|  |  |
| --- | --- |
| **Exercise** | Angular.1 |
| **App** | (None – create a new **Angular** app from scratch) |
| **Purpose** | Get up-and-running with **Angular** |
| **Description** | We first install the **Angular CLI** (Command-Line Interface) tool, and then use this tool to create an actual Angular app. |
| **Steps** | 1. Start up Visual Studio Code. 2. Open a terminal window, and run: **> npm install –g @angular/cli** (**NB**: be sure to remember to include the **@**-symbol). This will install the **Angular CLI** tool, and make it globally available. **NB**: it may take several minutes before the installation is complete. 3. Now open a terminal window in the root of the folder where you wish to create **Angular** apps, and run **> ng new helloworld**. This will create a new **Angular** app for you, in the folder **helloworld**. During the creation process, you will be prompted for making a choice two times. In both cases, just choose the default option by typing **return**.  **NB**: it may (again) take a few minu­tes before the creation process is complete. 4. Once the creation is done, navigate into the newly created folder **hello­world** (in the terminal window, do **> cd helloworld**), and run the command **> ng serve --open**. This will start up the **Angular** development server (you can shut down the server using **Ctrl+C** later on), and your app will show up in your browser at **localhost:4200**. This may also take a few seconds… 5. Once the app launches, you should see something like this:      1. Now open the file **app.component.ts**, found in the folder **/src/app**. Change the value of the property **title** to something else (maybe your name), save the file, and see if the change shows up in the browser. 2. If you have time left, see if you can also change the **Angular** logo image to something else (Hint: put an image file into the **/src/assets** folder, and use it in the **src** property in the **<img>** tag in **/src/app/app.component.html**) |

|  |  |
| --- | --- |
| **Exercise** | Angular.2 |
| **App** | **Helloworld** app from **Angular.1** exercise |
| **Purpose** | Implement and use a new component in an **Angular** app. Use **Boot­strap** for styling. |
| **Description** | The **helloworld** app initially only contains the top-level **AppComponent** compo­nent, found in the **/src/app** folder. We will now create a new component, and use it as well. Also, we will install and use **Bootstrap** for styling. |
| **Steps** | 1. In a terminal window in Visual Studio Code, navigate to the folder which contains your **helloworld** App. 2. In that folder, run the command **> ng g c MyImage** (g and c are short for “generate” and “component”). Once the command completes, you should see a new folder named **my-image** under **/src/app**. Take a look inside the files created in this folder. Also try to open the **app.modules.ts** file, and see the references to your new component. 3. Go back to the terminal window, and run **> npm install bootstrap jquery popper**. This installs **Bootstrap** plus its main dependencies for the app (you may see some warnings during the install). 4. Once the install is finished, open the file **angular.json** in the root folder, and scroll down to the **styles** array under **architect/build/options** (it is around line 25). Now add the entry ***"node\_modules/bootstrap/dist/css/ bootstrap.min.css"*** to the styles array, exactly as written here. Save the file, and reboot the app with **> ng serve --open**. You should see that the style has now changed to **Bootstrap** default style. 5. If you did not do so during **Angular.1**, find an image file and place it in the **/src/assets** folder. Then open the **my-image.component.html** file in the folder for your newly created component, and replace the auto-generated code with an **<img>** tag with **src** referring to your image file (you can wrap the **<img>** tag itself in a **<p>** tag for better formatting). 6. Open the file **app.component.html** in the **/src/app** folder. Delete all the content below the **<div>** tag (i.e. the **<h2>** and **<ul>** tags). Also delete the **<img>** tag in the **<div>**. 7. In the **<div>** tag, add your own component just under the **<h1>** tag, simply by writing **<app-my-image></app-my-image>** (**NB**: you must explicitly write both the open and close tag!). Save, and (hopefully) see your image being shown in the browser ☺ |

|  |  |
| --- | --- |
| **Exercise** | Angular.3 |
| **App** | (None – create a new **Angular** app from scratch) |
| **Purpose** | Implement an **Angular** app which uses various forms of **Data Binding**. |
| **Description** | In this exercise, you will build an **Angular** app which has some simple calculator functionality. The primary focus is on using Data Binding for handling data input and user events. |
| **Steps** | 1. Create a new **Angular** app named **SimpleCalc**, by going through the steps described in **Angular.2** (steps 1-4). 2. Open **app.component.html** and replace the default code with the below code. Save the file, and check that the app is now up-and-running:   <div class="jumbotron">  <h1>  Angular Calculator v1  </h1>  </div>   1. Now create a new component named **Calculator**, by running the command **> ng g c Calculator**. This should create a new folder under **/src/app**. 2. Implement the **CalculatorComponent** class (in **calculator.component.ts**). Do this by adding the below code to the class:   x: number;  y: number;  res: number;  constructor() {  this.x = 0;  this.y = 0;  this.res = 0;  }  Add(){ this.res = this.x + this.y; }  Sub(){ this.res = this.x - this.y; }  Mul(){ this.res = this.x \* this.y; }  Div(){ this.res = this.x / this.y; }   1. In order to use input controls, we also need to import the **FormsModule** component. Open the **app.module.ts** file, and add the statement **import { FormsModule } from ‘@angular/forms’** at the top of the file, and also add **FormsModule** to the **imports** array. 2. Implement the template for the new component. Do this by deleting all existing code from **calculator.component.html** , and replace it with the below code (feel free to change the specific styling to suit your taste…):   <div class="row" style="margin-top: 100px">  <div class="col-6">  <h2>  X : <span><input type="number" [(ngModel)]="x"></span>  </h2>  <h2>  Y : <span><input type="number" [(ngModel)]="y"></span>  </h2>  <h2>= {{res}}</h2>  <button class="btn btn-primary" (click)="Add()">ADD</button>  <button class="btn btn-success" (click)="Sub()">SUB</button>  <button class="btn btn-info" (click)="Mul()">MUL</button>  <button class="btn btn-warning" (click)="Div()">DIV</button>  </div>  <div class="col-6"></div>  </div>     1. Before proceeding, take a bit of time to make sure you understand the various types of **binding** we see in this code. Can you find examples of:    1. One-way data binding    2. Two-way data binding    3. Event binding    4. Interpolation 2. In order to use the new component, now insert the tag **<app-calculator> Loading ... </app-calculator>** in the **app.component.html** file, e.g. just under the **<div>** tag. 3. Save, reboot the App (if needed) and see what the browser now shows. Hopefully, you now see a func­tional calculator looking something like this: |

|  |  |
| --- | --- |
| **Exercise** | Angular.4 |
| **App** | **SimpleCalcSolved** (or continue working on your App from **Angular.3**) |
| **Purpose** | Implement an **Angular** app which uses a **Service**. |
| **Description** | The App developed in **Angular.3** implements a simple calculator. In this exercise, you must “refactor” the App to use a Service. More specifically, the calculation functionality is to be extracted into a new Service, which the App will then use. |
| **Steps** | 1. Open the App in Visual Studio Code, and open a terminal window in the root of the App. Run the command **> ng g s calculation**; this will generate a new file **calculation.service.ts** in the **/src/app** folder. 2. Now implement the four calculation operations in this class. Each should be implemented as a function taking two **number** arguments and return­ing a **number** value. 3. Next, register the **CalculationService** in **app.module.ts**. Do this by first adding an **im­port** statement in the top of the file, and then adding the service to the **providers** array in **@NgModule**. 4. The existing **CalculatorComponent** should now be updated to use the new **CalculationService** service. This requires a couple of steps:    1. In **calculator.component.ts**, add an **import** statement at the top of the file to import the service, similar to how it was done in step 3.    2. Inject the **CalculationService** into the **CalculatorComponent**, by adding the service as a parameter to the constructor.    3. Rewrite the code for the four calculation functions to use the ser­vice, i.e. call the corresponding function on the service. 5. Save, reboot if needed, and check that the App still behaves as expected. 6. Now create a new component (not a new service) named **ResultCompo­nent**, by running the command **> ng g c Result**. 7. In **result.component.ts**, first import **Input** from **@angular/core**, and also import **CalculatorComponent**. 8. Add a property named**result**to the class **ResultComponent**. The property must be annotated with **@Input()**. Note that the name **result** is chosen to distinguish it from the **res** property in the existing component. 9. Move the view of the result from **CalculatorComponent** to **ResultCompo­nent** (Hint: move the code presenting the result from **calculator.compo­nent.html** to **result.component.html**). Remember to change **res** to **result**, since the template will now refer to the property in the new component. 10. Finally, use the new component from the existing component. That is, you should insert this code: **<app-result [result]='res'></app-result>** into the **CalculatorComponent**, at theplace where the result was originally shown. 11. Save, reboot, and check that the App still works as before. |

|  |  |
| --- | --- |
| **Exercise** | Angular.5 |
| **App** | **ClubRouting** |
| **Purpose** | Work with Angular routing and directives. |
| **Description** | The **ClubRouting** app can – when completed – display a bit of information about a football club. More specifically, the app already contains four components **home**, **player**, **roster** and **schedule**, each of which displays a portion of this information. The components **home** and **schedule** display static data, while the components **roster** and **player** are intended to be more dynamic. |
| **Steps** | 1. Start out by taking a look at the **Player** interface (in the **Shared** folder), and the **PlayerService** service in ***player.service.ts***. The latter provides access to a static set of **Player** objects, through the methods **getPlayers** and **getPlayer**. 2. Now open the ***app-routing.module.ts*** file. The **Routes** array is empty; fill in some reasonable route definitions, to enable navigation to the four com­ponents mentioned above. Note that the URL for navigating to the **Player** component should include a parameter, like e.g. **/player:no**. Also remem­ber to handle the root URL. 3. Open ***app.component.html***, and insert some **<a>** elements to enable navi­gation to the three top-level components (**home**, **roster** and **schedule**). Remember to use **routerLink** for navigation, i.e. not **href**! 4. Open ***roster.component.html***. This component is supposed to be able to display a list of all players. Implement this by defining a suitable tag for presenting a single player, and then use the **\*ngFor** directive to generate a tag for each player (Hint: the **RosterComponent** class contains a proper­ty **players**, which contains all players). 5. It should be possible to navigate to the **player** component by clicking on a player in the player list. Implement this by adding a handler for the ***click*** event to the tag you defined in step 5), and implement the handler itself in the **RosterComponent** class. The handler should perform the actual navi­ga­tion (Hint: use the **router** service which is injected in the constructor). 6. In ***player.component.html***, insert a button labeled “Back”, which when clicked should navigate back to the **roster** component. The handler itself should be implemented in ***player.component.ts.*** 7. Once everything works as intended, feel free to extend the app with fur­ther components, and/or improve the styling of the App. |

|  |  |
| --- | --- |
| **Exercise** | Angular.6 |
| **App** | **FinanceWebAPI** |
| **Purpose** | Work with Angular **HHTPClient** and RxJS **Observable**. |
| **Description** | The **FinanceWebAPI** app can – when completed – display a list of real financial news, retrievd from a Web service. The news is retrieved from the URL:  <https://api.iextrading.com/1.0/stock/market/news/last/5> |
| **Steps** | 1. Try calling the API directly, e.g. by pasting the URL into a browser address field. The data is fairly simply structured, since it is just an array of indivi­dual objects, each representing a “news object”. 2. In the given App, take a look at the interface **News** in ***News.ts***. This repre­sents the interface to a “news object”. 3. Implement a new service named **news** (e.g. by running the command **>ng g s news**). The service should invoke the API, in the same manner as in the **F1WebAPI** service in the **F1WebAPIExample** app. The implementation will probably be somewhat simpler, since the data has a simpler structure. 4. Implement a new component named **newslist** (e.g. by running the com­mand **>ng g c newslist**). The class **NewsListComponent** should contain a property name **newsList**, of type **News[]**. Also, you should inject the **News­Service** service into the component, in the component constructor. 5. In **ngOnInit** in **NewsListComponent**, set up a subscription to the **get…** method you defined in **News­Service**, i.e. the method with return type **Observable<News[]>**. The argument to **subscribe** should be a function which sets the value of **news­List** to the data received, i.e. like **data => this.newsList = data**. 6. In the template for the new component, define some HTML code which can display the list of news objects (Hint: define a suitable tag for presen­ting a single news object, and then use the **\*ngFor** directive to generate a tag for each news object). 7. If you now see five news objects in your browser, you are done ☺. |