Lab 1 - Angular Communication with REST Services

The Angular HTTP client service provides the ability to interact with REST web services from Angular applications. It is a best practice to perform this interaction from a custom service.

In this lab you will implement various methods in a custom service that will provide data to a completed component to display this data. Once the methods of the service are implemented, the features of the component displaying data will become active.

Part 1 - Getting Started

To save time most of the application has already been implemented. You will setup this starter project.

__1. If you have not done so already create a **C:\LabWork** directory on your development machine.

__2. If you have not already done so, copy **C:\LabFiles\project-seed.zip** to the '**C:\LabWork**' folder.

__3. In the '**C:\LabWork**' folder, extract **project-seed.zip**.

__4. Rename the **project-seed** folder to **angular-REST**.

__5. Extract the file **C:\LabFiles\angular-REST-overlay.zip** directly into the **C:\LabWork\angular-REST** folder. Confirm that you want to overwrite files which will also let you know you are extracting in the correct location.

__6. Open a command prompt and go to the root folder of the project **C:\LabWork\angular-REST** .

__7. Start the Typescript compiler and embedded server using the following single command:

```
npm start
```

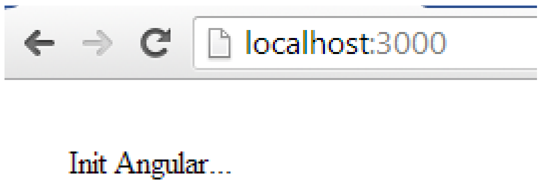
__8. Click **Allow access** if a window opens.

The above command will start the embedded server and run the Typescript compiler with the "w" - Watch option. In watch mode the Typescript compiler will keep an eye on files in the project directory and sub-directories and will re-run the compile if any are changed (and saved) or added.

__9. Open a browser and enter the following address.

```
http://localhost:3000/
```

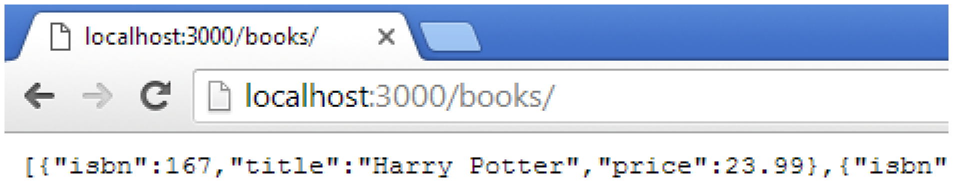
__10. Check that you briefly get a message about Angular initializing before seeing a blank screen. This is expected as currently the application doesn't have any data to display.



```
Init Angular...
```

__11. Change the address in the browser to the following which will show some data obtained directly from the REST service that is part of the application.

```
http://localhost:3000/books
```



```
[{"isbn":167,"title":"Harry Potter","price":23.99},{"isbn"
```

__12. Lab setup is now complete.

Part 2 - Retrieve List of Books

Since the REST service in the project is already providing data, the only thing to implement is the custom service using the Angular HTTP client service to retrieve it.

__1. Open the following file in your text editor (path is referenced from the project root):

```
\src\app\services\data.service.ts
```

Note: You can use NotepadPlus to open all the files during the course from C:\Software\NotepadPlus.

__2. Examine how the imports in the service component declaration import the relevant HTTP service module and some of the RxJS Observable API.

__3. Find the '**onError**' method near the end of the file. This method will become a basic error handling error.

__4. Modify the implementation of the method to match that shown below. This will extract some response details into a custom error message that is returned with an Observable that will throw the error.

```
onError(resp: any): Observable<any> {
  let msg = resp.status + ":" + resp.statusText;
  console.log(msg);
  return Observable.throw(msg);
}
```

__5. Find the '**getBooks**' method. Make sure it is the one with no parameters and implies that it will return multiple books.

__6. Modify the implementation of the method as shown in bold. This uses the HTTP service to send a 'GET' request to the same URL you tried earlier in the browser. It will map the response to the body of the response being returned.

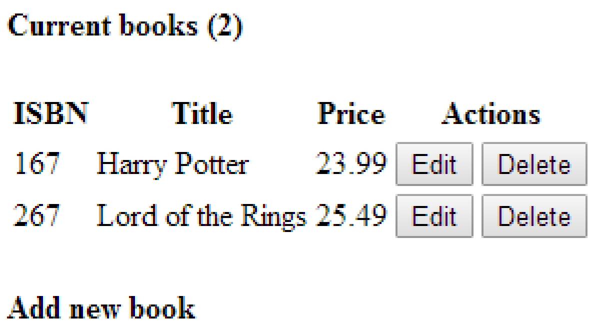
```
getBooks(): Observable<any> {
  return this.http.get('/books').map(resp => resp.json())
         .catch(this.onError);
}
```

__7. Save the file in the text editor.

__8. Return to the command prompt running the project and check that the files are compiled without any errors.

__9. Return to the browser and redisplay the main URL of the application. It should now display a list of 2 books as shown below. None of the other features work yet.

```
http://localhost:3000/
```



Current books (2)

ISBN	Title	Price	Actions	
167	Harry Potter	23.99	Edit	Delete
267	Lord of the Rings	25.49	Edit	Delete

Add new book

Part 3 - Implement Book Removal

The next feature that will be easy to implement will be the DELETE request to remove a book from the list.

__1. Return to the '\src\app\services\data.service.ts' file in the text editor, opening it if required.

__2. Find the '**mapResponseStatus**' method near the end of the file. This method will extract some details from a response that has no body and return a custom object if those details are required.

__3. Modify the implementation of the method to match that shown below. This will extract some response details into a custom object to be returned. This way the client of this custom service does not need to know the HTTP client API but can get some of the response details if required.

```
mapResponseStatus(resp: any): any {
  return {
    status: resp.status,
    statusText: resp.statusText,
    ok: resp.ok
  };
}
```

__4. Find the '**deleteBook**' method.

__5. Modify the implementation of the method as shown in bold. This uses the HTTP service to send a 'DELETE' request to a URL that includes the ISBN of the book. This is defined by the REST service as the way to identify which book is being deleted. Also, the 'mapResponseStatus' method is being used to map a successful response and uses the same 'onError' method to handle errors.

```
deleteBook(book: Book): Observable<any> {
  return this.http.delete(`/books/${book.isbn}`)
      .map(this.mapResponseStatus).catch(this.onError);
}
```

__6. Save the file in the text editor.

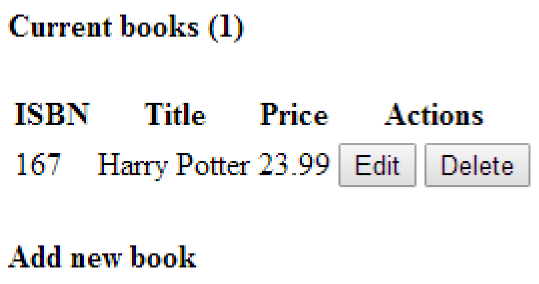
__7. Return to the command prompt running the project and check that the files are compiled without any errors.

__8. Return to the browser and refresh the main URL of the application. It should still display the same list of books.

```
http://localhost:3000/
```

__9. Click on the '**Delete**' button next to one of the entries in the list and then click on the '**OK**' button on the confirmation dialog to confirm you want to delete the item.

__10. Make sure the book is removed from the list.



Current books (1)

ISBN	Title	Price	Actions
167	Harry Potter	23.99	Edit Delete

Add new book

__11. Refresh the page again and make sure the list continues to display only one entry. It is best to avoid deleting the last book if possible.

Note: If while testing you no longer have any books in the list, you can always return to the command prompt running the project, use 'CTRL-C' to stop running the project, then run the 'npm start' command again to restart the project. The initial list of two books will be restored.

Part 4 - Retrieve Single Book

Another feature of the REST service allows the retrieval of a single book item. This is used to populate the edit form with the current details of a book. You will implement this feature even though you won't be able to submit any edited data.

__1. Return to the '\src\app\services\data.service.ts' file in the text editor, opening it if required.

__2. Find the '**getBook**' method. Make sure it is the one with a parameter for the ISBN of the book.

__3. Modify the implementation of the method as shown in bold. This uses the HTTP service to send a 'GET' request to a URL that includes the ISBN of the book. It will map the response to the body of the response being returned. In this case it will be the data for a single book.

```
  getBook(isbn: number): Observable<any> {
    return this.http.get('/books/${isbn}').map(resp => resp.json())
        .catch(this.onError);
  }
```

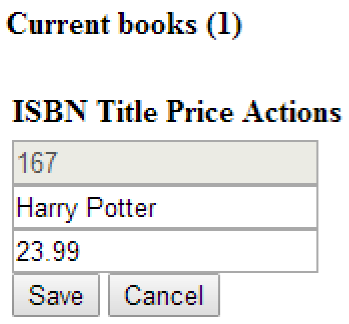
__4. Save the file in the text editor.

__5. Return to the command prompt running the project and check that the files are compiled without any errors.

__6. Return to the browser and refresh the main URL of the application. It should still display the same list of books.

```
http://localhost:3000/
```

__7. Click on the '**Edit**' button next to one of the entries in the list and verify the form is displayed with the current item that was chosen to edit.


Current books (1)

ISBN Title Price Actions

167

Harry Potter

23.99

__8. Click the '**Cancel**' button and verify the application returns to the main list of books.

The 'Save' function does not work yet.

Part 5 - Implement Add/Edit Function

The REST service also accepts a PUT request to add or edit book entries. In this section you will implement the code needed to use this feature.

__1. Return to the '\src\app\services\data.service.ts' file in the text editor, opening it if required.

__2. At the top of the service implementation, add the following fields for 'headers' and 'options' objects. These will be used with sending the PUT request.

```
export class DataService {
  private headers = new Headers({ 'Content-Type': 'application/json',
      'charset': 'UTF-8' });
  private options = new RequestOptions({ headers: this.headers });

  constructor(private http: Http) { }
```

__3. Find the '**addEditBook**' method.

__4. Modify the implementation of this method as shown in bold below. Be careful as there are many parameters to the 'put' method that wrap between lines below. This method will construct a PUT request with the appropriate request properties and using the existing methods for mapping the response and errors.

```
  addEditBook(book: Book): Observable<any> {
    return this.http.put('/books/${book.isbn}', JSON.stringify(book),
        this.options).map(this.mapResponseStatus).catch(this.onError);
  }
```

__5. Save the file in the text editor.

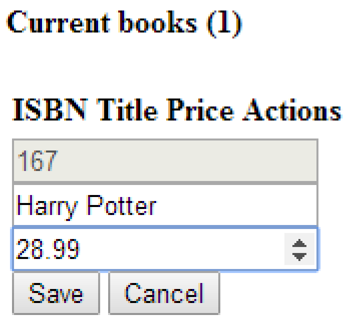
__6. Return to the command prompt running the project and check that the files are compiled without any errors.

__7. Return to the browser and refresh the main URL of the application. It should still display the same list of books.

```
http://localhost:3000/
```

__8. Click on the '**Edit**' button next to one of the entries in the list and verify the form is displayed with the current item that was chosen to edit.

__9. Modify the data in some way that will be easy to observe if the data has been changed.



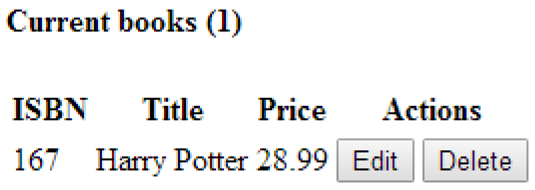
Current books (1)

ISBN Title Price Actions

167
Harry Potter
28.99

__10. Click the '**Save**' button to submit the modified data.

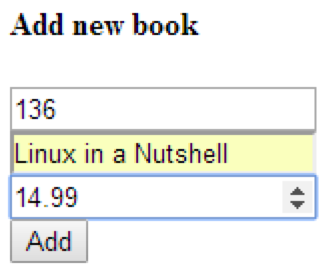
__11. Verify that the data for the item you modified is indeed changed in the list.



Current books (1)

ISBN	Title	Price	Actions
167	Harry Potter	28.99	Edit Delete

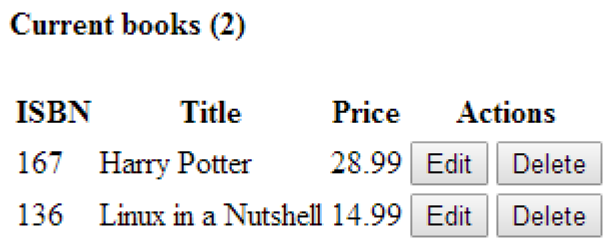
__12. Fill in the 'Add new book' form with some sample data. Make sure the ISBN number is different than any currently in the list.



Add new book

136
Linux in a Nutshell
14.99

__13. Click the '**Add**' button and make sure the new entry shows up in the list.



Current books (2)

ISBN	Title	Price	Actions
167	Harry Potter	28.99	Edit Delete
136	Linux in a Nutshell	14.99	Edit Delete

__14. In the command prompt running the project, press 'CTRL-C' to stop the project.

__15. Close all open text editors and browser windows.

Part 6 - Review

In this lab you implemented some methods of a custom service that used the Angular HTTP client service. This was used to communicate with a REST web service and send and retrieve data used by the Angular application. The properties of the requests that are sent, and the responses returned, are dependent on the REST web service.