EXERCISES ANGULAR 10

01 - EASY TO START

a) The general Github address with example code is

https://github.com/petereijgermans11/workshop-angular10

b) Create / start your own Hello world app. This can be done in various ways:

o Use AAA helloworld as base.

o OR: go to cli.angular.io yourself, install this tool and create a new project and start it. You can also find it in the Angular QuickStart, at https://angular.io/guide/quickstart.

o Code example: AAA helloworld.

c) Try if you can create a new component and bootstrap it. Follow these steps:

o Manually? - create a new file, for example \ app \ new.component.ts.

Import the correct dependencies.

Write a selector and an HTML template.

o Using the Angular CLI? - use the following command in the root of the project:

ng generate component <component name>

o Customize the bootstrapper (= app.module.ts) so that the new part is imported and started.

o Adjust the index.html to use the correct selector.

o Run npm install and npm start to test your new component.

02 - DATA BINDING

a) Extend your app from exercise 1 (Hello world) with a field / property. Binds this property in the template used.

o Do this explicitly with separate declaration and initialization in the ngOnInit ()

life cycle hook

o Read the operation and order of angular life cycle hooks at https://angular.io/guide/lifecycle-hooks.

o Code example (solution): BB1 data binding.

b) Create a series of objects. Bind them in the template with the \* ngFor directive.

o Specify explicitly via TypeScript that the property must consist of an array of strings or an array of objects (make your own choice).

o Code example: BB1 data binding.

c) Make explicit a model for the content of your array. This can be an object with one or more properties. Make sure that the contents of the array are objects of type <Model>.

o Code example: BB1 data-binding cities according to model

d) Come up with an application for the \* ngIf directive and use it in conjunction with your class and properties.

e) Transfer the template to an external .html file. Then use the property in @Component

templateUrl to refer to here. Test if the data binding continues to work.

03 - EVENT BINDING

a) Add an event-bound element to your application. For example, catch a (click) on a button. Call an event handler in the component when the event occurs. For example, have them show an alert () or a console.log () notification.

o Code example: BB2 event binding.

b) Test multiple types of DOM events, for example blur, focus, keypress, mousemove, etc.

See possibly Mozilla.org (https://developer.mozilla.org/en-US/docs/Web/Events) for an overview.

c) Create a form field (for example, a text field) with a local template variable. The format for this is #myFieldName. Pass the variable to the event handler and process it in an alert (), console.log () message, to update a value in the DOM, and so on.

d) Create a simple client-sided CRUD application: users can add elements to an array

(names, products, and so on) and remove from the array.

o For deletion, use the JavaScript function ArrayName.splice (…) or ArrayName

- ArrayName.filter (item =>…).

Code Listing: BB2 event binding \ ... \

app-02-complete.component.ts.

04 - ATTRIBUTE BINDING AND TWO ‐ WAY BINDING

a) Create a component with a text box. If the user enters an (English) color in the text box and clicks on a button, a corresponding <div> will get this background color.

o Investigate how this works if the colors can be selected from a series of radio buttons, or from a selection list / dropdown.

o General code example attribute binding: BB3 attribute binding and

BB5 selection list-change

b) Optional: research for yourself how the Angular concepts of class binding and style binding are used.

Directions for this can be found at https://angular.io/guide/template‐syntax#ngclass.

c) Create a text field in your component with two-way binding. The syntax for this is [(ngModel)].

Display the value of the entered text in the page.

o Consider adding FormsModule to app.module.ts.

o Create a copy function: Create two text boxes on the page. Text entered in one text box also appears in the other text box.

o Code example: BB4 two-way binding.

05‐ SERVICES

a) You should have some components with various data. Move this data from the component to a service.

o Inject the service into the constructor of the component in which you want to use the data.

o Consider the property [providers] in the @ ngModule annotation.

o Code example: C00-services-static.

b) Working with async services: create your own .json file with data and load it into your application. Think of:

o injecting HttpClientModule into the module;

o adjusting the component in which the service is consumed:

inject private http: HttpClient into the constructor

and use the observable notation with: .subscribe ().

• Code examples: C01-services-http.

c) More about RxJS: also use other operators within .pipe (). For example, test:

o retry (), delay (), possibly filter () and others in the Component.

• Do that in code example: C02-services-rxjs.

d) Read the README.md from C05-services-http-CRUD to start the application. A json server is also used. With a json server you can change a json file into a RESTFull database.

Assignment: "getCity", "deleteCity" and "updateCity" have not been implemented in the cityService. Make this work.

e) Working with Live APIs: Use an API of your choice to retrieve live data. Create a new application and use this data.

o APIs are available, for example, at openweathermap.org/API or

www.omdbapi.com.

o See also the site https://github.com/toddmotto/public‐apis and the file JavaScript

APIs.txt for more API endpoints.

o Code example: C07-services-live.

06 - APPLICATIONS WITH MULTIPLE COMPONENTS

a) Create a new application, or start from your application from the previous exercises.

b) Add a detail component. The details are displayed after a mouse click on the main component.

o Follow the step-by-step plan (4 steps) from the presentation, in which the component is created, embedded in the main component and properly invoked in HTML.

o OR: use the CLI to generate new component. Use the selector to insert it into the HTML of AppComponent (for example, <my-new-component> </my-new-component>).

o Code example: D00 components.

c) Use the @Input () annotation to pass data from the main component to the detail component.

o Make sure that an object is passed to the detail component. To do this, use the format [objName] = "objData" in the template of the main component.

o Show the contents of the object in the detail component.

• Code example: D01 component inputs.

• Also create a CityListComponent. Pass the data (list) via @Input to this component.

d) Use the @Output () annotation to pass events from the detail component to the main component. Consider, for example, passing on a rating (as in the example), or something else.

o Code example: D02 components outputs.

e) Advanced: Complete application: create a simple eCommerce application with the following components:

o A Store. Write a service for the store that provides 4 or 5 products (this may be static data).

o A Detail component. Clicking on a product shows details about the product.

o A Shopping Cart. The user can place the product in a shopping cart. The contents of the shopping cart are always visible in the main component.

o An "Order" button. When clicked, the contents of the shopping cart are displayed and the total amount is calculated.

o Code example: D03 pub sub-order component.

07 - ROUTING

a) Make your application from the previous exercises suitable for routing, or start a new application. The requirements are the following:

o Make sure that the application has a rootComponent, showing the "main menu" with options.

o Make sure that different components are loaded when a choice is made in the main menu.

o Use Code example: E00 router.

b) Optional: create a completely new application via the CLI and immediately set this application to use routing. The syntax for this is ng new <application>

o See how the CLI generated an app-routing.module.ts and its syntax. Also see how it is used in the main module app.module.ts.

c) Make your application suitable for working with route parameters. Consider, for example, creating a detailed route for a specific product in your application. Think of:

o extending app.routes.ts with parameters;

• embedding ActivatedRoute to be able to pick up the parameters in the detail component;

o Add detail component to app.module.ts !!!

o Adjusting the HTML to be able to pass parameters ([routerLink] =…)

o Request data in the detail component based on the parameter provided.

o Code example: E01 router parameter.

08 - PIPES

a) Test some standard pipes. To do this, create a new component with some statistics: strings, dates, some numbers and an array of data. Format the data in the view including: DatePipe, UpperCasePipe, LowercasePipe and CurrencyPipe. Search the Angular docs for more examples.

b) Create your own custom pipe that sorts the list of cities by alphabet.

Apply the pipe to the \* ngFor. Use the Javascript sort () function for this!

Use code example: G01-pipes

09 ‐ **ANGULAR FORMS**

**First steps:**

|  |
| --- |
| **Use ‘npm install’, to install the dependencies for ‘examples/F00-forms-template-driven’**  **Use ‘ng serve’, to start this application at** [**http://localhost:4200**](http://localhost:4200) |

**1) Make Component0 in ‘examples/F00-forms-template-driven’ active as a ‘*Route*’:**

* **Add this Component0 to the router (app.routes.ts)**
* **Import and define the Component0 in the app Module (app.module.ts).**
* **And add a routerLink in the MainComponent for this Component0.**

|  |
| --- |
| **Use for all *template driven* exercises: forms-template-driven, Component0** |

**2) Goal: build a template driven form**

o Create a simple HTML5 form in Component0 in **‘examples/F00-forms-template-driven’**.

* Import FormsModule into your app.module.ts

o Add the local template variable #myForm="ngForm" to the <form> tag.  
o Add the directives ngModel to the separate form fields. You don't need two‐way databinding with [()].  
o Write for example myForm.value to the user interface or show the contents of the form in an alert (or in the console).  
o Demo code available at **examples/F00-forms-template-driven**, Component 1.

**3) Goal: address individual controls inside the form and add HTML5 validators**

o Assign a **local template variable** to the form fields.  
o Bind ngModel to the **local template variable**. The code can look like: #email="ngModel"  
o Retrieve the values from the **local template variable** and show them in the user interface, for example its value and its validity.  
o Add the HTML5 attribute required to the form fields and see how this affects the state of the form field. Write its validity to the user interface.  
o Demo code available at **/ examples/F00-forms-template-driven**, Component 2.

**4) Goal: combining individual form fields to an ngModelGroup**

o Add some field to the form (for example some extra text fields, selectbox or checkboxes ).  
o Group them inside a <div>, assign the <div> the directive ngModelGroup. The code can look like:

<div ngModelGroup="customer" #customer="ngModelGroup">  
o Run the code and identify the model group in the returned form value object.

<pre>value {{ customer.value | json }} <br /> valid: {{ customer.valid}}</pre>

o Optional: set the value of a form field from inside your class, by using the local template variable and bind to [ngModel].  
o Demo code available at **examples/F00-forms-template-driven**, Component 3.

**5) Goal: submitting template driven forms**

o Add a submit button to the form.  
o Make sure the submit button is only active when the form as a whole is valid. Your code can look like:

<button type="submit"

(click)="onSubmit(myForm)"

[disabled]="!myForm.valid">

...

</button>

o Demo code available at **examples/F00-forms-template-driven** Component 4.

**6) Goal: working with model driven forms**

o Start with a simple form, for example build a form with the following elements:

* Username
* Surname
* *Telephone* number
* BirthDate
* Email
* Gender (m.b.v. Radio-button)
* DeveloperSkillLevel (dropdown)

o Import **ReactiveFormsModule** into your app.module.ts.

and Import FormGroup, FormBuilder and AbstractControl into your **app.component.ts**.

import {FormGroup, FormBuilder, Validators, AbstractControl} from '@angular/forms';

o Add the [formGroup]="..." directive to the <form> tag, add formControlName="..." to the individual controls.  
o Import FormGroup and FormBuilder into your class and build the form, based on the layout of your HTML.  
o Submit the form and write the value to an alert box or to the console.  
o Demo code available at **examples//F01-forms-model-driven, Component 1**

**7) Goal: add validation to model driven forms**

o Import Validators into your app.component.ts.

import {FormGroup, FormBuilder, Validators, AbstractControl} from '@angular/forms';

o Add ‘default’ validations to the form fields.

o Add ‘custom’ validations to the form fields.

* Validate your *email pattern, unique username (names from an array), telephone number (min 10, max 15).*

Demo code available at **examples//F01-forms-model-driven, Component 2**

**8) unittest model-driven-forms**

See: <https://codecraft.tv/courses/angular/unit-testing/model-driven-forms/>

**9) FormArray**

Add an address list to the existing form. Each address has:

* Street
* House number
* Postal Code

Also ensure that the user can add **new addresses** and remove existing addresses.

Write an AddressValidator. Requirements: minimal 2 addresses and a unique Postal Code.

And validate the *postal Code (regex: /^[1-9][0-9]{3} ?(?!sa|sd|ss)[a-z]{2}$/i )*

**10) Form-type-ahead:**

Experiment with **form-type-ahead**:

<https://ng-bootstrap.github.io/#/components/typeahead/examples>