Ex. Angular Architecture

Intro

Understanding the **Architecture** of **Angular** is understanding 70% of the framework. This is the most important lesson of the course and everything we will teach here we will exercise to make sure we really understand everything. If you manage to understand the architecture properly, everything else will be a breeze to understand.
Important: the idea is to also finish the exercise at a good pace, so if you get stuck make sure to call Yariv for help. **Don't get stuck on an exercise more than 10 minutes! Call for help.**
The idea of the task is to create a really simple **ToDo** application.

Modules

1. In your workspace create a directory called: **angular-architecture-ex**
2. cd into that directory and init **npm**
3. install **webpack** and configure it's entry point, output, and loaders (ts-loader)
4. create a **tsconfig.json** file using the **tsc --init** command, important things to note:
    a. in the **lib** array add **"es6"**
    b. experimentalDecorators have to be set to **true**
    c. emitDecoratorMetadata have to be set to **true**
    d. skipLibCheck has to be set to **true**
    e. add **node_modules** to the **exclude** array
5. Create two modules for your app
    a. root module called **AppModule**
    b. another module called **UserModule**
    c. the root module will use the **UserModule** so add it to the **imports** array of the **AppModule**
    d. bootstrap the root module to the dom
    e. For now there are no components and no services
6. make sure your can run webpack and there are no errors and the new file is created in the output folder
7. call Yariv to check your work

Components

EX1: Hello World

1. Create an **Angular Component** called **AppComponent**
2. This component should only have a **selector** and a **template**
3. the **template** should display an **hello world** message
4. Add the component to the **AppModule** we created earlier (also to the **bootstrap** array of the module)
5. create an **index.html** file with the selector of the component you created
6. add a **script** tag to the **index.html** file referencing the file compiled by webpack
7. run webpack to create you compiled **js** file
8. run a local web server by installing the package **http-server**
9. Open your browser and make sure you see the hello world message

EX2: RegisterComponent

1. This component will belong to the **UserModule** you created earlier so make sure to add the component in the **declarations** array and the **exports** array.
2. the component will display a register form with the following inputs:
    a. first name
    b. last name
    c. email
    d. username
3. add the selector of this component to the **AppComponent** and make sure you can see the form you just created (for this to work you have to make sure the **AppModule** has the **UserModule** in the **imports** array)

Services

User Model

1. create a user model containing the properties:
    a. firstName : string
    b. lastName : string
    c. email : string
    d. username : string
2. create a getter for **fullName** that will print the full name of the user

UserService

1. We are going to use **rxjs Subject** to **broadcast** messages to **subscribers**
    a. Create a **BehaviorSubject** that is initiated with null and will contain the registered user

2. create a **register** method that will receive a user model as argument and will add the user to the **BehaviorSubject**
3. add the service you created to the **UserModule**

Connecting to the RegisterComponent

1. use **ngModel** directive to connect the inputs of the component to public properties of the class
2. in the **constructor** of the component, inject the **UserService** as a private property of the class
3. add **ngSubmit** to the form and attach it to an event in the component class
4. in the submit event create a new user model and use the **UserService register** method to register that user

Create LoggedAsComponent

This component will display a message of the currently logged in user. If there is no logged in user, the component will display the message: **hello stranger!** otherwise it will display a greeting message for the registered user: **hello Yariv Katz**

1. add the component to the **UserModule**
2. call the selector of the component in the **AppComponent**
3. inject the **UserService** in the component and **subscribe** to the **BehaviorSubject** of the component
4. the component should have a property of the current user logged in, this property should be of type **User model**.
5. When you get a new user from the **BehaviorSubject** you update the user property
6. The template of the component should display the message based on the user property

Directives (really hard!!!)

1. using **npm** install **jquery** and **ladda**
2. you need to add **jquery** using **webpack require**
3. You need to require the **ladda** package after **jquery** (try to do 2,3 using webpack and not the cdn of those package)
4. The directive selector is a button with the attribute **ladda-button**

5. the directive will get an **isLoading** as input. example:

HTML
<button ladda-button [isLoading]=”booleanExpression” >Submit</button>

TS
@Input() set isLoading(value: boolean) {
...
}

6. when the is loading is true you will use ladda to display a loading scroll bar
7. attach the directive to the submit button of the form you created earlier make sure you change the is loading with a **setTimeout** so we can see the loading sign for a few seconds

Pipes

create a pipe that’s when applied to a string will change the string to be Capital letter, Lowercase, Capital … etc
Apply this pipe on the **LoggedAsComponent** when you display the name of the user

Conclusion

In this exercise we practiced the pillars of angular architecture.
This lesson is the foundation on which the entire framework relies on, so it’s super important to understand and exercise the architecture of the framework.