Worldline Angular Workshop
18 – 21 March 2019

1 Labs

1.1 Software installation

1. Node/NPM:
https://nodejs.org/en/download/

2. TypeScript:
https://www.typescriptlang.org/#download-links
To install, make sure that Node has been installed and type in a shell terminal:

```
npm install -g typescript
```

3. Install Angular CLI - Follow Steps 1 - 3 at:
https://angular.io/guide/quickstart

Type in a shell terminal (with admin privilege – Windows or sudo – Mac/Linux) :

```
npm install -g @angular/cli
```

You can check the current version of angular with:

```
ng --version
```

If you have serious problems at any point, you can uninstall with:

```
npm uninstall -g @angular/cli
```

https://stackoverflow.com/questions/39566257/how-to-uninstall-angular-cli

and then reinstall it again.

4. Install the Augury Chrome extension
Go to
https://chrome.google.com/webstore/category/extensions
and search for Augury and add it to the Chrome browser that you will be using for the workshop labs

5. Visual Studio Code (optional):
https://code.visualstudio.com/

This is the code editor that will be used during the workshop. Feel free to use whatever editor/IDE that you are familiar with. Other popular IDEs include Atom, Sublime Text, Webstorm, Brackets.

1.2 References

1.2.1 HTML

https://www.w3schools.com/tags/ref_byfunc.asp
https://www.w3schools.com/tags/ref_standardattributes.asp
https://www.w3schools.com/tags/ref_attributes.asp

1.2.2 HTML DOM

https://www.w3schools.com/js/js_htmldom.asp
https://www.w3schools.com/jsref/dom_obj_all.asp
https://www.w3schools.com/jsref/dom_obj_event.asp

1.2.3 CSS

https://www.w3schools.com/css/default.asp
https://www.w3schools.com/cssref/css_selectors.asp

1.2.4 Javascript

https://www.w3schools.com/js/default.asp
https://javascript.info/

1.2.5 Angular

https://angular.io/docs
https://angular.io/api

1.3 TypeScript

We will compile the Typescript files using `tsc` and execute the resulting Javascript files using Node.
https://www.keycdn.com/blog/typescript-tutorial#Part-2-Compiling-to-JavaScript

The sample files are in `tsdemo`

We can compile individual files and execute them one at a time using Node:

```
tsc myfile.ts
node myfile.js
```

Alternatively we can run the compiler in watch mode:

```
tsc -w *.ts
```

to monitor all the Typescript source code files in our directory and trigger recompilation on changes.

With the property:

```
    "include": [
      "*.ts"
    ]
```

in `tsconfig.json`, we can just type `tsc` at the shell prompt to compile all the Typescript source code files.

More details on using the Typescript compiler:

https://blog.angularindepth.com/configuring-typescript-compiler-a84ed8f87e3
https://www.typescriptlang.org/docs/handbook/compiler-options.html

1.4 Angular lab overview

Create an empty folder (which I will call the working folder) where you will generate the project folders for the various Angular apps that you will build in this workshop (e.g. `C:\code`). DO NOT generate your project folders directly in your root drive C:\

You should be able to see a `workshop labs` folder in the zip file that you downloaded from GitHub. The various subfolders in `workshop-labs` contain information related to the various Angular projects that we will be building, either from scratch or from an initial app configuration. The creation of the final app will proceed in a step-wise fashion so that the various key features of the Angular framework can be clearly demonstrated at specific steps.

Each subfolder contains the following additional folders:

a) `xxx-start` (optional) – If the project is going to be built from an initial app configuration, the source code for this initial app will be here
b) `additions` – This contains the modifications to the various project source files as the app is gradually built up in a step-wise manner. For e.g. `xxx.v1.html`, `xxx.v2.html`, etc represents successive changes that are to be made to the `xxxx.html` in the actual project folder.
c) `xxx-final` – The complete source code for the final working project for reference

Both `xxx-start` and `xxx-final` are missing the crucial `node_modules` subfolder which contain all the dependencies (Javascript libraries) that your Angular app needs in order to be built properly. These dependencies are specified in the `package.json` file in the root project folder. They are omitted in the project commit to GitHub because the number and size of the files are prohibitively large.

In order to build and run the app defined in either `xxx-start` or `xxx-final`, you will need these files. There are two ways to accomplish this:

- Copy the relevant project folder (`xxx-start` or `xxx-final`) to the working folder that you use for creating actual apps, and run `npm install` in a shell prompt in the root project folder. This can take quite a while to complete (depending on your broadband connection speed), and will need to be repeated for each individual project folder.

- An alternative is to reuse the dependencies of an existing complete project in the working folder as most of the projects for our labs use the same dependencies. In this case, copy over `xxx-start` and `xxx-final` to the working folder. Then copy the `node_modules` subfolder from the existing complete project over to here, and you should be able to immediately start the app. If you encounter any errors, this probably means that there are some missing dependencies or Angular version mismatch, in which case you will then have to resort to `npm install`.

1.5 Intro: Basic app with binding

1.5.1 stock-market

This application is built from scratch

Step 1: Creating a new Angular app

From a shell terminal type:

```
ng new stock-market
```

Press enter to accept the defaults for all the questions that come up (if you are using Angular 7)

The newly created `stock-market` folder is the root project folder for the Angular App. Change to this directory in an independent shell terminal (or open the folder using VS-Code and open a terminal here) and type:

```
ng serve
```

to start Angular's customized web server that will serve your Angular app dynamically

Step 2: Generating a component

In `src/app`

```
ng generate component stock/stock-item
```

Step 3: Basic interpolation and addition of child component

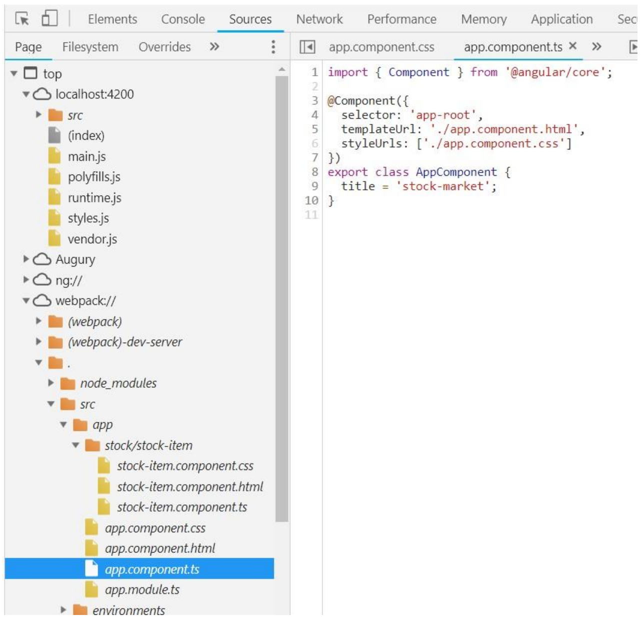
`app.component.v1.html`

Step 4: Accessing DOM and individual component source code files

View the contents of the main page (`index.html` in `src/app`) -> Right click and select `View page source`. Notice that it only contains a single element in the body `<app-root>`. The DOM tree visible in the Elements tab of the Developer tools is generated from the various scripts (`runtime.js`, `polyfill.js`, etc), and you can view these contents by clicking on them in the View page source window. The transpiled Javascript for the components of the app can be viewed in `main.js`.

You also view the individual source code files for the various components by going to `webpack://.src` in the Sources tab of the Developer tools.

Any run-time error messages typically appear in the console tab of the Developer tools, so we should keep this open at all times when developing.



```
import { Component } from '@angular/core';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'stock-market';
}
```

Step 5: Creating properties and initializing in ngOnInit

`stock-item.component.v1.ts`

Step 6: Creating customized CSS for the component

`stock-item.component.v1.html`
`stock-item.component.v1.css`

Step 7: Template expressions

`stock-item.component.v2.html`

Step 8: Adding additional Boolean property

`stock-item.component.v2.ts`

Step 9: Adding additional classes to the CSS

`stock-item.component.v2.css`

Step 10: Class binding

`stock-item.component.v3.html`

Experiment with changing the values of `price` and `previousPrice` in `stock-item.component.ts` to change `positiveChange` and observe the effect in `stock-item.component.html`

Step 11: Adding a new method

`stock-item.component.v3.ts`

Step 12: Event binding and property binding

`stock-item.component.v4.html`

Click to see button become disabled
https://www.w3schools.com/jsref/prop_pushbutton_disabled.asp

Step 13: Using $event object in the template

`stock-item.component.v5.html`

Step 14: Accessing properties of the $event object in the component

`stock-item.component.v4.ts`
https://www.w3schools.com/jsref/obj_mouseevent.asp

Step 15: Creating a data model

In `src/app`

```
ng generate class model/stock
```

`stock.v1.ts`

Step 16: Refactoring the component to use the data model

`stock-item.component.v5.ts`

Step 17: Modifying template to reflect this use

`stock-item.component.v6.html`

Step 18: Using Augury

If you have installed the Augury Chrome extension, you should be able to view it from the main menu in the Developer tools window. This allows you to view the nesting of the components in the component tree as well, the state of the component (i.e. contents of its properties) and also access the source code of the individual components in `webpack://` in the Sources tab of the Developer tools.

1.5.2 exercise-databinding

Run `npm install` in `exercise-databinding-start`.

Implement the following functionality:

a) Clicking on the button changes the button message and the displayed image. There are a total of 4 images (red, green, blue, yellow) and the button message indicates the next image to be displayed. The view cycles between these 4 images

b) Typing characters into the text box changes the message displayed below:
   - If no characters are entered, the message is : Nothing entered yet
   - Less than 6 characters: Entering a string
   - Less than 11 characters: Medium string
   - 11 characters or longer: Long string

Hints:
- The 4 image files are located in `/src/assets`
- The event corresponding to an entry in the input text box is `(input)`. To obtain the value of an element corresponding to this event in the template use `event.target.value`

The answers are in `exercise-databinding-final`. Run `npm install` in here before running the app.

You can run both versions at the same time by running them on different ports with `ng serve` (which currently runs on the default port 4200). For e.g. you could run the version in `exercise-databinding-final` with:

```
ng serve --port=xxxx
```

where `xxxx` can be any free port on your machine, e.g. 9999

1.6 Directives

1.6.1 stock-market-directives

Start from the code base in `stock-market-directives/stock-market-directives-start`. This is the final code base from the `stock-market` lab and can also be found in `stock-market/stock-market-final`

Step 1: Adding new classes to the CSS

`stock-item.component.v1.css`

Step 2: Create object for use with NgClass in component

`stock-item.component.v1.ts`

Step 3: Using NgClass directive

`stock-  item.component.v1.html`

Experiment with instantiating Stock object instance with different values for `previousPrice` and `price` to view the effect on the template.

Step 4: Create object for use with NgStyle in component

`stock-item.component.v2.ts`

Step 5: Using NgStyle directive

`stock-item.component.v2.html`

Experiment with instantiating Stock object instance with different values for `previousPrice` and `price` to view the effect on the template.

Step 6: Adding / removing one particular class with class binding

```
stock-item.component.v3.html
```

Step 7: Using NgIf Directive

```
stock-item.component.v4.html
```

Clicking on the Add to Favourite button should remove it

Step 8: Modify component to use an array of Stocks

```
stock-item.component.v3.ts
```

Step 9: Using the NgFor directive

```
stock-item.component.v5.html
```

Step 10: Add categorization method to data model

```
stock.v1.ts
```

Step 11: Using the NgSwitch directive

```
stock-item.component.v6.html
```

Step 12: Combining NgIf and NgFor

```
stock-item.component.v7.html
```

Step 13: Add function to track individual items `stock-item.component.v4.ts`

Step 14: Using modified NgFor directive

`stock-item.component.v8.html`

1.6.2 exercise-debugging

Run `npm install` in `exercise-debugging-start`.

This app is supposed to add an item to a list when the Add Car button is clicked and display the list of items immediately below. Then clicking on a specific item in display should remove it from the list.

There is both a run-time error and logic error in this app (we are not able to remove the last item from the displayed list). Run-time error messages typically appear in the console tab of the Developer tools, and we should therefore keep this open at all times when developing. Logic errors can be debugged by placing appropriate breakpoints in the affected source code files (available in `Sources -> webpack://.`) and stepping through the code execution. We will examine both approaches.

1.6.3 exercise-directives

Run `npm install` in `exercise-directives-start`.

Implement the following functionality:

a) Clicking on the `save string` button adds the current entry in the input text box into an internal list in the component.

b) Clicking on the `Show list of strings entered` button toggles between showing and hiding the strings in the list

c) The list of strings are styled in the following manner:
   - Strings with a length of 5 or longer appear in red. The class for this style is already in `app.component.css`. Use `NgClass` to accomplish this.
   - From the 5th string onwards in the list, the background color is light blue. Earlier strings do not have any background color. Use `NgStyle` to accomplish this.

The answers are in `exercise-directives-final`. Run `npm install` in here before running the app.

You can run both versions at the same time by running them on different ports with `ng serve` (which currently runs on the default port 4200). For e.g. you could run the version in `exercise-directives-final` with:

```
ng serve --port=xxxx
```

where `xxxx` can be any free port on your machine, e.g. 9999

1.7 Working with components

1.7.1 component-stuff

Start from the code base in `component-stuff/component-stuff-start`. This is the final code base from the `stock-market` lab and can also be found in `stock-market/stock-market-final`

Step 1: Using an inline template

`stock-item.component.v1.ts`

Step 2: Using inline styles

`stock-item.component.v2.ts`

Step 3: Style encapsulation

`app.component.v1.css`

Step 4: Style encapsulation continued

`app.component.v1.ts`

Step 5: Input decorator in child component

`stock-item.component.v3.ts`

Step 6: Creating object in parent component

`app.component.v2.ts`

Step 7: Passing object via component property binding

`app.component.v1.html`

Step 8: Output decorator in child component

`stock-item.component.v4.ts`

Step 9: Modify template to use NgIf

`stock-item.component.v1.html`

Step 10: Add trigger method to parent component

`app.component.v3.ts`

Step 11: Passing output data through component event binding

`app.component.v2.html`

Step 12: Creating ChangeDetectionStrategy in child component

`stock-item.component.v5.ts`

Step 13: Adding a button in the child template

`stock-item.component.v2.html`

Step 14: Adding buttons and event binding in the parent template

`app.component.v3.html`

Step 15: Adding event-related functions to test in parent component

`app.component.v4.ts`

Experiment with pressing all buttons

Step 16: Component life cycle hooks on parent component

`app.component.v5.ts`

Step 17: Component life cycle hooks on child component `stock-item.component.v6.ts`

Run this a few times, clicking on the various buttons of the parent (`app.component`) and child component (`stock-item.component`) templates. Notice:

- All the `Init` methods are only called once (`OnInit`, `AfterContentInit`, `AfterViewInit`) is only called once for both components
- The `AfterView` methods of the bottom-most child in the component hierarchy is called first, followed by the second bottom-most child and so all the way up to the root component.
- Clicking on the `Add to Favourite` buttons and the `Change Price` buttons in the child component (`stock-item.component`) result in cascade of `DoCheck` and `AfterContentChecked` method calls starting from the root component all the way down to the bottom-most child in the hierarchy.
- Clicking on these 2 buttons however DOES NOT result in `OnChanges` called, because there is no change in the component property binding from parent (`app.component`) to child
- Clicking on the `Change Stock` button in the parent component results in call to `OnChanges` in the child component, as there is now a change in the component property binding from parent to child (a new Stock object is created).
- Clicking on the `Change Price` button in the parent component DOES NOT result in a call to `OnChanges` in the child component, as the change is in the property of the object (`stockObj.price`) and not the object itself (`stockObj`), and therefore there is no change to in the component property binding from parent to child

Step 18: Using ngContent for content projection

`stock-item.component.v3.html`

Step 19: Add test method in parent component

`app.component.v6.ts`

Step 20: Modify parent template for content projection

`app.component.v4.html`

1.7.2 exercise-component

Run `npm install` in `exercise-component-start`.

Implement the following functionality:

a) The app uses a data model based on the class `Employee.ts`. The `languages` and `os` properties are only meaningful in the context of the value of the `role` property. If the `role` property has the value `Admin`, then only the `os` property is used, and if the `role` property has the value `Developer`, then only the `languages` property is used.

b) There are 3 components in this app: `AppComponent`, `EmployeeDetailComponent` and `EmployeeFormComponent`.

c) `AppComponent` contains a radio button selector which allows the selection of either a Developer or Admin role. There are 2 additional buttons which can toggle the display of the employee form (handled by `EmployeeFormComponent`) and the list of employee details (handled by `EmployeeDetailComponent`)

d) Depending on the role value selected in the radio button, the employee form will show the relevant fields for that role (as explained in (a)).

e) When the `Add Record` button is clicked, the relevant info for a single employee is added to the main employee list that is maintained in `AppComponent`.

f) When the employee list is shown, only the relevant info for that particular employee is displayed in the context of that employee's role (as explained in (a)).

Hints:

- Create and maintain the main employee list (as an array of Employee objects) in `AppComponent`
- Use the `ngIf` directive in `AppComponent` along with two `boolean` properties to toggle the display of the `EmployeeFormComponent` and `EmployeeDetailComponent`.
- Pass the selected value in the radio button in `AppComponent` to `EmployeeFormComponent` via an `Input` property and component binding.
- Use this value in `EmployeeFormComponent` along with `ngIf` or `ngSwitch` directive to control the relevant input form fields to display.
- When the `Add Record` button is selected, create a new `Employee` object and pass it back to `AppComponent` via an `Output` property and component binding. Add this object to the main employee list in `AppComponent`
- Each `EmployeeDetailComponent` should ideally display details for only one employee record. Use the `ngFor` directive to iterate over the main employee list and then pass each individual `Employee` object to `EmployeeDetailComponent` via an `Input` property and component binding
- Just as in `EmployeeFormComponent`, use the `ngIf` or `ngSwitch` directive to control the relevant fields to display.

The answers are in `exercise-component-final`. Run `npm install` in here before running the app.

You can run both versions at the same time by running them on different ports with `ng serve` (which currently runs on the default port 4200). For e.g. you could run the version in `exercise-component-final` with:

```
ng serve --port=xxxx
```

where `xxxx` can be any free port on your machine, e.g. 9999

1.8 Forms

1.8.1 forms-basic

Start from the code base in `forms-basic\forms-basic-start`

Step 1: Import appropriate modules

`app.module.v1.ts`

Step 2: Create a new component

In `src/app`

```
ng generate component stock/create-stock
```

Step 3: Modify create-stock

`create-stock.component.v1.ts`

Step 4: Edit new template

`create-stock.component.v1.html`

Step 5: Modify parent template to include new template

`app.component.v1.html`

Fill in the text form and note the changes

Step 6: Using NgModel and NgModelChange directives

`createstock.component.v2.html`

Note that using these directives mean that we no longer need to know what is the specific input event and specific property to bind to for that given element.

Step 7: Using banana-in-the-box syntax

`create-stock.component.v3.html`

Step 8: Modifying data model and components

`stock.v1.ts`

`create-stock.component.v2.ts`

`app.component.v1.ts`

Step 9: Complete form using NgSubmit and ngModel

`create-stock.component.v4.html`

Note that NgModel and NgModelChange are still used separately in the event there is some additional operation to be performed besides a pure two-way binding between a component property and form control element value.

Step 10: Modification to use NgFor and NgValue for select drop-down box

`createstock.component.v5.html`

`create-stock.component.v3.ts`

Step 11: Modification of CSS for color scheme for validation

`create-stock.component.v1.css`

Inspect the various form control elements in the Elements tab of the Chrome developer tools and notice the change in the classes that are applied to them (`ng-valid`, `ng-dirty`, `ng-touched`, etc) when you interact with them (when you touch them and when you start typing in them).

Step 12: Modification of template for validation

`create-stock.component.v6.html`

For the stock price form control, you will need to click in it and then click out of it (without typing anything) for the class to change from `ng-untouched` to `ng-touched`.

Step 13: Modification of CSS for color scheme for validation

`create-stock.component.v2.css`

Step 14: Change to component to do logging

`create-stock.component.v4.ts`

Step 15: Using template reference variables and detailed validation requirements

`create-stock.component.v7.html`

Notice that attempting to submit the form that has even one invalid form control value results in an error message.

Step 16: Modifying component to work with FormGroups

`create-stock.component.v5.ts`

Step 17: Modifying template to work with FormGroups using NgModelGroup

`create-stock.component.v8.html`

Notice that with using `NgModelGroup`, there is no longer need to individually bind each form control element to a corresponding property of the `Stock` object in the component. Instead, you can access the entire contents for a stock object directly from the `stockForm` parameter.

1.8.2 exercise-forms-template

Run `npm install` in `exercise-forms-template-start`.

This is the code base for the solution of `exercise-component`. You can also find it in `exercise-component-final`

Implement the following functionality:

1. Add an additional property to `Employee`, which specifies an email (string type)
2. Revise the `App` template (`app.component.html`) and `EmployeeForm` template (`employee-form.component.html`) to use the template-driven approach to creating forms. The requirements are as follows:

- All fields are compulsory
- The employee name can only contain alphabets and spaces
- The employee email must follow a standard email format (xxx@yyy.zzz)

- If the employee is an Admin, the OS used can have a maximum length of 10 characters
- If the employee is a Developer, the languages used can have a minimum length of 4 characters
- Appropriate messages and styling are used for all fields to indicate validity as well as the type of errors (e.g. for the case of employee name, an empty field should result in a message saying the field is mandatory and an entry with non-alphabets should result in a message saying that only alphabets are accepted.
- Clicking the Add Record button with at least one invalid entry should result in an alert box indicating this situation.
- If the form is valid, then all entries are stored into the Employee object and transmitted from the `EmployeeForm` component in the usual manner.

Hints:
For the requirement of alphabets and spaces, as well as standard email format, you can use the `email` and `pattern=' ???  '` validator specified at https://angular.io/api/forms/Validators

The answers are in `exercise-forms-template-final`. Run `npm install` in here before running the app.

You can run both versions at the same time by running them on different ports with `ng serve` (which currently runs on the default port 4200). For e.g. you could run the version in `exercise-forms-template-final` with:

```
ng serve --port=xxxx
```

where xxxx can be any free port on your machine, e.g. 9999

1.8.3 forms-reactive

Start from the code base in `forms-reactive/forms-reactive-start`. This is the final code base from the `forms-template` lab and can also be found in `forms-template\forms-template-final`

Step 1: Import ReactiveFormsModule

`app.module.v1.ts`

Step 2: Using formControl binding in template

`create-stock.component.v1.html`

Step 3: Changing component to support template `create-stock.component.v1.ts`

Try entering a single value

Step 4: Modifying template to obtain all fields

`create-stock.component.v2.html`

Step 5: Modifying component to support FormGroup

`create-stock.component.v2.ts`

Try entering values for the various form controls and check that classes that are applied to them (`ng-valid`, `ng-dirty`, `ng-touched`, etc) in accordance to the specified requirements

Step 6: Modifying component to support FormBuilders

`create-stock.component.v3.ts`

Step 7: Modify template for error messages

`create-stock.component.v3.html`

Step 8: Modify template to simulate resetting and loading to a server

`create-stock.component.v4.html`

Step 9: Change classes to fit these changes

`create-stock.component.v4.ts`

`stock.v1.ts`

`app.component.v1.ts`

Experiment with the buttons in the template. Note that the `setValue` method call requires the object being passed as a parameter (`stockFormModel`) to be of the same shape as the `FormGroup` variable (in this case, `stockForm`): meaning it must have exactly a name, code and price property. The `patchValue` method call will simply try to match the properties in the object being passed as a parameter to the shape of the `FormGroup` variable, automatically dropping properties from the parameter which are not used.

Step 10: Update model to include array

`stock.v2.ts`

Step 11: Update component to use FormArray

`create-stock.component.v5.ts`

Step 12: Update CSS for separation

`create-stock.component.v1.css`

Step 13: Modify template to include array

`create-stock.component.v5.html`

1.8.4 exercise-forms-reactive

Run `npm install` in `exercise-forms-reactive-start`.

This is the code base for the solution of `exercise-forms-template`. You can also find it in `exercise-forms-template-final`

Implement the following functionality:

a) Refactor the template-driven form approach in `employee-form.component.html` to use a reactive approach instead. Leave `app.component.html` in a normal or template-driven approach.

b) Maintain the same validation requirements as previously. In addition, ensure that the employee `age` property is between 20 and 80. Add appropriate validation error messages if this requirement is violated

Hints:

- You must include BOTH `ReactiveFormsModule` and `FormsModule` in your root module (`app.module.ts`) to use both normal (`app.component.html`) and reactive forms (`employee-form.component.html`) together in a single app

- If you have more than one validator to be applied to a form control with the `FormGroup`, ensure that these validators are grouped into an array (e.g. `[Validators.required, Validators.min(20),`…etc])
- For the requirement of alphabets and spaces, as well as standard email format, you can use the `email` and `pattern='??? ' validator` specified at https://angular.io/api/forms/Validators

The answers are in `exercise-forms-reactive-final`. Run `npm install` in here before running the app.

You can run both versions at the same time by running them on different ports with `ng serve` (which currently runs on the default port 4200). For e.g. you could run the version in `exercise-forms-reactive-final` with:

```
ng serve --port=xxxx
```

where xxxx can be any free port on your machine, e.g. 9999

1.9 Services

1.9.1 services-basic

Start from the code base in `services-basic/services-basic-start`. This is the final code base from the `forms-template` lab and can also be found in `forms-template\forms-template-final`

Step 1: Add additional buttons to template

`stock-item.component.v1.html`

Step 2: Create Stock-list component

In `src/app`

```
ng generate component stock/stock-list
```

Step 3: Modify new component

`stock-list.component.v1.ts`

Step 4: Modify template of new component

`stock-list.component.v1.html`

Step 5: Modify parent component and template

`app.component.v1.ts`

`app.component.v1.html`

Step 6: Create a stock service

In `src/app`

```
ng generate service services/stock
```

Step 7: Edit stock service to return dummy data

`stock.service.v1.ts`

Step 8: Registering service as provider

`app.module.v1.ts`

Step 9: Using service in stock list component

`stock-list.component.v2.ts`

Modify the properties of the various Stock objects in `stock.service.ts` to verify that the list of stocks used by `stock-list.component` is indeed obtained from this service.

Step 10: Modify template to show service

`create-stock.component.v1.html`

Step 11: Modify component to use service

`create-stock.component.v1.ts`

Experiment with creating new stocks in the Create Stock Form part of the page. Note the appearance of the informational messages, including when you attempt to create a new stock with a code that matches an existing stock. The `CreateStock` component and the `StockList` component both access the Stock service to add new stocks and retrieve the main list of stocks.

Step 12: Creating a message service and registering it as a provider

In `src/app`

```
ng generate service services/message
```

`app.module.v2.ts`

Step 13: Update the message service

`message.service.v1.ts`

Step 14: Modify the main component and template to use it

`app.component.v2.ts`

`app.component.v2.html`

Step 15: Modify child component to use the same service

`create-stock.component.v2.ts`

Step 16: Modify template to show the service

`create-stock.component.v2.html`

Experiment with creating new stocks in the Create Stock Form part of the page. Notice that the informational messages regarding the success or otherwise of the stock creation is shown in the template of the main component (app.component.html). The `CreateStock` component and the `App` component communicate using the Message service, for which there is only one single instance

Step 17: Adding provider at a child component level

`create-stock.component.v3.ts`

With the `MessageService` registered in the `providers` array of the `Component` metadata for `CreateStock`, this means that the instance of `MessageService` created here is DIFFERENT from the instance of the `MessageService` created in `App` component. Therefore the changes made to the `MessageService` instance registered with `CreateStock` DO NOT influence the `MessageService` instance of `App` component and vice versa.

1.9.2 exercise-services

Run `npm install` in `exercise-services-start`.

This is the code base for the solution of `exercise-forms-template`. You can also find it in `exercise-forms-template/exercise-forms-template-final`

Implement the following functionality:

a) Create a new service `EmployeeService` and a new component `EmployeeDisplay`. Refactor the app code base so that this service maintains the list of employees while the `EmployeeDisplay` displays the details of the employees via the `EmployeeDetail` component. Both `EmployeeDisplay` and `EmployeeForm` component interact directly with this service (instead of the App component) to retrieve the list and also add new entries to the list.

b) Create another service `HighlightService`. In the `App` component, allow the user to specify several in-demand languages / OS and the employee displayed by `EmployeeDetail` will be highlighted in a specific manner if the employee has those specific languages / OS skills. You can use any particular styling you want for the highlight (e.g. change background color, emphasize text, increase font size, use borders for div, etc).

Hints:
You may find using the `split` and `indexOf` methods available for Javascript strings helpful in your solution:

https://www.w3schools.com/js/js_string_methods.asp
https://www.w3schools.com/jsref/jsref_split.asp

The answers are in `exercise-services-final`. Run `npm install` in here before running the app.

You can run both versions at the same time by running them on different ports with `ng serve` (which currently runs on the default port 4200). For e.g. you could run the version in `exercise-services-final` with:

```
ng serve --port=xxxx
```

where `xxxx` can be any free port on your machine, e.g. 9999

1.10 Observables

1.10.1 observables-basic

Start from the code base in `observables-basic\observables-basic-start`

Step 1: Add observables to service

`stock.service.v1.ts`

Step 2: Change components to read directly from observable

`stock-list.component.v1.ts`

`create-stock.component.v1.ts`

Step 3: Further simplification to use observable

`stock-list.component.v2.ts`

Step 4: Modification of template to fit this

`stock-list.component.v1.html`

1.11 HTTP

1.11.1 http-basic

Start from the code base in `observables-basic\observables-basic-final`

In the folder `http\basic-server`
run:

```
npm install
node index.js
```

to start the back-end server. This basic server exposes 3 APIs:

- GET on `/api/stock` to get a list of stocks
- POST on `/api/stock` with the new stock as a body to create a stock on the server
- PATCH on `/api/stock/:code` with the stock code in the URL and the new favorite status in the body of the request, to change the state of favorite for the particular stock.

Type this URL into the address bar of the browser to test out the GET API:

```
http://localhost:3000/api/stock
```

Confirm the return response of this JSON document:

[{"name":"Test Stock Company","code":"TSC","price":85,"previousPrice":80,"exchange":"NASDAQ","favorite":false}, {"name":"Second Stock Company","code":"SSC","price":10,"previousPrice":20,"exchange":"NSE","favorite":false}, {"name":"Last Stock Company","code":"LSC","price":876,"previousPrice":765,"exchange":"NYSE","favorite":false}]

Step 1: Add a dependency on HttpClientModule

`app.module.v1.ts`

Step 2: Change component to make HTTP calls

`stock.service.v1.ts`

Step 3: Change data model to interface

`stock.v1.ts`

Step 4: Modifying component and template to use new data model

`stock-list.component.v1.ts`

`stock-list.component.v1.html`

Step 5: Modifying another component and template

`stock-item.component.v1.ts`

`stock-item.component.v1.html`

Step 6: Modify creation of stock

`create-stock.component.v1.ts`

Step 7: Create proxy file

Create a file `proxy.conf.json` in the main project folder and populate it with:

```
{
  "/api": {
    "target": "http://localhost:3000",
    "secure": false
  }
}
```

Step 8: Serve the app with reference to the proxy

In the main project folder, type:

```
ng serve --proxy-config proxy.conf.json
```

1.11.2 http-advanced

Start from the code base in `http-basic\http-basic-final`

In the folder `http\basic-server`
run:

```
npm install
node index.js
```

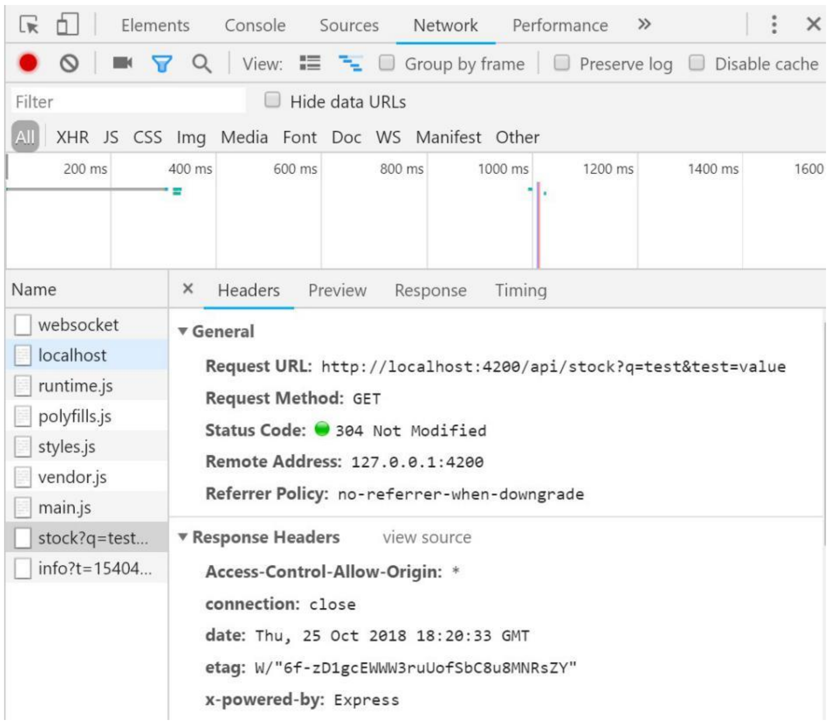
In the main project folder, type:

```
ng serve --proxy-config proxy.conf.json
```

Step 1: Adding HTTP headers

`stock.service.v1.ts`

After reloading, select Network and in the Headers tab for the stock connection, you should be able the new request URL and header. The server in this case will only provide a response of a single stock corresponding to those URL parameters (`q=test&test=value`)



```
General
  Request URL: http://localhost:4200/api/stock?q=test&test=value
  Request Method: GET
  Status Code: ● 304 Not Modified
  Remote Address: 127.0.0.1:4200
  Referrer Policy: no-referrer-when-downgrade
Response Headers
  Access-Control-Allow-Origin: *
  connection: close
  date: Thu, 25 Oct 2018 18:20:33 GMT
  etag: W/"6f-zD1gcEWWW3ruUofSbC8u8MNRsZY"
  x-powered-by: Express
```

Step 2: Modify request to add observe property

`stock.service.v2.ts`

Step 3: Modify component to make calls to the APIs

`stock-list.component.v1.ts`

1.11.3 http-interceptor

Start from the code base in `http-basic\http-basic-final`

In the folder `http\basic-server`
run:

```
npm install
node index.js
```

In the main project folder, type:

```
ng serve --proxy-config proxy.conf.json
```

Step 1: Generate a service for authorization and register it

In `src/app`

```
ng generate service services/auth
```

`app.module.v1.ts`

Step 2: Add property to Auth service

`auth.service.v1.ts`

Step 3: Add HTTP API call

`stock.service.v1.ts`

Step 4: Change component to add extra buttons

`stock-list.component.v1.ts`

Step 5: Change template to accommodate new buttons

`stocklist.component.v1.html`

Step 6: Create the interceptor

Create:
`src/app/services/stock-app.interceptor.ts`

Step 7: Add interceptor to main module

`app.module.v1.ts`

Step 8: Adding functionality to the interceptor

`stock-app.interceptor.v2.ts`

1.11.4 observables-extra

Start from the code base in `http-basic\http-basic-final`

In the folder `http\basic-server`
run:

```
npm install
node index.js
```

In the main project folder, type:

```
ng serve --proxy-config proxy.conf.json
```

Step 1: Edit template, show number of stocks

`stocklist.component.v1.html`

Check that 2 calls are made to the server

Step 2: Modify template to make calls

`stock-list.component.v1.ts`

Check that 1 call is made to the server

Step 3: Add code to search for stocks based on query string

`stock.service.v1.ts`

Step 4: Change template to make updated call

`stock-list.component.v2.html`

Step 5: Modify component to reflect change

`stock-list.component.v2.ts`

Reload and note in the network component that a HTTP call is made for every key stroke

Step 6: Augment component with observable operators `stock-list.component.v3.ts`

1.12 Routing

1.12.1 Routing

Start from the code base in `routing\routing-start`

In the folder `routing\second-server`
run:

```
npm install
node index.js
```

In the main project folder, type:

```
ng serve --proxy-config proxy.conf.json
```

Step 1: Setting up index.html

`index.v1.html`

Step 2: Generate routing module

In `src/app`
```
ng generate module app-routes --flat --routing
```

Step 3: Update routing module

`app-routes-routing.module.v1.ts`

Step 4: Linking route to main module

`app.module.v1.ts`

Step 5: Modify main template to load components for routes

`app.component.v1.html`

Step 6: Modify main template to allow navigation

`app.component.v2.html`

When loading the app, notice that you are not able to get the list of stocks from the stock list section yet as this is a protected section, only accessible after a valid login

Step 7: Modify CSS for main template

```
app.component.v1.css
```

Step 8: Provide default route for initial load

```
app-routes-routing.module.v2.ts
```

Step 9: Provide default route for incorrect URL

```
app-routes-routing.module.v3.ts
```

Step 10: Add new route

```
app-routes-routing.module.v4.ts
```

Step 11: Update new component

```
stock-details.component.v1.ts
```

Step 12: Modify registration component

```
register.component.v1.ts
```

Step 13: Modify login component

```
login.component.v1.ts
```

Verify that you can register and login using an appropriate user name / password combination. Also incorrect user name / password combinations for login or existing user names for registration are flagged according. After successful login, you are redirected to the stock list route, where the stocks are finally retrieved.

Step 14: Modify template to navigate to specific stock

```
stock-item.component.v1.html
```

Now you should be able to display a specific stock by clicking on one of the stocks in the list

Step 15: Modify component to pass query params

`login.component.v2.ts`

Step 16: Modify component to read query params

`stock-list.component.v1.ts`

Step 17: Modify template to include button

`stock-list.component.v1.html`

Step 18: Modify component to use observable

`stock-list.component.v2.ts`

Step 19: Generate and update authentication guard

In `src/app`

```
ng generate guard guards/auth
```

`auth.guard.v1.ts`

Step 20: Update main module with the guard

`app.module.v2.ts`

Step 21: Add auth guard to the routing module

`app-routes-routing.module.v5.ts`

Check that trying to navigate directly to the stock list or create stock page will end up redirecting you to the login page.

Step 22: Generate and update deactivation guard

In `src/app`

```
ng generate guard guards/CreateStockDeactivate
```

`create-stock-deactivate.guard.v1.ts`

Step 23: Update main module with the guard

`app.module.v3.ts`

Step 24: Add deactivation guard to the routing module

`app-routes-routing.module.v6.ts`

Reload the app, log in and navigate to the create stock page, and then try clicking any of the links at the top. At that point, you should see the confirmation asking whether you really want to navigate away.

Step 25: Generate a Resolver service

In `src/app`

```
ng generate service resolver/stock-load-resolver
```

Step 26: Update Resolver service

`stock-load-resolver.service.v1.ts`

Step 27: Update main module with the guard

`app.module.v4.ts`

Step 28: Add deactivation guard to the routing module

`app-routes-routing.module.v7.ts`

Step 29: Modify component to prefetch information

`stock-details.component.v2.ts`

1.13 Additional apps for demo

1.13.1 angular-reddit

Final code base in `angular-reddit`

31

1.13.2 portfolio

Start from the code base in `portfolio\portfolio-start`

Do `npm install` first. You will get a bunch of warning messages as this uses a few deprecated modules and an older version of Angular is being used.

Then do `ng serve`.

Step 1: Modify the existing account service

133

Listing 6.1
`account.service.v1.ts`

Step 2: Modify app component to use service

134

Listing 6.2
`app.component.v1.ts`

Step 3: Register service in the module

135

`app.module.v1.ts`

Step 4: Modify component to use service

136

`stocks.component.v1.ts`

Step 5:

136

Listing 6.3
`investments.component.v1.ts`

Step 6: Include top bar in template

137

`app.component.v1.html`

Toolbar should now be populated with values from Account Service

Step 7: Create a class with static values for configuration

141

Listing 6.4
`config.service.v1.ts`

Step 8: Use configuration class to set values in main

142

Listing 6.5
`main.v1.ts`

Step 9: Use HTTP client in stock service

143

Listing 6.6
`stocks.service.v1.ts`

Step 10: Register service in the module

135

`app.module.v2.ts`

Step 11: Use stock service in app component

144

Listing 6.7
`app.component.v2.ts`

Step 12: Include remaining components in template

145

`app.component.v2.html`

You should now be able to see the full stock display panel

Step 13: Use the HTTP interceptor

147

Listing 6.8
`interceptor.service.v1.ts`

Step 14: Import token in app module

148

`app.module.v3.ts`

Step 15: Implement local storage

150

Listing 6.9
`local-storage.service.v1.ts`

Step 16: Use local storage in account service

150

Listing 6.10
`account.service.v2.ts`

Step 17: Do initialization from app component

152

`app.component.v3.ts`

You should be able to see the full performance of the main dashboard view by now

Step 18: Add service for alerts

154

Listing 6.11
`alert.service.v1.ts`

Step 19: Use service in component

154

`alert.component.v1.ts`

Step 19: Register provider with module

155

`app.module.v4.ts`

Step 20: Include alert in main template

155

`app.component.v3.html`

Step 21: Modify app component to use the alert service

155

`app.component.v4.ts`

Step 22: Modify account service to use the alert service

156

Listing 6.12
`account.service.v3.ts`

You should now be able to see the alert corresponding to buying and selling actions

1.13.3 Music

Final code base in `routing\music`

You will need to generate a new API key before running this app.
Go to `src/environments`, rename the existing Typescript file containing the hardcoded API key to `old-spotifyApiKey.ts`

Go to scripts, and execute the Javascript file there to generate a new API key in `src/environments`

```
node spotifyKey.js
```

Return back to the root project folder and start the app in the usual way:
```
ng serve
```

1.14 Testing

1.14.1 unit-test-basic

Start from the code base in `unit-test-basic\unit-test-basic-start`

Step 1: Initial app unit test

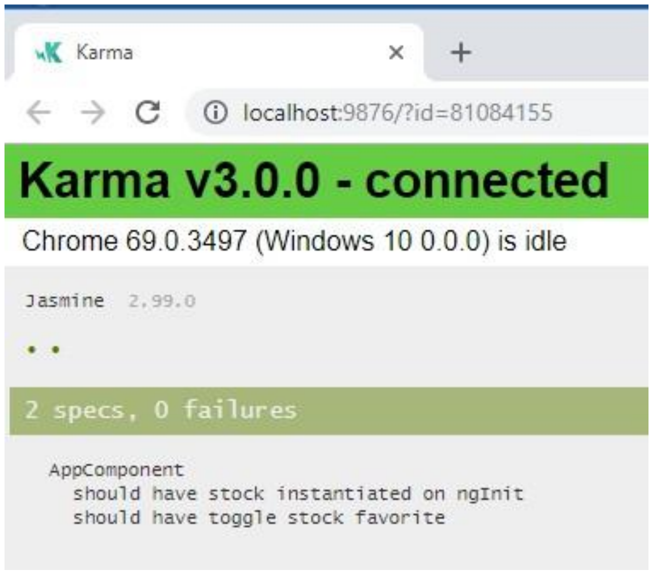
In `src\app`, create

`app.component.spec.ts`

In the main project folder, run test with:

```
ng test
```

A successful test should give a result similar to below:



```
Karma v3.0.0 - connected
Chrome 69.0.3497 (Windows 10 0.0.0) is idle
Jasmine 2.99.0
• •
2 specs, 0 failures
  AppComponent
    should have stock instantiated on ngInit
    should have toggle stock favorite
```

Step 2: Creating an Angular aware test

Create
`stock-item.component.spec.ts`

Step 3: Testing component interaction

`app.component.spec.v2.ts`

1.14.2 unit-test-service

Start from the code base in `observables-basic\observables-basic-start`

Step 1: Original test: (no need for modification)

`stock.service.spec.ts`

Step 2: Add test for adding and fetching a list of stocks

`stock.service.spec.v2.ts`

Step 3: Modify component to test with real service

`stock-list.component.spec.ts`

Step 4: Modify component to test with mock calls

`stock-list.component.spec.v2.ts`

Step 5: Modify component to test with fake service

`stock-list.component.spec.v3.ts`

1.14.3 unit-test-async

Start from the code base in `observables-basic\observables-basic-final`

Step 1: Testing async

`create-stock.component.spec.ts`

Step 2: Testing using fake async

`create-stock.component.spec.v2.ts`

1.14.4 unit-test-http

Start from the code base in `http-basic\http-basic-final`

Step 1: Test initialization logic of fetching using HTTP GET

`stock-list.component.spec.ts`

1.14.5 Music

Final code base in `routing\music`

We integrate PhantomJS here with Karma to perform headless testing of web applications, suitable for as part of a continuous integration system.

To run unit tests

```
ng test
```

To run end to end test

```
ng e2e
```

1.14.6 unit-test-forms

Final code base in `unit-test-forms\forms`

To run unit tests

```
ng test
```

To run end to end test

```
ng e2e
```

1.15 Deploying

Once we have created a production build, you can upload the bundled assets and scripts generated from the build to a public server. Typically, this will be server within your company infra or an external cloud server instance. Many platforms now offer serverless deployment which greatly simplifies the process of deploying onto a public cloud by abstracting away the lower-level details of server management. An example is: https://zeit.co/ which we will use:

Install Zeit Now for deployment with:

```
npm install -g now
```

We will use the `angular-reddit` app to demonstrate this process. In the main project folder, type

```
ng build --prod
```

Then switch to dist folder in the command prompt and deploy with now:

```
now
```

Enter a valid email address to complete the initial registration. Check the inbox for an email and click on the link to complete verification. A confirmation message is displayed at the prompt. Type

```
now
```

again to deploy the app and obtain the public URL to access it (this will be in the form https://xxxxx.now.sh)