Documentation of Shopify Developer Intern Challenge Project

Wei Wei

0. Before the introduction

I am an international CS student from University of Ottawa, last term our professor asked all the students in CSI 5380 to form a team of 4 people to build a online bookstore website, which have functions like user register and login, querying for all products, purchase products.

At that time I was responsible for **database design and the development of Service layer and DAO layer.** Instead of using API, I used normal Java class as Service, which resulted in a successful project:
https://github.com/SevenWG/BookStore_Parto.git

Then I used Jersey to rebuild the Service layer in that project, and also I uploaded it to Github:
https://github.com/SevenWG/BookStoreAPI.git

After I saw the Summer Intern developer challenge, I found that the only difference is that there is no “inventory count” in my BOOK table, so I altered my database table and modified some functions in the Second project to meet the requirements in Intern challenge.
I also uploaded my project for Intern challenge to Github:
https://github.com/SevenWG/InternChallengeQuestion.git
(The database file is in the Root directory)

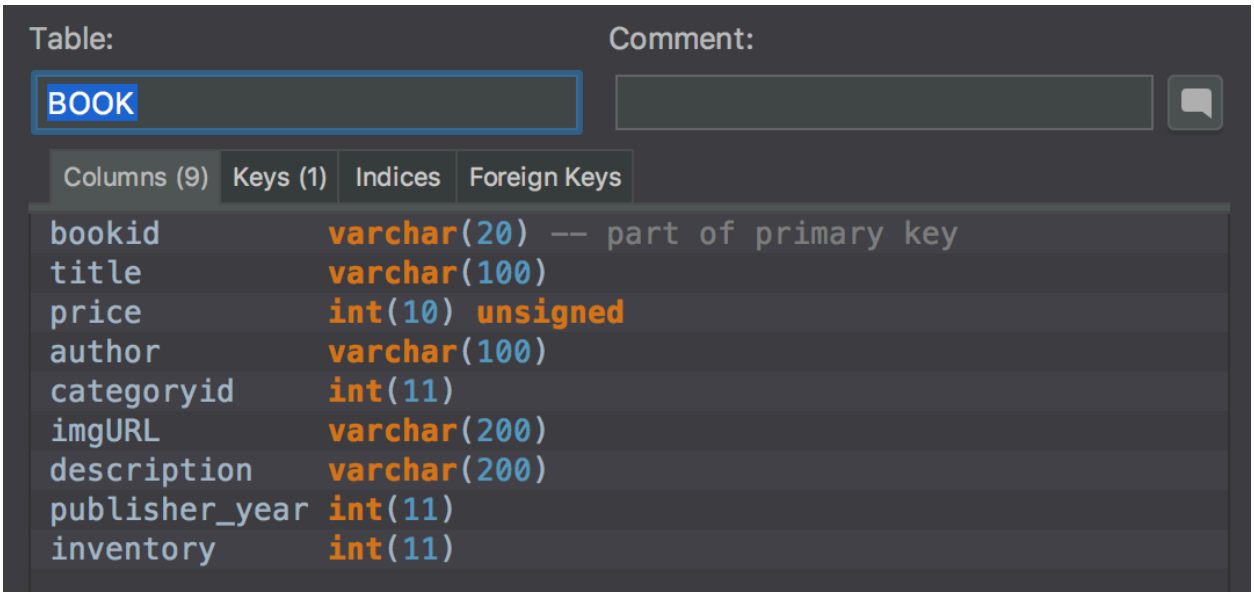
I didn’t delete other functions and files which are not related to this challenge just because I want to show you the project I developed before. Sorry about the inconvenience .

1. Development Environment

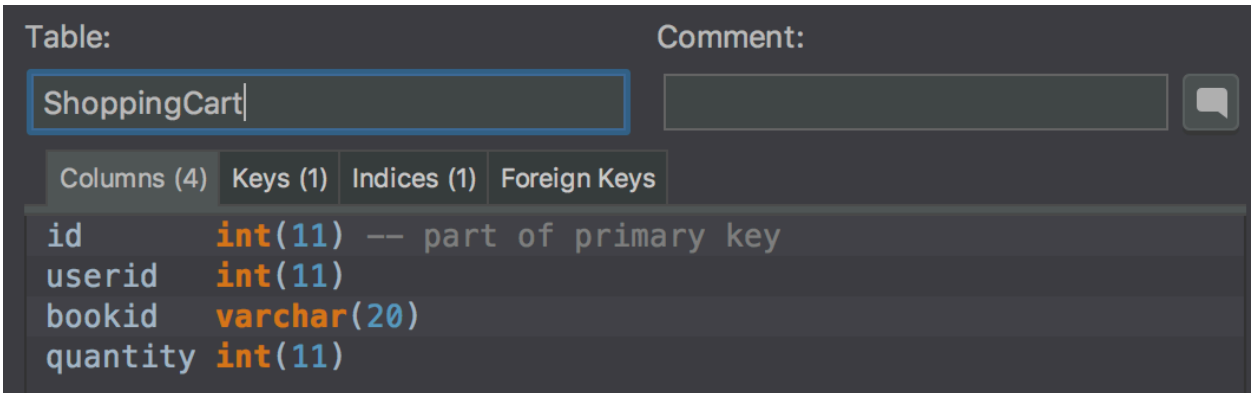
System: Mac OS 10.13.6
IDE: IntelliJ IDEA 2018.2.4 (Ultimate Edition)
Database: MySQL 8.0.12
Programming Language : Java 10 + Hibernate + REST API + Maven

2. Database Design

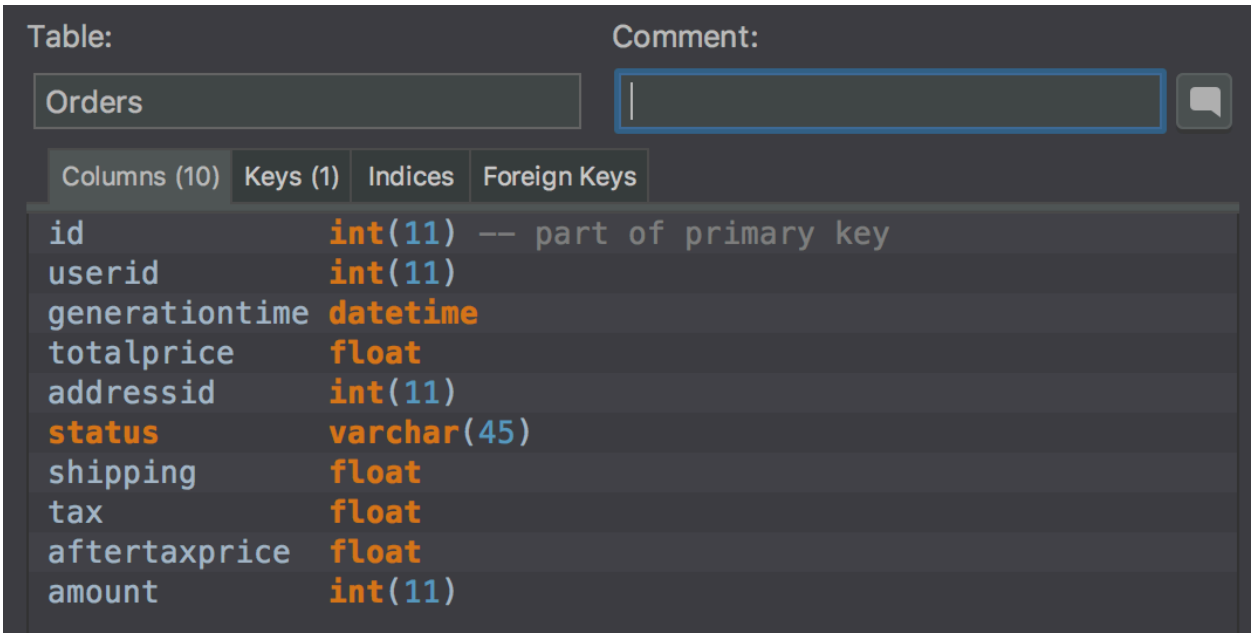
Product Table: BOOK
Primary key: bookid
title: title
price: price
inventory_count: inventory



Shopping cart Table: ShoppingCart



Orders Table: Orders



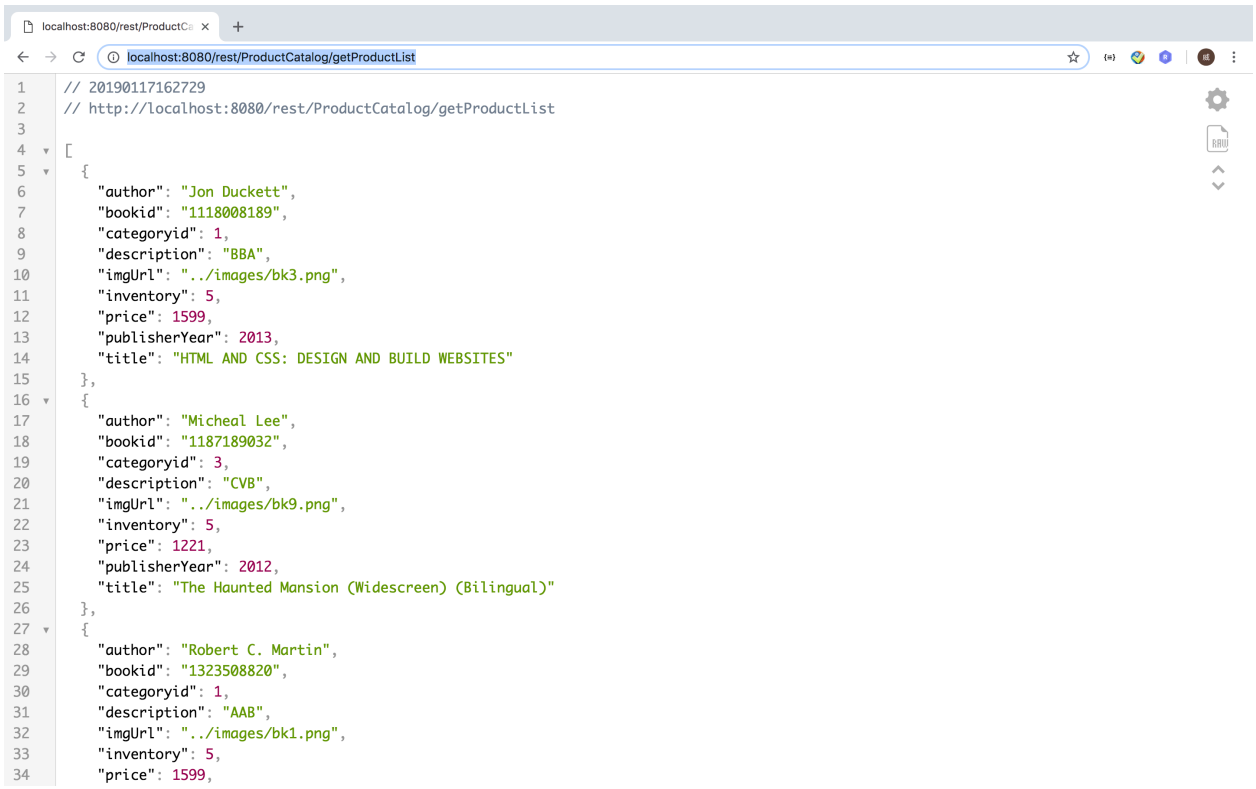
(This part my design is a little different with the requirement. I will explain in the following parts)

3. Methods Design

① Querying for all products:

Set up and run the server

input "**http://localhost:8080/rest/ProductCatalog/getProductList**" in the browser, and results will be output by JSON format:



Or you can input "**http://localhost:8080/rest/ProductCatalog/getProductList**" **+ specific category id**, then it will only output products in that category

Note that Only those products whose inventory value is more than 0 will be output.

Source code:
Service layer:
com.team404.bookstore.service.ProductCatalogAPI.java

```java
@GET
@Path("/getProductList")
@Produces(MediaType.APPLICATION_JSON)
/*gets the list of products*/
/*
 * Implementation of Factory Pattern
 * */
public Response getProductList() {

    List<BookEntity> list = null;

    list = (List<BookEntity>)daoFactory.ListSomethingById( classname: "BookDao",
            methodName: "getListById", id: 0);

    return Response.status(Response.Status.OK).entity(jsonb.toJson(list)).build();
}
```

```java
/*gets the list of products for a specific category*/
/*
 * Implementation of Factory Pattern
 * */
/*
When the size of list is 0(Wrong Category or no book in this catefory)
return HTTP 400 + wrong info message
otherwise, return 200 + list
* */
@GET
@Path("/getProductList/{categoryid}")
@Produces(MediaType.APPLICATION_JSON)
public Response getProductList(@PathParam("categoryid") int categoryid) {

    List<BookEntity> list = null;

    list = (List<BookEntity>)daoFactory.
            ListSomethingById( classname: "BookDao", methodName: "getListById", categoryid);

    if(list.size() == 0) {
        String erroMessage = "Wrong Category or No book in this Category!";
        return Response.status(Response.Status.BAD_REQUEST).entity(jsonb.toJson(erroMessage)).build();
    }
    else {
        return Response.status(Response.Status.OK).entity(jsonb.toJson(list)).build();
    }
}
```

DAO layer:
com.team404.bookstore.dao.BookDao.java

```
    public List<BookEntity> getListById(int categoryid) {
        List<BookEntity> list = null;
        Session session = sessionFactory.openSession();

        Transaction transaction = null;
        try {
            if(categoryid != 0)
            {
                transaction = session.beginTransaction();
                Query query = session.getNamedQuery( s: "ListBookByCidQuery");
                query.setParameter( s: "categoryid", categoryid);
                list = query.list();
                transaction.commit();
            }
            else {
                transaction = session.beginTransaction();
                list = session.getNamedQuery( s: "ListBookQuery").list();
                transaction.commit();
            }
        } catch (HibernateException e) {
            if (transaction != null) transaction.rollback();
            e.printStackTrace();
        } finally {
            session.close();
        }
        return list;
    }
```

HQL query:
com.team404.bookstore.entity.BookEntity.hbm.xml

```
<query name="ListBookQuery">FROM BookEntity WHERE inventory > 0</query>
<query name="ListBookByCidQuery">FROM BookEntity WHERE categoryid = :categoryid AND inventory > 0</query>
```

② Create shopping cart and add items into it

Because in this project shopping cart is also a data table in database, it also has Hibernate entity class and mapping file.

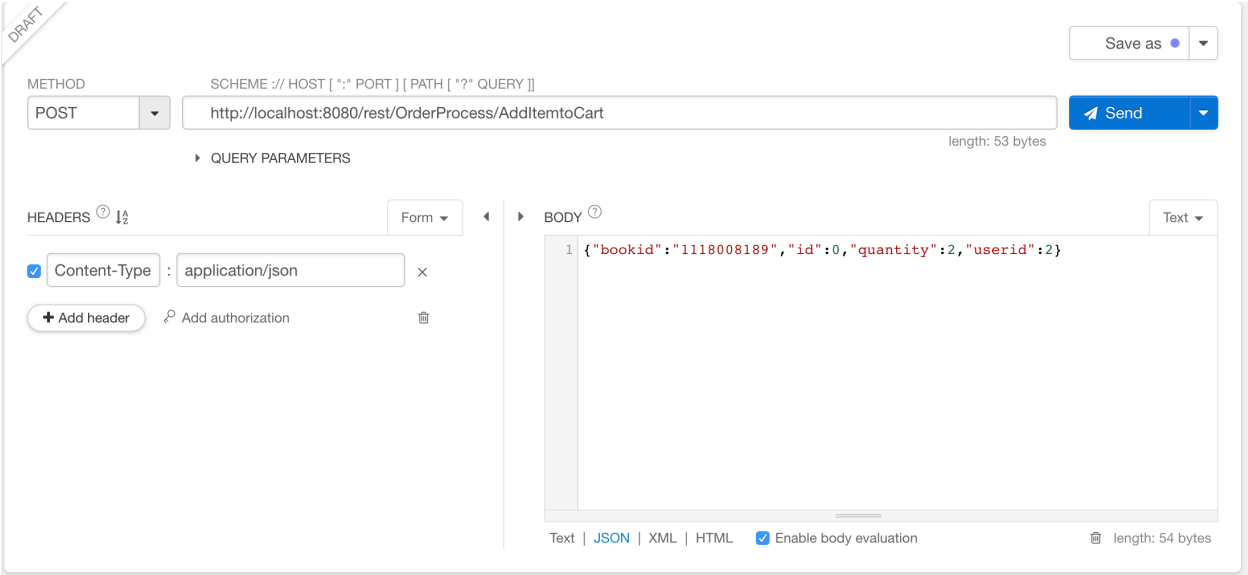
In service layer, the method receive a String by POST, which is a shopping cart object in JSON format, then the method will transform it back to a shopping cart object, then use the DAO method to store it in database. Because we don't have a front end. Firstly we need initiate a shopping cart entity(object), then transform it into JSON string, then we use **Restlet Client** to do the POST action.

Create JSON string: com.team404.bookstore.service.ServiceTest

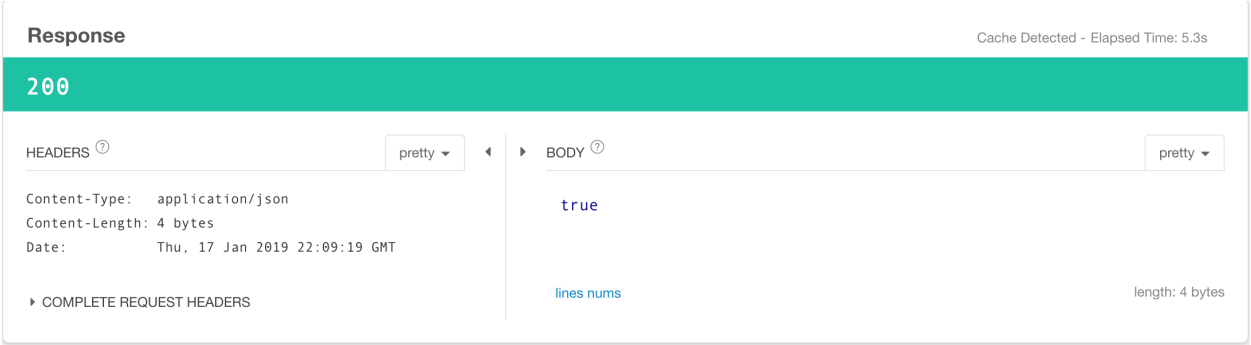
```
public class ServiceTest {

    private static Jsonb jsonb = JsonbBuilder.create();
    public static void main(String args[]) throws Exception {

        ShoppingCartEntity shoppingCartEntity = new ShoppingCartEntity();
        shoppingCartEntity.setBookid("1118008189");
        shoppingCartEntity.setQuantity(2);
        shoppingCartEntity.setUserid(2);

        String json = jsonb.toJson(shoppingCartEntity);
        System.out.println(json);

    }
}
```

```
/Library/Java/JavaVirtualMachines/jdk-10.0.2.jdk/Contents/Home/bin/java ...
{"bookid":"1118008189","id":0,"quantity":2,"userid":2}

Process finished with exit code 0
```

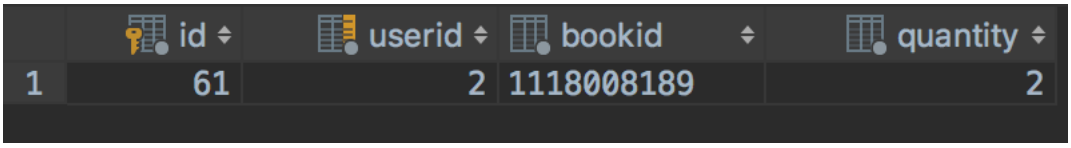
Then we paste it into Restlet Client and input url:
"http://localhost:8080/rest/OrderProcess/AddItemtoCart"



After clicking “Send”, the Response will be shown below:



And the shopping cart object will be added into ShoppingCart table:



	id	userid	bookid	quantity
1	61	2	1118008189	2

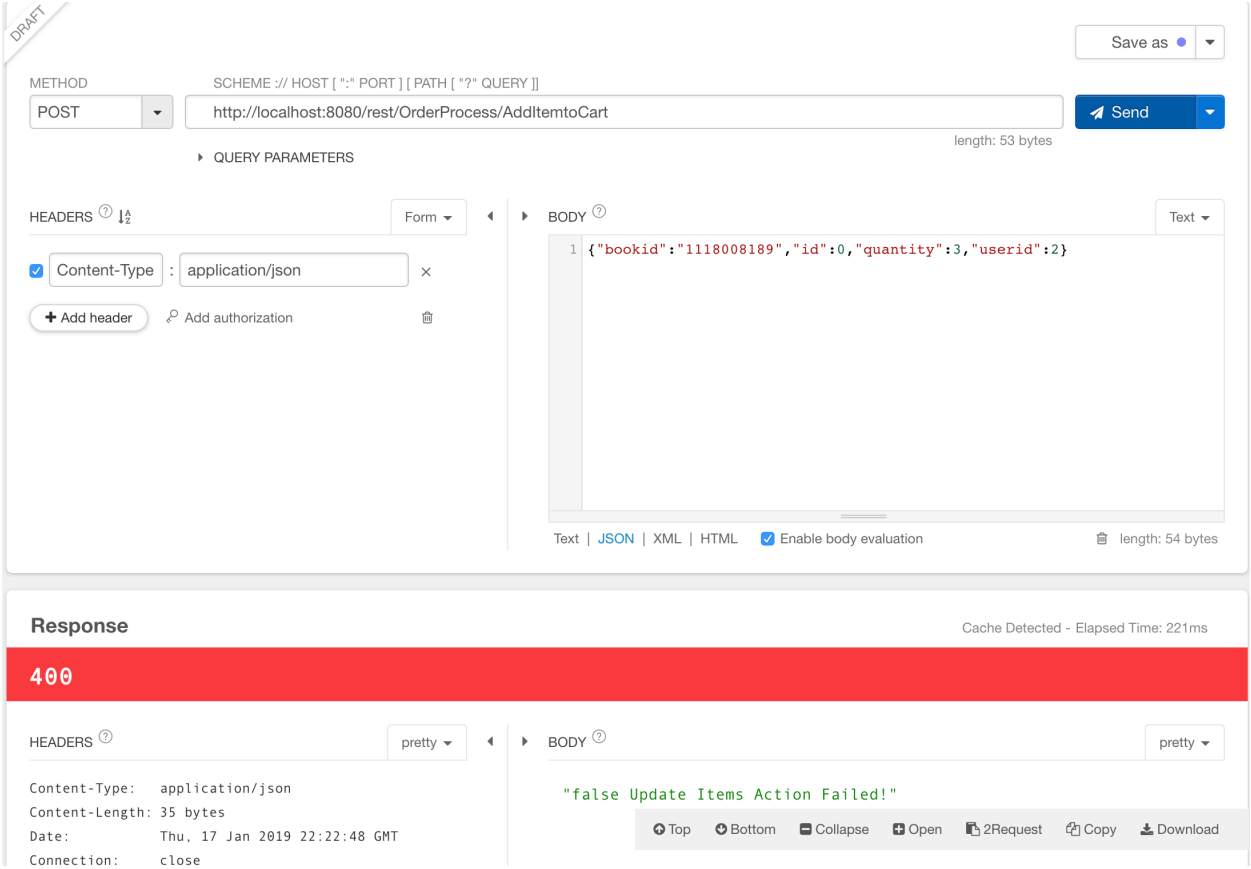
If the same user add same book again before he check out (assume this time he add 1 book with same Book id, so the json string is {“bookid":"1118008189","id":0,"quantity":1,"userid":2})

Then it will simply update the previous row instead add a new row.

	id	userid	bookid	quantity
1	61	2	1118008189	3

If the total number of books a user wants to buy is greater than the inventory, then the add or update action will fail.

Assume now there is 3 book in this user's shopping cart, if he still wants to buy 3 more books with the same book id but the inventory count is 5, then this action will fail:



Source Code:

Service layer: com.team404.bookstore.service.OrderProcessAPI.java

Method: public Response AddItemtoCart(String json)

```
@POST
@Path("/AddItemtoCart")
@Consumes(MediaType.APPLICATION_JSON)
@Produces(MediaType.APPLICATION_JSON)
public Response AddItemtoCart(String json) {

    shoppingCartDao = new ShoppingCartDao();

    Jsonb jsonb = JsonbBuilder.create();
    ShoppingCartEntity shoppingCartEntity = jsonb.fromJson(json, ShoppingCartEntity.class);

    if(shoppingCartDao.GetCartItem(shoppingCartEntity.getUserid(),
            shoppingCartEntity.getBookid()) == null) {
        boolean flag = shoppingCartDao.AddShoppingCart(shoppingCartEntity);
        if(!flag) {
            String errorMessage = "Add Items Action Failed!";
            return Response.status(Response.Status.BAD_REQUEST).entity(jsonb.toJson( o: flag + " " + errorMessage)).build();
        }
        else {
            return Response.status(Response.Status.OK).entity(jsonb.toJson(flag)).build();
        }
    }

    else {
        boolean flag = shoppingCartDao.UpdateItemQuantity(shoppingCartEntity);
        if(!flag) {
            String errorMessage = "Update Items Action Failed!";
            return Response.status(Response.Status.BAD_REQUEST).entity(jsonb.toJson( o: flag + " " +errorMessage)).build();
        }
        else {
            return Response.status(Response.Status.OK).entity(jsonb.toJson(flag)).build();
        }
    }
}
```

Dao layer: com.team404.bookstore.dao.ShoppingCartDao.java

Method: public boolean AddShoppingCart

```
public boolean AddShoppingCart(ShoppingCartEntity shoppingCartEntity) {
    boolean flag = true;
    Session session = sessionFactory.openSession();
    Transaction transaction = null;

    /*Check whether the book's inventory is less than customer's requirement quantities
    * if it is, then return false to service layer
    * else, add this shoppingCartEntity
    * */
    BookDao bookDao = new BookDao();
    BookEntity bookEntity = bookDao.getEntityById(Integer.parseInt(shoppingCartEntity.getBookid()));

    if(bookEntity.getInventory() < shoppingCartEntity.getQuantity()) {
        flag = false;
    }
    else {
        try {
            transaction = session.beginTransaction();
            session.save(shoppingCartEntity);
            transaction.commit();
        } catch (HibernateException e) {
            if (transaction != null) transaction.rollback();
            e.printStackTrace();
            flag = false;
        } finally {
            session.close();
        }
    }
    return  flag;
}
```

Method: public boolean UpdateItemQuantity

```
public boolean UpdateItemQuantity(ShoppingCartEntity shoppingCartEntity) {
    Session session = sessionFactory.openSession();
    boolean flag = true;
    Transaction transaction = null;

    ShoppingCartEntity shoppingCartEntity1 = GetCartItem(shoppingCartEntity.getUserid(), shoppingCartEntity.getBookid());
    int previouQuantity = shoppingCartEntity1.getQuantity();

    int totalQuantity = shoppingCartEntity.getQuantity() + previouQuantity;
    BookDao bookDao = new BookDao();
    BookEntity bookEntity = bookDao.getEntityById(Integer.parseInt(shoppingCartEntity.getBookid()));
    /*Check whether the book's inventory is less than customer's requirement quantities
     * if it is, then return false to service layer
     * else, update this shoppingCartEntity
     * */
    if(bookEntity.getInventory() < totalQuantity) {
        flag = false;
    }
    else {
        try {
            transaction = session.beginTransaction();
            Query query = session.getNamedQuery( s: "UpdateItemQuantityQuery");
            query.setParameter( s: "quantity", totalQuantity);
            query.setParameter( s: "id", shoppingCartEntity1.getId());
            int result = query.executeUpdate();
            System.out.println("Rows affected: " + result);
            transaction.commit();
        } catch (HibernateException e) {
            if (transaction != null) transaction.rollback();
            e.printStackTrace();
            flag = false;
        }finally {
            session.close();
        }
    }
    return  flag;
}
```

③Check out
Check out is divided into 2 parts: Create Order and Confirm Order

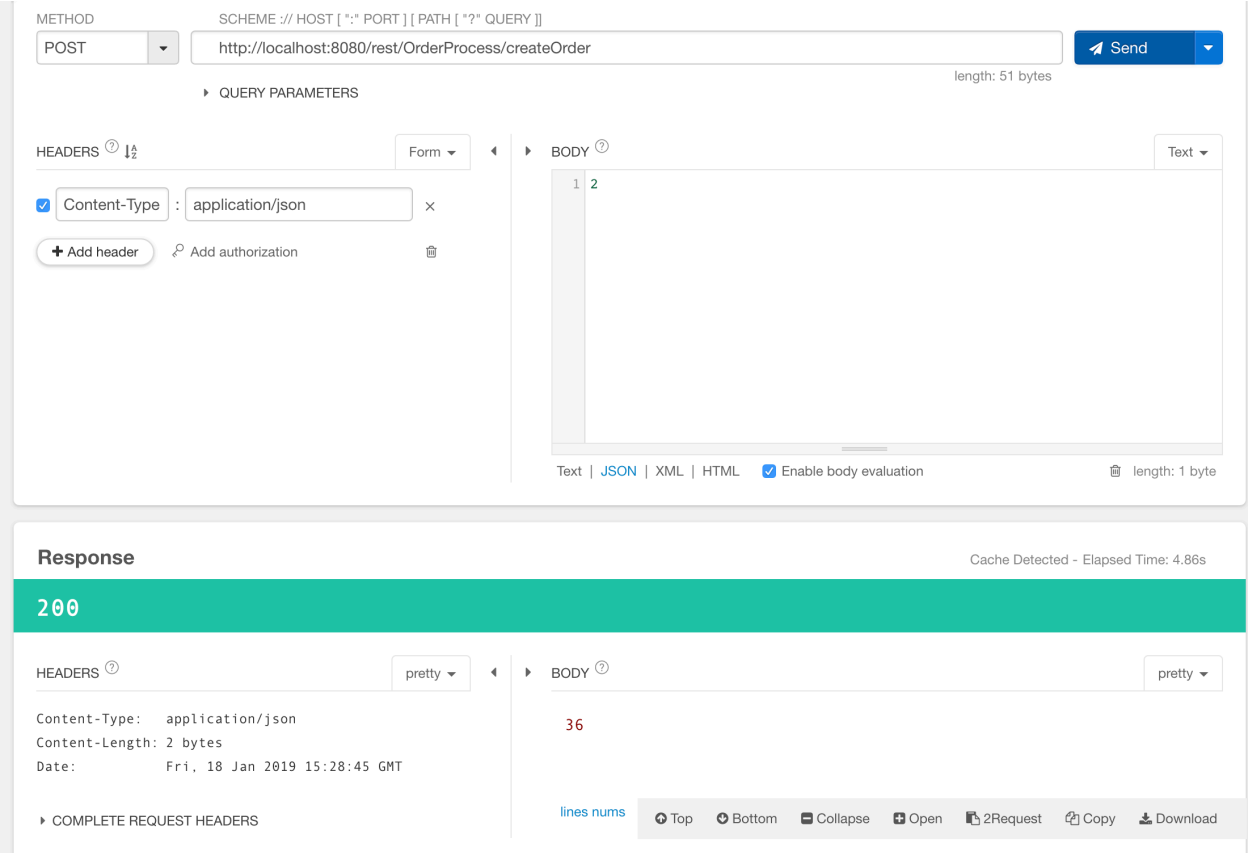
In Create Order part, the createOrder method in service layer will receive an userid by POST, then it will generate a order object according to user id and all the rows in ShoppingCart Table which have the same user id.

The order object contains total amount of products, total price (before and after tax), generation time and some other values. The total price, total amount of products and other values are generated by different specific methods.

Then a method in DAO layer will store this object into database and return this order's id(primary key) to Service layer.

Go back to Restlet Client, and input the url: **"http://localhost:8080/rest/OrderProcess/createOrder"** and the userid.

After clicking “Send”, then it will get the response, which is the order’s id



In Orders Table, we can see that this order object has been added into the table. But now it status is still “Processing”, and the book’s inventory count in BOOK table has not been changed yet.

Orders Table:

	id	userid	generationtime	totalprice	addressid	status	shipping	tax	aftertaxprice	amount
1	36	2	2019-01-18 15:28:45	4797	2	Processing	8	383.76	5188.76	3

BOOK Table:

	bookid	title	price	author	categoryid	imgURL	...	p...	inventory
1	1118008189	HTML AND CSS: DE…	1599	Jon Duckett	1	../images/bk3.png	BBA	2013	5

Source Code
Service layer: com.team404.bookstore.service.OrderProcessAPI
Method: public Response createOrder(String json)

```java
@POST
@Path("/createOrder")
@Consumes(MediaType.APPLICATION_JSON)
@Produces(MediaType.APPLICATION_JSON)
public Response createOrder(String json) {

    Jsonb jsonb = JsonbBuilder.create();
    int userid = jsonb.fromJson(json, int.class);

    orderServiceFacade = new OrderServiceFacade();

    int id = orderServiceFacade.OrderGnerator(userid);

    if(id != 0)
        return Response.status(Response.Status.OK).entity(jsonb.toJson(id)).build();
    else{
        String errorMessage = "Create Order Failed!";
        return Response.status(Response.Status.BAD_REQUEST).entity(jsonb.toJson(errorMessage)).build();
    }
}
```

Service layer: com.team404.bookstore.service.PriceCalculator
Method: public float CalculateTotalPrice

```java
public float CalculateTotalPrice(List<ShoppingCartEntity> list) {
    bookDao = new BookDao();
    float totalPrice = 0;

    for(ShoppingCartEntity i : list) {
        BookEntity bookEntity = bookDao.getEntityById(Integer.valueOf(i.getBookid()));
        totalPrice += bookEntity.getPrice()*i.getQuantity();
    }
    return totalPrice;
}
```

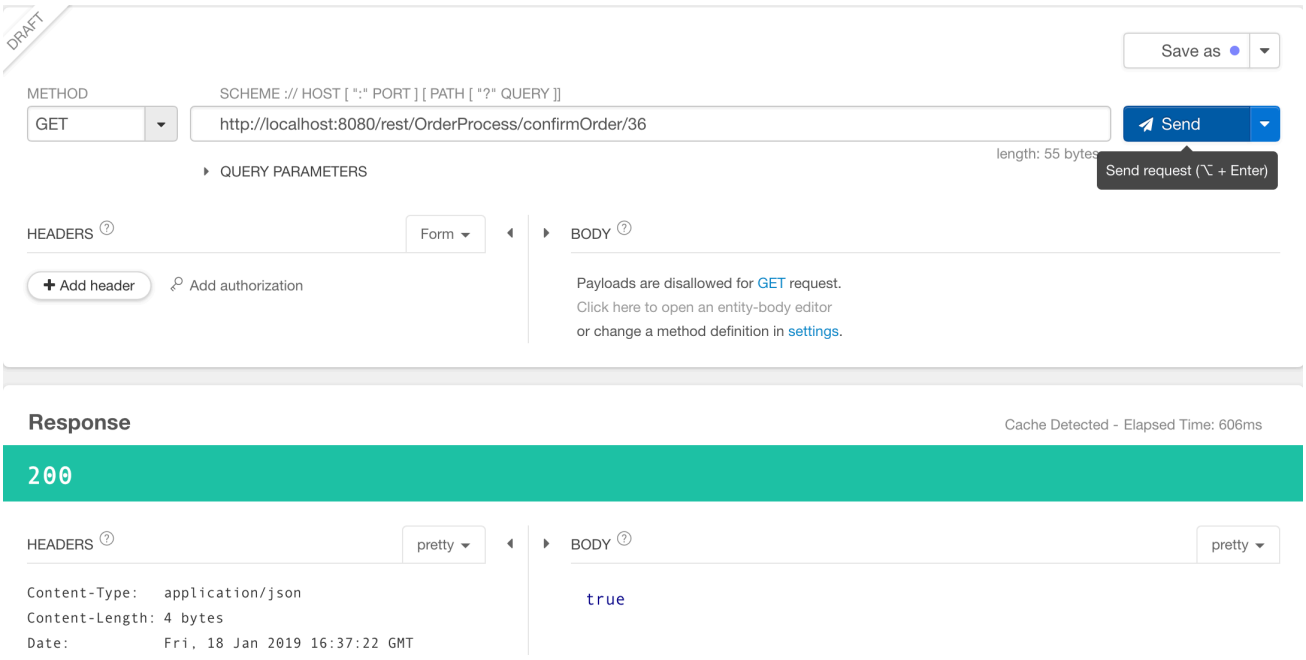
DAO layer: com.team404.bookstore.dao.OrderDao
Method: public int AddOrder

```
public int AddOrder (OrdersEntity orderEntity) {
    Session session = HibernateConnection.getSession();
    int id = 0;
    Transaction transaction = null;

    try {

        transaction = session.beginTransaction();
        int id1 = (Integer) session.save(orderEntity);
        id = orderEntity.getId();
        transaction.commit();

    } catch (HibernateException e) {
        if (transaction != null) transaction.rollback();
        e.printStackTrace();
    }finally {
        session.close();
    }
    return id;
}
```

In Confirm Order Part, the confirmOrder method in service layer will receive a order id by GET, then it will call a method in DAO lay to update the order's status from "Processing" to "Success" or "Failed".

After the update action, if the order's status is Success, then another method in DAO layer will update(reduce) the inventory count of the book(s) in this orders.

Go back to Restlet Client, select “GET” method and input the url: **“http://localhost:8080/rest/OrderProcess/confirmOrder/” + Order id**, then cli



ck “Send”, the response will be shown below:

Check the Orders Table and BOOK Table, the order status and inventory have been updated:

	... 1	userid	generationtime	totalprice	addressid	status	shipping	tax	aftertaxprice	amount
1	36	2	2019-01-18 15:28:45	4797	2	Success	8	383.76	5188.76	3

	bookid	title	price	author	categoryid	imgURL	..	...	inventory
1	1118008189	HTML AND CSS: DESIGN AND BUIL…	1599	Jon Duckett	1	../images/bk3.png	BBA	2013	2

4. Unit Test:
Test Tools: **Junit 4 + Rest-Assured**
Test Files:
/src/resources/java/ProductCatalogAPITest.java

```java
public class ProductCatalogAPITest {
    private static Jsonb jsonb = JsonbBuilder.create();


    /**...*/
    @Test
    public void testGetProductList() throws Exception {
        Response response = RestAssured.get( path: "http://localhost:8080/rest/ProductCatalog/getProductList");
        int code = response.getStatusCode();
        System.out.println("Status Code: " +code);
        Assert.assertEquals(code,  actual: 200);
        String data = response.asString();
        System.out.println(data);
        List<BookEntity> bookEntityList = jsonb.fromJson(data,
                new ArrayList<BookEntity>(){}.getClass().getGenericSuperclass());
        for(BookEntity i : bookEntityList) {
            System.out.println(i.toString());
        }
    }

    /**...*/
    @Test
    public void testGetProductListCategoryid() throws Exception {
        /*Change the id whatever you want*/
        String id = "1";
        Response response = RestAssured.get( path: "http://localhost:8080/rest/ProductCatalog/getProductList/"  + id);
        int code = response.getStatusCode();
        System.out.println("Status Code: " +code);
        Assert.assertEquals(code,  actual: 200);
        String data = response.asString();
        System.out.println(data);
        List<BookEntity> bookEntityList = jsonb.fromJson(data,
                new ArrayList<BookEntity>(){}.getClass().getGenericSuperclass());
        for(BookEntity i : bookEntityList) {
            System.out.println(i.toString());
        }
    }

    @Test
    public void testGetProductListwithWrongCatagoryid() throws Exception {
        String woringId = "100";
        Response response = RestAssured.get( path: "http://localhost:8080/rest/ProductCatalog/getProductList/"  + woringId);
        int code = response.getStatusCode();
        System.out.println("Status Code: " +code);
        String data = response.asString();
        System.out.println(data);

    }
```

/src/resources/java/OrderProcessAPITest.java

```java
public class OrderProcessAPITest {

    private static Jsonb jsonb = JsonbBuilder.create();
    /**...*/
    @Test
    public void testAddItemtoCart() throws Exception {
        ShoppingCartEntity shoppingCartEntity = new ShoppingCartEntity();
        shoppingCartEntity.setBookid("1323508820");
        shoppingCartEntity.setQuantity(5);
        shoppingCartEntity.setUserid(21);

        String json = jsonb.toJson(shoppingCartEntity);

        RequestSpecification request = RestAssured.given();

        request.header( s: "Content-Type",  o: "application/json");
        request.body(json);

        Response response = request.post( s: "http://localhost:8080/rest/OrderProcess/AddItemtoCart");

        int code = response.getStatusCode();
        System.out.println("Status Code: " + code);

        Assert.assertEquals(code,  actual: 200);

        String data = response.asString();

        System.out.println(data);
    }

    @Test
    public void testAddTooManyItemtoCart() throws Exception {
        ShoppingCartEntity shoppingCartEntity = new ShoppingCartEntity();
        shoppingCartEntity.setBookid("1187189032");
        shoppingCartEntity.setQuantity(20000);
        shoppingCartEntity.setUserid(21);

        String json = jsonb.toJson(shoppingCartEntity);

        RequestSpecification request = RestAssured.given();

        request.header( s: "Content-Type",  o: "application/json");
        request.body(json);

        Response response = request.post( s: "http://localhost:8080/rest/OrderProcess/AddItemtoCart");

        int code = response.getStatusCode();
        System.out.println("Status Code: " + code);

        Assert.assertEquals(code,  actual: 400);

        String data = response.asString();
```