Assignment 8

Recommended readings:

- Lecture Slides as starting literature
- Assignment 8 – Angular Guide available on Moodle
- https://www.typescriptlang.org/docs/home.html
- https://docs.angularjs.org/guide/introduction
- AngularJS Quick Guide

Exercise 1 – TypeScript

1. a) Use `npm` to install TypeScript. Explain the contents of `ex1/tsconfig.json`. Compile the provided file `hello.ts` to JavaScript. Compare the results to the source and explain the differences. Take the resulting JavaScript file and execute it in browser e.g. by pasting its contents into console.
   Hint: with `tsconfig.json` in `ex1`, `tsc` can be run without parameters from the `ex1` directory. All configuration (including the directories for input and output files) is taken from that file.
   b) Change line 13 in `hello.ts` into

   ```
   let y: string = x.a;
   ```

   Recompile the file. What happened and why?
2. Implement the Image-Video class hierarchy from Assignments 4-5 using TypeScript. Note that the name `Image` conflicts with a predefined name in TypeScript environment, so use e.g. `MyImage` instead. Make sure that the following requirements are met:
   - Every attribute for both `MyImage` and `Video` must have both getter and setter and be properly encapsulated;
   - Other elements of the classes must also have proper visibility modifiers and a proper type;
   - Every image and video must now have a name holding a string value, all other attributes (`width`, `height`, `numberOfFrames` etc.) are assumed to be of number type.
3. Create an additional attribute called `type` for `MyImage` which is an enumeration with values corresponding to the common image formats (JPG, GIF, PNG). Provide setter and getter and test this attribute by passing correct and incorrect values to the setter. Which problems can we see now with the design of our class hierarchy?
4. a) Create `Rectangular` and `Printable` interfaces, where `Rectangular` provides `width` and `height` properties and `Printable` provides function declarations for `print()` and `printMore()`. Make `print()` mandatory and `printMore()` optional.

b) Change the hierarchy to make both `MyImage` and `Video` implement `Rectangular` and `Printable` and to eliminate inheritance of `Video` from `MyImage`. Implement properties with setters and getters.

c) Create a class `Printer` with a static function `printAll()` which prints an array of `Printable` references, test it with a mixed array of references to `MyImage`s and `Video`s.

d) Create another class `WidthCalculator` with a static function `getTotalWidth()`, which returns a total width for the array of `Rectangular` references, test it with the same mixed array.

e) Comment out the `print()` implementation in `MyImage`. Recompile and explain the results.

Exercise 2 – Getting started with Angular

1. Install Angular CLI to be globally available in the Node.js environment.
2. Generate the application called `ex2` by running `ng new ex2` in the directory under which the project is going to be placed, accept the default answers to all the questions.
3. Run the application with Angular CLI (ng) by running `ng serve` from the project directory (ex2). Note: there could be errors if the project directory is under a symbolic link (or file system junction under Windows), or a symbolic link itself, check that it is not the case in case of such errors.
4. Enable support for Material Design framework to ensure uniform visual appearance (we will not deal with visual representation further in this Assignment) by running `ng add @angular/material`, accept the default answers to all the questions.
5. Replace the default page with the page showing the following message: "*applicationName* is running", where the *applicationName* is defined as an attribute of the class responsible for rendering the default page of the application (`AppComponent` class defined within `app.component.ts`).
5. Declare a class to represent a book in the library e.g. taking the subset of the syntax from the library JSON of the Assignment 7 containing a title, a list of authors, and a rented status. Make authors and rented fields optional. Create an array of books in the `AppComponent` class which is available immediately after initializing the class. Hint: it can be done in its constructor.
6. Show the list of books from that array on the web page, represent a book as a , and the authors as the elements of the bulleted list `<ul></ul>`.
7. Change the code to show the rented book title in green followed with the text "(rented)".
8. Build the application for production and deploy it to the Apache web server by running `ng build --prod` and copying the contents of the `ex2/dist` directory under its document root. Note that if the deployment is going to the directory under the document root, the name of this directory has to be specified as an additional parameter to the build command as follows: `ng build --prod --base-href=/`*`directory-name`*`/`

Note: further exercises will require Angular framework to be available for the application. You can repeat step 2 before working on every exercise and then add the code where needed, or you can copy the solution of the previous exercise to the new project and start working over it replacing the code where necessary. If you do not want to copy the contents of `node_modules` directory every time, you can skip it, then you must run `npm install` in the project directory before running `ng serve`; this also must be done if `node_modules` got deleted for some reason.

Common note to the Exercises 3 and 4

Exercises 3 and 4 will use the shortened JSON file of the Assignment 7 residing in the `assets` directory as a source of data. To make use of this file in Angular application (without reading it from disk at runtime as it is not possible in a browser environment), perform the following steps:

1. Use the provided `model/book.ts` file containing the `Book` class corresponding to the JSON data
2. Add the following two lines into the `compilerOptions` section of the `tsconfig.json` file

```
"compilerOptions": {
  ...
  "resolveJsonModule": true,
  "esModuleInterop": true,
  ...
      }
```

   This will allow TypeScript to read JSON file as the fragment of the legitimate code.

3. Add the file to the main component file the same way as if it were another Angular component or data structure:

```
import globalBookList from '../assets/book_library.json';
import {Book} from "./model/book";
// now globalBookList contains the data from the json file
export class AppComponent {
   bookList: Book[] = [];
   constructor() {
       this.bookList = globalBookList;
  }
}
```

Exercise 3 – Angular Components and Forms

1. Show the list of books in the library (see Exercise 2), where the list of all books and every book within the list is displayed by its own component. The component for the book list should not accept parameters, the component for the book should accept one input parameter corresponding to the book being displayed. Generate your components with Angular CLI running `ng generate component` *`component-name`*, accepting all default answers.
2. Create a form which allows for adding new books. Every book is defined by its title, ISBN and rented status. The title and ISBN are required elements. ISBNs must be unique: if the ISBN for the new book exists in a list, it should not be added, and an error message should be displayed below the form. The rented status should be specified by means of a checkbox. The added book should be immediately shown in a list. Hint: the form can be created within the book list component.
3. Add the possibility for renting individual books. Add “rent” or “return” button to the book display depending on its status (i.e. “rent” is visible when the book can be rented, “return” – when it can be returned). Clicking on such buttons must immediately affect the display of this book.
4. Inform the book list component about the event of renting or returning a specific book. Show the message “*Title* was rented” or “*Title* was returned” above the book list in case of either

event. Clear the previous message on issuing the next one. Implement this functionality by means of output parameters and event emitters.

5. Add the dynamic filter to the book list. Place the input control above the list and filter the list to show only the books with the title containing the filter string (case-insensitively). Change the list on every change of the filter value.
6. Extend the form for adding new books with the means of adding book authors. As every book can have more than one author, this can be implemented by adding the input field for entering the name of the new author, the list of currently entered authors, and two buttons "add author" and "delete author". Clicking on the first button adds author name to the list (duplicate names should not be allowed), clicking on the second button deleted the currently selected author from the list. The list of authors which is formed this way has to be always connected to the newly added book, so on submitting the form such list becomes the part of the book added to the book list and should be displayed immediately. Use two-way input binding to implement such functionality, so the changes in one elements of the form are immediately reflected in others.

Exercise 4 – Angular Routing and Services

1. Implement the book list application which includes two pages accessible with different URLs: one for the main display (book list component), another for the information about the library application (the "about" page). Make every page a separate component, use hash-based routing to switch between pages, include page links within the view for the application component (below the "running application" message). Hint: to turn on hash-based routing, add the parameter to the import declaration of the `RouterModule` in `app.module.ts`:

```
imports: [ ... RouterModule.forRoot(ROUTES, { useHash: true }) ... ]
```

2. Encapsulate the book list in a service. Implement `getBookList()`, `addBook()`, `rentBook()`, `returnBook()`, and `filterBooksByTitle()` service methods. Inject this service in the book list component. Make the book list component use service methods instead of using book list directly.
3. Implement the detail view for the book display. Add the "detail" link to the book display and create an additional component to show book details accessible by following this link. Make this component accessible by the route parameterized by the book ISBN. Use the book list service to provide the necessary book to the component, implement `getBookByISBN()` service method for this. The component should be shown in place of the main or help component i.e. below the page links. Test the component by specifying the URL http://localhost:4200/#/book/*isbn* in a browser address line: what will happen?
   Hint: to get the value of the route parameter, inject the `ActivatedRoute` object into the component and call `route.snapshot.paramMap.get(`*`parameter-name`*`)` to get the route parameter from the injected route object. It can be done e.g. in the component constructor.

Exercise 5 – REST Client in Angular

Preliminaries: for completing this exercise, a running instance of the REST service implemented according to Assignment 7, exercises 4-5 is necessary. It is available on Moodle as the "Aufgabenblatt 7 REST service" link.

1. Implement the functionality to get the list of all books, add a new book, rent and return a book, get book by ISBN, and filter books by title making use of the REST service mentioned above. Therefore:
   a) implement the book list service calling the REST service of the Assessment 7 which provides the above calls. Make every method of this service return `Observable<type>` where *type* is the data to be returned by the method. Do not catch errors in the service.
   b) Make use of this service in the application (in the book list component, book component, and book detail component). Set values of the component properties in the handlers of successful completion of the calls, catch errors returned by the REST service to handle possible errors (e.g. when no elements are returned by the filter call). While handling the dynamic filtering issue queries on every value change of the corresponding filtering field (hint: this can be done by handling the input event for this field: `<input type="text" ... (input)="handler()">`

2. Implement filtering by author name. To do so, add new `filterByAuthorName()` method to the service which calls the corresponding REST API call, add the second filter input field above the book list, and call this method in the (input) event handler of this input field.
3. Allow deleting a book, by extending the service with the `deleteBook()` method, using the corresponding REST API call. Add new “delete” button to the book display, in the click handler of this button, use a dialog to confirm that the book should be deleted (e.g. by dynamically adding a div with this question to the book display), and call the service in the positive answer handler.