Angular
Lab 8
RxJS and HttpClient

1 Lab setup

2 Working with JSON

3 Accessing JSON from a public REST API

4 Generating a new Angular project

5 Creating and consuming observables

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.

6.1 Todo.ts

```
// Q1
export class Todo {
    userId: number;
    id: number;
    title: string;
    completed: boolean;

    constructor(userId: number, id: number, title: string, completed: boolean) {
        this.userId = userId;
        this.id = id;
        this.title = title;
        this.completed = completed;
    }
}
```

6.2 fakeAPI.service.ts

```
// Q2
import { Todo } from './Todo';

    // Q2
    getSingleTodo(todoId: number) : Observable<Todo> {
        let finalUrl = this.baseURL + 'todos/' + todoId;
        console.log("Sending GET request to : ",finalUrl);
        return this.http.get<Todo>(finalUrl);
    }

// Q6
import { User } from './User';

    // Q6
    getAllUsers() : Observable<User[]> {
        let finalUrl = this.baseURL + 'users';
        console.log("Sending GET request to : ",finalUrl);
        return this.http.get<User[]>(finalUrl);
    }


    // Q9
    getSomePosts(userId: number) : Observable<Post[]> {
        let finalUrl = this.baseURL + 'posts?userId=' + userId;
        console.log("Sending GET request to : ",finalUrl);
        return this.http.get<Post[]>(finalUrl);
    }
```

6.3 User.ts

```
// Q5
export interface User {
    id: number;
    name : string;
    userName : string;
    email : string;
    address : Address;
    website : string;
}
```

```
interface Address {
    street : string;
    suite : string;
    city : string;
}
```

6.4 app.component.html

```
<!-- Q4 -->
<hr>

<h3>Retrieving Todo via user-specified todoId</h3>

<label for="todoid">Enter id of Todo to retrieve:</label>
<input #todoinput type="number" id="todoid">

<button type="button" (click)="retrieveTodo(todoinput.value)">Retrieve
Todo</button>

<p>{{ errorMessage }}</p>

<div *ngIf="singleTodo">
    <p>User Id : {{singleTodo.userId}} </p>
    <p>Id : {{singleTodo.id}} </p>
    <p>Title : {{singleTodo.title}} </p>
    <p>Completed : {{singleTodo.completed}} </p>
</div>



<!-- Q8 -->

<hr>

<h3>Retrieving all users and displaying in a table</h3>


<button type="button" (click)="retrieveAllUsers()">Retrieve all
users</button>

<table class='table'>
    <thead>
        <tr>
            <th>User Id</th>
            <th>Name </th>
            <th>Username </th>
            <th>Email</th>
            <th>Address</th>
            <th>Website</th>
        </tr>
    </thead>
    <tbody>
        <tr *ngFor="let user of allUsers">
            <td>{{user.id}}</td>
            <td>{{user.name}}</td>
            <td>{{user.username}}</td>
            <td>{{user.email}}</td>
            <td>{{user.address.suite + ' ' + user.address.street + ' ' +
user.address.city}}</td>
            <td>{{user.website}}</td>
        </tr>
    </tbody>
</table>

<!-- Q11 -->

<hr>

<h3>Retrieving subset of posts using userId, upper and lower boundary for
id</h3>

<label for="selectUserId">Enter user Id for subset of posts to
retrieve:</label>
<input type="number" id="selectUserId" [(ngModel)]="selectUserId">

<br>

<label for="lowerId">Enter lower boundary id for subset of posts to
retrieve:</label>
<input type="number" id="lowerId" [(ngModel)]="lowerId">

<br>

<label for="upperId">Enter upper boundary id for subset of posts to
retrieve:</label>
<input type="number" id="upperId" [(ngModel)]="upperId">

<br>

<button type="button" (click)="retrieveSomePosts()">Retrieve a subset of
posts</button>
```

```
<table class='table'>
    <thead>
        <tr>
            <th>User Id</th>
            <th>Post Id</th>
            <th>Title</th>
            <th>Body</th>
        </tr>
    </thead>
    <tbody>
        <tr *ngFor="let post of subsetPosts">
            <td>{{post.userId}}</td>
            <td>{{post.id}}</td>
            <td>{{post.title}}</td>
            <td>{{post.body}}</td>
        </tr>
    </tbody>
</table>
```

6.5 app.component.ts

```
// Q11
import { Component } from '@angular/core';
import { RouterOutlet } from '@angular/router';
import { FormsModule } from '@angular/forms';

@Component({
 selector: 'app-root',
 standalone: true,
 imports: [RouterOutlet, FormsModule],
 templateUrl: './app.component.html',
 styleUrl: './app.component.css'
})

//Q3
import { Todo } from './Todo';

  //Q3
  singleTodo!: Todo | undefined;

  retrieveTodo(todoId : string) {
    this.errorMessage = "";

    this.fakeAPIService.getSingleTodo(parseInt(todoId)).subscribe({

      next: (val:Todo) => {
        this.singleTodo = { ...val};
      },

      error: (errorVal: HttpErrorResponse) =>
{
        console.error('Request failed !');
        this.errorMessage = "GET request failed with error code " +
errorVal.status;
      },
    });
  }



//Q7
import { User } from './User';

  // Q7
  allUsers: User[] = [];
  retrieveAllUsers() {

    this.fakeAPIService.getAllUsers().subscribe(
      (val:User[]) => {
        this.allUsers = val;
      }
    );

  }

  // Q10

  selectUserId = 0;
  upperId = 0;
  lowerId = 0;
  subsetPosts: Post[] = [];

  retrieveSomePosts() {

    this.fakeAPIService.getSomePosts(this.selectUserId).subscribe(

      (originalPosts :Post[]) => {

        this.subsetPosts = originalPosts.filter(post => post.id >=
this.lowerId && post.id <= this.upperId)
```

```
      },

    );

  }
```

7 Implementing a local fake API service using JSON Server

8 Accessing a local API service from an Angular app

8.1 localAPI.service.ts

```
  // Q1
  getHeroesWithJob(job : string) : Observable<Hero[]> {

      let queryString = '';
      if (job) {
          queryString += "job=" + job;
      }

      let finalUrl = this.baseURL + '?' + queryString;
      console.log("Sending GET request to : ",finalUrl);
      return this.http.get<Hero[]>(finalUrl);
  }
```

8.2 app.component.ts

```
// Q2

// This is intended to hold all Heroes returned from a call to:
  //http://localhost:3000/heroes?job=???
  heroesWithJob: Hero[] = [];
  errorMessageForHeroesWithJobs = "";

  getHeroesWithJob(job : string) {

    this.errorMessageForHeroesWithJobs = "";
    job = job.trim();

    this.localAPIService.getHeroesWithJob(job).subscribe({

      // Handle successful emissions
        next: (val:Hero[]) => {
          this.heroesWithJob = val;
        },

        // Handle errors in response
        error: (errorVal: HttpErrorResponse) => {

          console.error('Request failed !');
          console.log("The HTTP error code is ",
errorVal.status);

          if (errorVal.status === 404)
            this.errorMessageForHeroesWithJobs = "The server does not seem
to have the heroes";
          else if (errorVal.status === 500)
            this.errorMessageForHeroesWithJobs = "The backend server seems
to be down";
          else if (errorVal.status === 401 || errorVal.status === 401)
            this.errorMessageForHeroesWithJobs = "Server requires
authentication and / or authorization";
          else if (errorVal.status === 0)
            this.errorMessageForHeroesWithJobs = "No network connection to
server";
          // include more else if messages to cater for different status
codes
          else
            this.errorMessageForHeroesWithJobs = "Unknown error in network
request";

        },

        // Handle completion of stream
        // complete : () => {
        //   console.log('Request completed');
        // }
      }
    );


  }
```

8.3 app.component.html

```

<!-- Answer for Q3 -->

<h3>Retrieving heroes based on jobs</h3>

<label for="job">Enter hero's job :</label>
<input #job type="text" id="job">
<br>

<button type="button" (click)="getHeroesWithJob(job.value)">Retrieve
heroes</button>

<p>{{ errorMessageForHeroesWithJobs }}</p>

<table class='table'>
    <thead>
        <tr>
            <th>ID</th>
            <th>First Name</th>
            <th>Last Name</th>
            <th>Age</th>
            <th>Married</th>
            <th>Job</th>
        </tr>
    </thead>
    <tbody>
      @for (hero of heroesWithJob; track hero) {
        <tr>
          <td>{{hero.id}}</td>
          <td>{{hero.firstName}}</td>
          <td>{{hero.lastName}}</td>
          <td>{{hero.age}}</td>
          <td>{{hero.married}}</td>
          <td>{{hero.job}}</td>
      </tr>
      }
    </tbody>
</table>
```

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

9.1 localAPI.service.ts

```
    // Q1
    deleteExistingHero(heroId: number) : Observable<any> {

        // For DELETE, we just need the id of a resource that we want to
delete
        // no body in the request is required
        let finalUrl = this.baseUrl + "/" + heroId;

        return this.http.delete(finalUrl);

    }
```

9.2 app.component.ts

```
  // Q2

  deleteMessage!: string;

  deleteHeroUsingID(heroId: string) {

    this.deleteMessage = "";

    this.localAPIService.deleteExistingHero(parseInt(heroId)).subscribe({
      // Handle successful emissions
      next: (resp: HttpResponse<any>) => {
        console.log("The response received back is ", resp);
        this.deleteMessage = "Hero deleted successfully";
      },
      // Handle errors in response
      error: (errorVal: HttpErrorResponse) => {
        console.error('Request failed with response ', errorVal);
        this.deleteMessage = "The hero with id " + heroId + " does not
exist";
      },

      // Handle completion of stream
/*         complete: () => {
        console.log('Request completed');
      } */

    });


  }
```

9.3 app.component.html

```
<!-- Q3
 -->

 <h3>Delete an existing hero</h3>

 <label for="heroToDelete">Enter hero id :</label>
 <input #heroToDelete type="number" id="heroToDelete">
 <br>

 <button type="button"
(click)="deleteHeroUsingID(heroToDelete.value)">Delete hero</button>

 <p>{{ deleteMessage }}</p>
```