Angular
Lab 8
RxJS and HttpClient
Exercises

1 Lab setup

2 Generating a new Angular project

3 Creating and consuming observables

4 Working with JSON

5 Accessing JSON from a public REST API

6 Using HttpClient to send HTTP GET requests

Q1: Create a class Todo.ts which provides the structure for the JSON content returned from a call to retrieve a single post, for e.g.
https://jsonplaceholder.typicode.com/todos/3

Q2: Add a method `getSingleTodo` to FakeAPIService which makes a HTTP GET call to the URL above and returns a single Todo item. The id of the Todo item to be retrieved should be passed as an argument to `getSingleTodo`

Q3. Implement a method `retrieveTodo` in the root component which will make a call to the method from Q2 and store it in a new property `singleTodo`. Register only 2 callbacks for the subscribe call within this method (one to handle successful emissions and the other to handle errors in the response). Error messages are stored in the existing property `errorMessage`

Q4: Add additional HTML at the end of the root template to

- add a numeric input field to obtain id of the Todo to retrieve from the user
- add a button to trigger this retrieval process and perform event binding to the method `retrieveTodo` from Q3
- display the property `errorMessage`
- display the fields from `singleTodo` appropriately if it has valid content

Q5. Create an interface `User.ts` which provides the partial structure from the JSON content returned from a call to retrieve a single user, for e.g.
https://jsonplaceholder.typicode.com/users/1
The partial structure should include only the following fields:

```
{
  "id": 1,
  "name": "Leanne Graham",
  "username": "Bret",
  "email": "Sincere@april.biz",
  "address": {
    "street": "Kulas Light",
    "suite": "Apt. 556",
    "city": "Gwenborough",
  },
  "website": "hildegard.org",
}
```

Q6: Add a method `getAllUsers` to FakeAPIService which makes a HTTP GET call to the URL: https://jsonplaceholder.typicode.com/users in order to return a list of Users as an array.

Q7. Add a method `retrieveAllUsers` to the root component that will call `getAllUsers` of Q6 and register a single callback to handle the return of successful response and store the array of Users into a new property `allUsers`.

Q8: Add additional HTML at the end of the root template to create a table that will list the details of all Users returned from the call in Q7 via a ngFor directive. You can combine all the 3 fields of the `address` property into a single string to simplify the rendering of table content. Provide an additional button with an event binding to `retrieveAllUsers` from Q7 that will trigger this retrieval.

Q9. The `userId` can be supplied as a query parameter to the URL path that retrieves the list of all posts in order to return a subset of posts. For e.g.
`https://jsonplaceholder.typicode.com/posts?userId=3`
returns a subset of posts from the overall list of 100 posts related to the user with the given `userId`. These posts have an `id` field that increases by 1 for each consecutive post.

Add a method `getSomePosts` to FakeAPIService which makes a HTTP GET call to the URL: `https://jsonplaceholder.typicode.com/posts?userId=xxx` in order to return this subset of posts as an array. The value for userId should be passed as an argument to this method.

Q10.
Implement a method `retrieveSomePosts` to the root component that will call `getSomePosts` of Q9 and register a single callback to handle successful emissions. Create a new component property `selectUserId` that will hold the value that is to be passed as the argument to the `getSomePosts` method (the value for this property will be provided by the template in Q11).

We can filter the subset of posts returned from:
`https://jsonplaceholder.typicode.com/posts?userId=xxxx`
to get an even smaller subset of posts by specifying a lower and upper boundary for the `id` field. Assume that these values are stored in the new properties `upperId` and `lowerId` whose values will be obtained from the template in Q11.
As an example, with an initial call to
https://jsonplaceholder.typicode.com/posts?userId=3
and having the values `lowerId=22` and `upperId=24`, the final subset of posts we would get is:

```
  {
    "userId": 3,
    "id": 22,
    "title": "dolor sint quo a velit explicabo quia nam",
    "body": "eos qui et ipsum ipsam suscipit aut\nsed omnis non
odio\nexpedita earum mollitia molestiae aut atque rem suscipit\nnam
impedit esse"
  },
  {
    "userId": 3,
    "id": 23,
    "title": "maxime id vitae nihil numquam",
    "body": "veritatis unde neque eligendi\nquae quod architecto quo
neque vitae\nest illo sit tempora doloremque fugit quod\net et vel
beatae sequi ullam sed tenetur perspiciatis"
  },
  {
    "userId": 3,
    "id": 24,
    "title": "autem hic labore sunt dolores incidunt",
    "body": "enim et ex nulla\nomnis voluptas quia qui\nvoluptatem
consequatur numquam aliquam sunt\ntotam recusandae id dignissimos
aut sed asperiores deserunt"
  },
```

Provide an implementation for the callback to handle successful emission to obtain the final smaller subset of posts. This final smaller subset of posts should be stored in a new property `subsetPosts` Error messages are stored in the existing property `errorMessage` in the callback to handle errors

Q11. Add additional numeric input fields at the end of the root template to obtain 3 values

- the userId
- the upper boundary id
- the lower boundary id

Use two way binding to bind these input fields to the following new component properties declared in Q10: `selectUserId`, `upperId`, `lowerId`
Add a button and perform event binding to the method `retrieveSomePosts` from Q10.
Create a table that will list the details of Users contained in the property `subsetPosts` that was computed in Q10 via a ngFor directive.

7 Implementing a local fake API service using JSON Server

8 Accessing a local API service from an Angular app

Q1: Add a method `getHeroesWithJob` to LocalAPIService that accepts a single string parameter `job` which makes a HTTP GET call to the URL above and returns an array of Heroes with that `job`. For e.g. typing the URL:

http://localhost:3000/heroes?job=industrialist

returns a response of

```
[
  {
    "id": "2",
    "firstName": "Tony",
    "lastName": "Stark",
    "age": 45,
    "married": true,
    "job": "industrialist"
  }
]
```

Q2. Create a property in AppComponent called `heroesWithJob` with is an array of hero Objects. Add a method `getHeroesWithJob` to AppComponent which receives a parameter that represents the job, makes a call to `getHeroesWithJob` in LocalAPIService and stores the return result in `heroesWithJob`.

Register only 2 callbacks for the subscribe call within this method (one to handle successful emissions and the other to handle errors in the response). Error messages are stored in the existing property `errorMessageForHeroesWithJobs`

Q3. Add additional HTML at the end of the root template to

- add a text input field to obtain job of the hero to retrieve from the user
- add a button to trigger this retrieval process and perform event binding to the method `getHeroesWithJob` from Q2
- display the property `errorMessageForHeroesWithJobs` if appropriate
- display all the Hero objects in the array `heroesWithJob` with their properties arranged properly in a table format

9 Using other REST API methods (POST, PUT, DELETE)

Q1. Add a method `deleteExistingHero` to LocalAPIService that accepts a single number parameter `heroId` which makes a HTTP DELETE call to the URL of the service in the form of:

`http://localhost:3000/heroes/`*`heroId`*

which requests the backend service to delete the Hero record/object with value `heroId`

Q2. Create a new property `deleteMessage` to store messages pertaining to the operation below. Add a method `deleteHeroUsingID` to AppComponent which receives a parameter that is the Hero ID, makes a call to `deleteExistingHero` in LocalAPIService. Register only 2 callbacks for the subscribe call within this method (one to handle successful emissions and the other to handle errors in the response). Error messages are stored in the existing property `deleteMessage`

Q3. Add additional HTML at the end of the root template to

- add a numeric input field to obtain hero id of the hero to delete from the backend service
- add a button to trigger this retrieval process and perform event binding to the method `deleteHeroUsingID` from Q2
- display the property `deleteMessage` if appropriate

When you are done, test out the delete functionality by attempting to retrieve all heroes. Check the contents of the `heroes.json` file in the `localserver` directory to confirm that the specified object has in fact been deleted.