Building REST APIs Using MicroProfile

REST stands for representational state transfer and is a software architecture style for creating web services. The primary used HTTP verbs are GET, POST, PUT, PATCH and DELETE.

GET	Get a list of resources or a single resources
POST	Create a new resource
PUT	Update/Replace an existing resource
PATCH	Update/Modify an existing resource
DELETE	Delete an existing resource

Now how should we define URIs for our book store application:

GET	http://localhost:8080/restapi/books	Return a list of all Books
GET	http://localhost:8080/restapi/books/1	Return the Book whose ID is 1
POST	http://localhost:8080/restapi/books	Create a new Book resource
PUT	http://localhost:8080/restapi/books/1	Update the Book whose ID is 1
DELETE	http://localhost:8080/restapi/books/1	Delete the Book whose ID is 1

Now that we have defined the book store URIs, it's time to start coding. Create a new file called `BookStoreEndpoint.java` inside `com.kodnito.bookstore.rest`. We start creating the GET methods, open the `BookStoreEndpoint.java` file and add the following:

```
package com.kodnito.bookstore.rest;

import com.kodnito.bookstore.entity.Book;
import com.kodnito.bookstore.service.BookService;
import javax.enterprise.context.RequestScoped;
import javax.inject.Inject;
import javax.ws.rs.Consumes;
import javax.ws.rs.GET;
import javax.ws.rs.POST;
import javax.ws.rs.Path;
import javax.ws.rs.Produces;
import javax.ws.rs.core.MediaType;
import javax.ws.rs.core.Response;


@RequestScoped
@Path("books")
@Produces(MediaType.APPLICATION_JSON)
@Consumes(MediaType.APPLICATION_JSON)
public class BookStoreEndpoint {

    @Inject
    BookService bookService;

    @GET
    public Response getAll() {
        return Response.ok(bookService.getAll()).build();
    }

    @GET
    @Path("{id}")
    public Response getBook(@PathParam("id") Long id) {
        Book book = bookService.findById(id);

        return Response.ok(book).build();
    }
}
```

We start with the annotations we have added, the `@RequestScoped` annotation indicates that this class will be created once every request. `@Path` annotation identifies the URI path to which the resource responds. `@Produces` annotation will automatically convert the response to JSON format and `@Consumes` annotation will automatically convert the posted JSON string here to Book object. We inject the BookService with the `@Inject` annotation. We annotated the getAll method with `@GET` annotation, which maps /books HTTP GET request to the method and will retrieve all the books from the database and return the entire list. Parameters are accessed with the `@PathParam` annotation.

Next, we will create the POST method, add the following to the BookStoreEndpoint.java:

```
@POST
public Response create(Book book) {
    bookService.create(book);
    return Response.ok().build();
}
```

The create method is annotated with the `@POST` annotation, which indicates that HTTP POST request are mapped to this method. Now that we have GET and POST methods done we can test that our application works. Open the terminal and navigate to the project directory and type the following command to start our application.

```
mvn clean package payara-micro:start
```

We start with the GET method.

```
curl -i -H "Accept: application/json" -H "Content-Type: application/json" -X GET
http://localhost:8080/restapi/books
```

Because we don't have any objects in the database, we will only get an empty list.

Output

```
[]%
```

Time to create Book objects in the database.

```
curl -i -X POST -H "Content-Type:application/json" \
  --data '{"title":"This is my test book","description":"this is my book description",
            "isbn": "12xxxxxxxx", "publisher": "None Yet",
"language":"English","author":"Hayri Cicek",
             "price": "0.00","pages":"0"}' http://localhost:8080/restapi/books
```

Output

```
HTTP/1.1 200 OK
Server: Payara Micro #badassfish
Content-Length: 0
X-Frame-Options: SAMEORIGIN
```

Now that we have created one book object, we can go back and try the GET method again to see that we get the book object from the database.

```
curl -i -H "Accept: application/json" -H "Content-Type: application/json" -X GET
http://localhost:8080/restapi/books
```

Output

```
HTTP/1.1 200 OK
Server: Payara Micro #badassfish
Content-Type: application/json
Content-Length: 171
X-Frame-Options: SAMEORIGIN

[{"description":"this is my book
description","id":1,"isbn":"12xxxxxxxx","language":"English","pages":0,"price":0.0,"pu
blisher":"None Yet","title":"This is my test book"}]%
```

Now, we have a list with one object returned to us from the database. Create another one and try to get a single object back.

```
curl -i -X POST -H "Content-Type:application/json" \
  --data '{"title":"This is my second test book","description":"this is my second book
description",
            "isbn": "13xxxxxxxx", "publisher": "None Yet",
"language":"English","author":"Hayri Cicek",
             "price": "0.00","pages":"0"}' http://localhost:8080/restapi/books
```

```
curl -i -H "Accept: application/json" -H "Content-Type: application/json" -X GET
http://localhost:8080/restapi/books/2
```

Output

```
HTTP/1.1 200 OK
Server: Payara Micro #badassfish
Content-Type: application/json
Content-Length: 183
X-Frame-Options: SAMEORIGIN

{"description":"this is my second book
description","id":2,"isbn":"13xxxxxxxx","language":"English","pages":0,"price":0.0,"pu
blisher":"None Yet","title":"This is my second test book"}%
```

The GET and POST methods seems to work and it's time to create the rest of the methods, PUT and DELETE. Open BookStoreEndpoint and add the following for updating an existing object.

```
@PUT
@Path("{id}")
public Response update(@PathParam("id") Long id, Book book) {
    Book updateBook = bookService.findById(id);

    updateBook.setIsbn(book.getIsbn());
    updateBook.setDescription(book.getDescription());
    updateBook.setLanguage(book.getLanguage());
    updateBook.setPages(book.getPages());
    updateBook.setPrice(book.getPrice());
    updateBook.setPublisher(book.getPublisher());
    updateBook.setTitle(book.getTitle());

    bookService.update(updateBook);

    return Response.ok().build();
}
```

Here we annotate the update method with `@PUT` annotation, which maps HTTP PUT verb request to this method and the method takes two parameters, id and Book object. Next is to add the Delete method to the API, open BookStoreEndpoint.java and add the following:

```
@DELETE
@Path("{id}")
public Response delete(@PathParam("id") Long id) {
    Book getBook = bookService.findById(id);
    bookService.delete(getBook);
    return Response.ok().build();
}
```

Here we annotate the delete method with `@DELETE` annotation, which maps HTTP DELETE verb request to this method. We pass an id to this method, which finds and deletes the Book objects whose id match. Now in the terminal, if you haven't quit the Payara Micro server, then quit by Ctrl+c and start again using the same mvn clean package payara-micro:start command.

Open another terminal window and try both the Update and Delete functions.

```
curl -H 'Content-Type: application/json' -X PUT \
-d '{"title":"This is my second test book updated","description":"this is my second
book description updated",
            "isbn": "13xxxxxxxx", "publisher": "None Yet",
"language":"English","author":"Hayri Cicek",
             "price": "1.00","pages":"0"}' http://localhost:8080/restapi/books/2
```

```
curl -i -H "Accept: application/json" -H "Content-Type: application/json" -X GET
http://localhost:8080/restapi/books/2
```

Output

```
HTTP/1.1 200 OK
Server: Payara Micro #badassfish
Content-Type: application/json
Content-Length: 199
X-Frame-Options: SAMEORIGIN

{"description":"this is my second book description
updated","id":2,"isbn":"13xxxxxxxx","language":"English","pages":0,"price":1.0,"publis
her":"None Yet","title":"This is my second test book updated"}%
```

Here, I will update the Book with id 2 and if you don't have Book object with id 2 then take one that you have in your database, now if you get all objects again you will see that the object is updated. Next is to try the DELETE method, open a new terminal tab and use the command below.

```
curl -X DELETE http://localhost:8080/restapi/books/2
```

Now when you get the book list again, the book object is deleted.

```
curl -i -H "Accept: application/json" -H "Content-Type: application/json" -X GET
http://localhost:8080/restapi/books
```

Output

```
HTTP/1.1 200 OK
Server: Payara Micro #badassfish
Content-Type: application/json
Content-Length: 171
X-Frame-Options: SAMEORIGIN

[{"description":"this is my book
description","id":1,"isbn":"12xxxxxxxx","language":"English","pages":0,"price":0.0,"pu
blisher":"None Yet","title":"This is my test book"}]%
```

Summary

In this chapter we learned how to create a REST API using MicroProfile and curl to test our API.