Angular 2
HTTP REST Operations & Observables

Sang Shin
JPassion.com
"Code with Passion!"

Topics

- Setting up REST server
- Asynch. response handling in Angular 2
- Angular 2 Http service
- Observable operators
- Getting an item (HTTP GET)
- Posting an item (HTTP POST)
- Deleting an item (HTTP DELETE)
- Getting list of items (HTTP GET)
- Error handling on Observable

Setting up REST server

Lab: Start local REST server (for testing)

- Install json-server
  - > npm install -g json-server
- Create rest data file – let's call it db.json

```
{
  "users": [
    { "id": 1, "name": "sang", "age": 34 },
    { "id": 2, "name": "john", "age": 23 },
    { "id": 3, "name": "mary", "age": 99 },
    { "id": 4 "name": "mist", "age": 16 }
  ]
}
```

- Run local rest server
  - > json-server db.json
- You can now access the data via
  - > http://localhost:3000/users
  - > http://localhost:3000/users/1

Lab: Create some data in Firebase

- Firebase server is freely available online NoSQL server for posting and reading your own data via REST
  - https://www.firebase.com/
- You can use Firebase as test REST server

Asynch. Response Handling in Angular 2

Angular 2 supports Observable

- Angular 2 uses Reactive Extensions for JavaScript (RxJs) library for asynchronous operations
  - RxJs provides an event-driven stream of notifications to a caller by issuing *Observable* return type

- The caller performs *subscribe* method against the returned *Observable* object passing data handler and error handlers as arguments
  - The data handler will be invoked when data is received later on

- Why do we care?
  - Because REST methods (of Angular provided Http service) return Observable as return type

Angular 2 Http Service

Angular has built-in Http Service

- The built-in Http service (Anglar 4.2.x and below) or HttpClient service (Angular 4.3.x+) has methods
  - get, post, put, delete, head, patch
- If you are using Angular 4.3.x+
  - import { HttpClient } from '@angular/common/http';
- These methods return Observable<Response> object
- You need to call subscribe method of the returned Observable in order to receive the data asynchronously
- The subscribe method takes 3 handlers as arguments
  - data handler (most important and mandatory argument)
  - error handler (optional argument)
  - completion handler (optional argument)

Getting an item

Lab: Get an item via HTTP GET

- Create UserService – this is an intermediary service that uses Angular 2 Http service
  - ng g service user
- Get Angular 2 Http service injected into the UserService
- Make a GET call using Angular 2 Http service
  - this.http.get("http://localhost:3000/users/1")
  - this.http.get("https://test-3b25d.firebaseio.com/title.json")
- Subscribe the Observable
  - subscribe((response:Response) => console.log(response))
  - subscribe((response:Response) => console.log(response.json()) // json() method of the Response object converts the body of the response to JSON object (Only for Angular 4.2.x and below)

Observable Operators

Observable operators

- Observable operators can be applied to the items emitted by an Observable
  - > Each operator in turn returns Observable – so these operators can be chained
  - > Each operator in the chain modifies the Observable that results from the operation of the previous operator
  - > In order to use them, you need to import 'rxjs/rx”
- map operator is most useful
  - > It transform the items emitted by an Observable by applying a function to each of them

Lab: Apply map operator

- Add map operator to convert received data into json (only for Angular 4.2.x and below)

```
import 'rxjs/rx'

getData() {
  return this.http.get("https://test-3b25d.firebaseio.com/title.json")
  .map((response:Response) => response.json());
}
```

- Modify the data handler in the subscribe method to receive any type (instead of Response type) and does not need to call json()

```
getUser(id) {
   this.restService.getUser(id)
     .subscribe((data: any) => this.name = data.name
     );
 }
```

Post an item

Post an item

- You have to provide post data
- Angular provides *Headers* class for setting request headers if needed

```
const headers: Headers = new Headers();
headers.append('Accept', 'application/json');
return this.http.post("http://localhost:3000/users", user, {
  headers: headers
})
```

Get list of items

Get list of items (when using JSON serve

- In the RestService

```
getAllUsers(){
  return this.http.get(this.REMOTE_URL)
  .map((data: Response) => data.json());
}
```

- In the Component code that uses the RestService

```
users: User[] = [];

getAllUsers(){
  this.restService.getAllUsers()
   .subscribe((data: any) => this.users = data);
}
```

Get list of items (when using Firebase)

- You might want create a new array from the response

```
getAllUsers(){
  this.restService.getAllUsers()
        .subscribe(response => {
      console.log(response);
      let myArray: any[] = [];
      for (let key in response){
        myArray.push(response[key].username);
      }
      this.users = myArray;
    })
}
```

Delete an item

Delete an item

```
// local rest server example
deleteUser(id){
  return this.http.delete("http://localhost:3000/users/”+ id)
  .map((response: Response) => response.json());
}

// Firebase rest server example
deleteUser(userKey){
  return this.http.delete("https://test-3b25d.firebaseio.com/users"+"/"+
  userKey.name+".json")
  .map((response: Response) => response.json());
}
```

Error Handling

Error Handling in the subscribe method

- The subscribe method can take error handler as a second argument

```
getUser(id) {
this.restService.getUser(id)
  .subscribe((data: any) => this.name = data.name,
                (error:any) => this.errorMessage = error
  );
}
```

Code with Passion!
JPassion.com