# Angular Best Friends

## Module 8 Exercise 1 – Basic API calls

## Goals

We start off with a blank application where everything is hard coded. The goal of this exercise is to configure the data service to use an HttpClient and implement basic API calls to perform CRUD operations on books.

## Steps

### Setting up the environment

1. Go to the **Module8Exercise1** folder. Open the “**initial**” folder in VS Code.
2. Open the terminal and run **npm install** to install the needed node modules.
3. This app is slightly different than all others because it also has a NodeJS server that serves the initial index.html file and that has API endpoints to play around with resources. So to build everything up run **npm start**.
4. Wait until you see the Angular specific bundles in the terminal.
5. Open a browser tab and navigate to **localhost:3000** . The NodeJs server listens to this port and this is how we run this Angular application

### Setting up for the HTTP client

1. Go to the **app.module.ts** file. This is where we’ll import the HttpClientModule. We’ll do this with an import statement. Therefore, somewhere in the imports part also add this line of code:

import { HttpClientModule } from '@angular/common/http';

1. Add the HttpClientModule to the imports array of the NgModule declaration. The imports array should look like this:

imports: [

BrowserModule,

AppRoutingModule,

FormsModule,

CoreModule,

HttpClientModule

],

1. That’s all on importing the module. It will now be available throughout the app. We’ll use it in the data service. So go to the **core** folder and find the **data.service.ts** file.
2. Since the HttpClientModule is already imported in the root module here we need to import the HttpClient class. So add an import somewhere at the top:

import { HttpClient} from '@angular/common/http';

1. Next we need to inject the HttpClient in the constructor. This is how the constructor would looklike:

constructor(private http: HttpClient) { }

1. We’re almost ready to use the HttpClient. One last thing to add during this setting up part is adding an import for RxJS observables. This is needed since from an HttpClient call we’ll mostly return observables. So add the following import:

import { Observable } from 'rxjs';

### Making API calls

Now we are ready to make some API calls.

1. In the same **data.service.ts** file locate the **getAllBooks()** function. For now it retrieve some hard coded data. But we can use the HttpClient to make a real call to our back end API.
2. Delete the entire function and add this one instead:

getAllBooks(): Observable<Book[]> {

console.log('Getting all books from the server.');

return this.http.get<Book[]>('/api/books');

}

1. As you see we now use the httpClient. We also changed the return type of the method. Since the HttpClient is always returning Observables we also need to return an **Observable<Book[]>** instead of a simple **Book[]**.
2. A place where this method is consumed is the dashboard component. Locate the component in the “**dashboard**” folder. It is called **dashboard.component.ts**.
3. In **NgOnInit()** notice that we now have an error. This error is caused by the fact that we changed the return type of the **getAllBooks()** method.
4. We’ll have to fix this. So delete the entire line 24 and replace it with the following snippet:

this.dataService.getAllBooks()

.subscribe(

(data: Book[]) => this.allBooks = data,

(err: any) => console.log(err),

() => console.log('All done getting books.')

);

1. Just a few words on what’s happening here. First we subscribe to the Observable. In the subscribe function we can pass in 3 different callback functions. The first one will always be called when data is available. The second will be called if we have errors. The third will be called when everything is done. So we used arrow functions to take actions when each of these callbacks is executed. For the first one we of course set the value of our allBooks array to what we receive from the API. For the second we simply log the error to the console. And for the third callback we just write something to the console.
2. If the application is still running after step 5 just go to the browser, open the developer tools on the networking tab and refresh the page. You should see there a successful call to the API.
3. Now let’s retrieve a single book. So back to the **data.service.ts** file. Locate the getBookById method. Delete it and replace it with the following sample:

getOldBookById(id: number): Observable<Book> {

return this.http.get<Book>(`/api/books/${id}`)

}

1. This method is used in the edit-book.component.ts file which lies in the **edit-book** folder.
2. In NgOnInit delete line 21 and paste the following snippet instead:

this.dataService.getBookById(bookID)

.subscribe(

(data: Book) => this.selectedBook = data,

(err: any) => console.log(err)

);

1. Let’s now also perform other API operations. Go back to the data.service.ts file. Notice that we don’t have methods to add, update or delete books. So let’s add them right under the getBookById method:

addBook(newBook: Book): Observable<Book> {

return this.http.post<Book>('/api/books', newBook, {

headers: new HttpHeaders({

'Content-Type': 'application/json'

})

});

}

updateBook(updatedBook: Book): Observable<void> {

return this.http.put<void>(`/api/books/${updatedBook.bookID}`, updatedBook, {

headers: new HttpHeaders({

'Content-Type': 'application/json'

})

});

}

deleteBook(bookID: number): Observable<void> {

return this.http.delete<void>(`/api/books/${bookID}`);

}

1. What we have done here differently is that we have also added http headers. This causes some error since we haven’t imported the HttpHeaders class. So let’s import it where we have imported the HttpClient:

import { HttpClient, HttpHeaders } from '@angular/common/http';

1. Now we have to update the components that will actually use this methods. Let’s start with the **add-book** folder where we’ll find the **add-book.component.ts** file.
2. This component doesn’t use the data service yet. So we’ll have to import it:

import { DataService } from 'app/core/data.service';

1. …and inject it in the constructor:

constructor(private dataService: DataService) { }

1. Locate the saveBook method. Delete it and replace it with this snippet:

saveBook(formValues: any): void {

let newBook: Book = <Book>formValues;

newBook.bookID = 0;

console.log(newBook);

this.dataService.addBook(newBook)

.subscribe(

(data: Book) => console.log(data),

(err: any) => console.log(err)

);

}

1. Let’s move to the **edit-book** folder where we have the **edit-book.component.ts** file.
2. This component already uses the data service, so no import necessary. Locate the saveChanges method, delete it and place the following code snippet:

saveChanges(): void {

this.dataService.updateBook(this.selectedBook)

.subscribe(

(data: void) => console.log(`${this.selectedBook.title} updated successfully.`),

(err: any) => console.log(err)

);

}

1. Lastly, let’s head over to the dashboard.component.ts file. We have a deleteBook method there. Delete it and replace it with the following sample:

deleteBook(bookID: number): void {

this.dataService.deleteBook(bookID)

.subscribe(

(data: void) => {

let index: number = this.allBooks.findIndex(book => book.bookID === bookID);

this.allBooks.splice(index, 1);

},

(err: any) => console.log(err)

);

}

1. Now you can test in the application how these features are working and follow in the developer tools what calls are made, what gets returned and so on.