearch.firstName }}</div>
```

In this code, firstName has already been set and will be expressed whenever it's called. The input will also set that value, changing the value of firstName and expressing that new value every time it is changed.

Event listeners (click)

These are some of the more straight forward aspects of Angular interactivity. In a component's HTML, assigning (click) to an element will execute the method defined in that component's typescript:

<button (click)="findPeople(params)">Search</button>

Challenge

Set up the page so that you can search for people from the people service made above.

Approach it using MVC:

1. Set up a method inside the people service that will return an array of people based on a name parameter to search on.
2. Set up a method inside guestlist.component.ts that will retrieve the method from 1.
3. Set up a button and input(s) that are bound to a model using ngModel.
4. Clicking when clicking the button, set it to the method defined in 2.

gitHub.com/rockyshiba/school contains an example of this lab.

This lab is worth 5% of your grade. Full grade will be award upon attempt of this lab.