**BOOKSTORE – Project 4**

**DOCUMENTATION**

**By:**

**ASMA MEHJABEEN**

**DANIEL BEDNARCZYK**

**SOWMYA SAMA**

Table of Contents

[VERSION HISTORY: 3](#_Toc405912892)

[INTRODUCTION: 4](#_Toc405912893)

[FUNCTIONALITY: 4](#_Toc405912894)

[ARCHITECTURE: 4](#_Toc405912895)

[WEBSERVICE DESCRIPTION: 5](#_Toc405912896)

[DOMAIN CLASSES: 5](#_Toc405912897)

[Customer: 5](#_Toc405912898)

[Address: 6](#_Toc405912899)

[Book: 6](#_Toc405912900)

[Order: 6](#_Toc405912901)

[OrderDetail 6](#_Toc405912902)

[WORKFLOW ACTIVITY CLASSES: 6](#_Toc405912903)

[RESOURCE CLASS: 7](#_Toc405912904)

[REPRESENTATION CLASSES: 8](#_Toc405912905)

[USING THE WEBSERVICE: 11](#_Toc405912906)

[CONNECTING TO THE WEBSERVICE: 11](#_Toc405912907)

[USING THE RESOURCE CLASS: 11](#_Toc405912908)

## VERSION HISTORY:

|  |  |
| --- | --- |
| **Version Number** | **Description** |
| V1.0 | Initial bookstore project based on REST architecture without HATEOS. The client is a plain old java client |
| V1.0 | * Added support for HATEOS to make it a RESTful webservice. * Added authentication. * Added the client |

## INTRODUCTION:

Bookstore is an application system where in a customer or a client can search for books and buy books. The client launches the application. The application prompts the user to search for books. The application returns a list of books to the client. The client can click on each of those books in the list to view detailed information about the books. The application also lets clients to purchase books.

### FUNCTIONALITY:

* Search for a list of all books present in the inventory.
* Search a particular book by title, author or ISBN or all the above.
* Search for a particular book by Id.
* Customer authentication.
* Retrieve a customer by id.
* Place an order.
* View all orders of a particular customer.
* View the order status of a customer.
* Cancel the order.

## ARCHITECTURE:

The application follows the **client-server** architecture. The client sends in requests to the web-service and the web-service responds to the client by using the HTTP protocol.

**Bookstore Client**

(.NET)

0

Apache Tomcat Webserver

Apache CXF – JAX-RS

**Marshaller (Jackson)**

**Bookstore RESTful**

**Webservice (WAR)**

XML /HTTP

JSON /HTTP

GET

POST  
DELETE

***Figure 1:*** System Architecture

The webservice is implemented using the **RESTful Web-Service Architecture**. It follows Level 3 of the Richardson Maturity Model. Domain, workflow activities, resources, representation classes are written to build the service. They are explained in the document.

|  |
| --- |
| HTTP |
| Representation |
| Resource |
| Work Flow |
| Domain |

***Figure 2:*** Webservice Architecture.

## WEBSERVICE DESCRIPTION:

### DOMAIN CLASSES:

These are the underlying classes that represent the objects of the underlying domain. These are model classes that are created, read, updated or deleted by the workflow activity classes. In our project we have the following domain classes:

Customer:This class represents a customer of the application. A customer is anyone who launches the webservice to search for books, purchase them or use the webservice as a whole. This class has attributes:

* customerId,
* lastName,
* firstName,
* defaultBillingAddress (address object),
* defaultShippingAddress (address object),
* username,
* password,
* orders (list of order object)

All the attributes except when explicitly stated are of type string.

Address: This object is used for holding the address of a customer. This class has the following attributes:

* addressId,
* street,
* city,
* state,
* zip

All the attributes except when explicitly stated are of type string.

Book: This object represents a physical book. This class has the following attributes:

* id,
* title,
* price,
* isbn,
* author,
* shortDescription,
* detailedDescription

All the attributes except when explicitly stated are of type string.

Order:This object represents an order that can be placed by a customer. This class has the following attributes:

* orderId,
* orderDetails,
* orderState (enum representing the status eg: shipped, opened),
* paymentOption,
* billingAddress (address object),
* shippingAddress (address object)

All the attributes except when explicitly stated are of type string.

OrderDetail**:** This class contains information about a particular product in the order. This class has the following attributes:

* book(book object),
* quantity

### WORKFLOW ACTIVITY CLASSES:

These are units of work that are delegated to work with the domain objects on behalf of the Resource class. These classes execute some business interactions with the domain model.

**BookActivity:** With the following methods, primarily to search for books with difference in parameters accepted and the first gives a list of books where as the later has an output parameter which represents book with a particular id given as input parameter.

* **public** ArrayList<BookRepresentation> **searchBook**(String title, String author, String ISBN)
* **public** BookRepresentation **searchBookById**(String id)

**CustomerActivity:** With the following methods, primarily to get, add, and authenticate customer.

* **public** CustomerRepresentation **getCustomer**(String id)
* **public** CustomerRepresentation **authenticateCustomer**(String username, String password)
* **public** StringRepresentation **addCustomer**(String firstName, String lastName, Address billingAddress, Address shippingAddress)

**OrderActivity:** With the following methods to do perform different actions like place order, get a specific order, get all placed orders, get order status, delete a order placed by customer etc. Here get order and delete order of a customer will have same URI but different HTTP method at HTTP layer.

* **public** StringRepresentation **placeOrder**(OrderRequest orderRequest)
* **public** List<OrderRepresentation> **viewAllOrders**(String customerId)
* **public** OrderRepresentation **viewCompleteOrder**(String customerId, String orderId)
* **public** StringRepresentation **cancelOrder**(String customerId, String orderId)
* **public** StringRepresentation **getOrderStatus**(String customerId, String orderId)

### RESOURCE CLASS:

Resource class exposes the service implementation. A resource class is a Java class annotated with JAX-RS annotations to represent a Web resource. These are contollers for workflow activities.In our Project we have “**BookStoreResourceInterface”** which is an interface of our resource class. The following are the methods in the interface:

* Response getCustomer(String id);
* Response addCustomer(CustomerRequest customerRequest);
* Response searchBook(String title, String author, String ISBN);
* Response searchBookById(String id);
* Response placeOrder(OrderRequest orderRequest);
* Response getOrder(String customerId, String orderId, **boolean** statusRequired);
* Response cancelOrder(String customerId, String orderId);
* Response authenticateCustomer(String username, String password);

This interface has method declarations to add, get, authenticate customer, search book, place order, get order status, get order information or cancel orders.

**BookStoreResource** implements **BookStoreResourceInterface.**

### REPRESENTATION CLASSES:

It represents the underlying domain object, which will be sent over the wire after being converted to XML, JSON etc. It is not the actual domain object but just a representation and changes according to what fields from the domain object the service wishes to send. It is marked with JAXB annotation to support XML serialization and will contain the links in HATEOAS.

In our project we have an abstract class named **AbstractRepresentation** and all other representation classes are inherited as child classes to this abstract class. We have representation class for each domain class.

**Link** class is used to form object of links to for the representation. It has 4 attributes:

* use – Specifies what the link can be used for.
* url – The URI sent.
* method – The HTTP method that is used to invoke the service.
* mediaType – The content type method that is consumed by the webservice.

Example: The following represents link that get the customer details.

{"use":"View the customer's detailed information","url":"<http://ec2-54-148-67-53.us-west-2.compute.amazonaws.com/bookstoreservice/v1.0/customers/12345>","mediaType":"application/json","method":"GET"}

**CustomerRepresentation:** Represents the customer domain object. It has all fields that the domain object has but since it is marked with the Json serialization inclusion – NON\_EMTPY annotation, the only information that is sent is the information that the webservice sets in the representation object.

Example in JSON HATEOAS:

{"links":

[{"use":"View all orders","url":"<http://ec2-54-148-67-53.us-west-2.compute.amazonaws.com/bookstoreservice/v1.0/customers/12345/orders>","mediaType":"application/json","method":"GET"},

{"use":"View the customer's detailed information","url":"<http://ec2-54-148-67-53.us-west-2.compute.amazonaws.com/bookstoreservice/v1.0/customers/12345>","mediaType":"application/json","method":"GET"},

{"use":"Search Books","url":"[http://ec2-54-148-67-53.us-west-2.compute.amazonaws.com/bookstoreservice/v1.0/books?title={title}&author={author}&ISBN={isbn}](http://ec2-54-148-67-53.us-west-2.compute.amazonaws.com/bookstoreservice/v1.0/books?title=%7Btitle%7D&author=%7Bauthor%7D&ISBN=%7Bisbn%7D)","mediaType":"application/json","method":"GET"}],

"customerId":"12345",

"lastName":"Bednarczyk",

"firstName":"Daniel"}

**BookRepresentation** class extends from abstract class and contains all details of the book. It has all fields that the domain object has but since it is marked with the Json serialization inclusion – NON\_EMTPY annotation, the only information that is sent is the information that the webservice sets in the representation object.

Example HATEOAS :

[{"links":[{"use":"View all details","url":"<http://ec2-54-148-67-53.us-west-2.compute.amazonaws.com/bookstoreservice/v1.0/books?title=Title11111&author=Author11111&ISBN=ISBN11111>","mediaType":"application/json","method":"GET"}],

"bookId":"11111",

"title":"Title11111",

"price":20.46,

"isbn":"ISBN11111",

"author":"Author11111"}]

**OrderRepresenation** also extends Abstractrepresenation and contain details about the order. It has all fields that the domain object has but since it is marked with the Json serialization inclusion – NON\_EMTPY annotation, the only information that is sent is the information that the webservice sets in the representation object.

Example in XML:

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>

<Order>

<links>

<link>

<use>Cancel Order</use>

<url>http://ec2-54-148-67-53.us-west-2.compute.amazonaws.com/bookstoreservice/v1.0/customers/12345/orders/1234

</url>

<mediaType>application/json</mediaType>

<method>DELETE</method>

</link>

</links>

<customerId>12345</customerId>

<orderDetails><orderDetails>

<book>

<links>

<link>

<use>View all the details of the book</use>

<url>http://ec2-54-148-67-53.us-west-2.compute.amazonaws.com/bookstoreservice/v1.0/books?title=Title11111&amp;author=Author11111&amp;ISBN=ISBN11111

</url>

<mediaType>application/json</mediaType>

<method>GET</method>

</link>

</links>

<bookId>11111</bookId>

<title>Title11111</title>

<price>20.46</price>

<isbn>ISBN11111</isbn>

<author>Author11111</author>

</book>

<quantity>2</quantity>

</orderDetails><orderDetails>

<book>

<links>

<link>

<use>View all the details of the book</use>

<url>http://ec2-54-148-67-53.us-west-2.compute.amazonaws.com/bookstoreservice/v1.0/books?title=Title11112&amp;author=Author11112&amp;ISBN=ISBN11112

</url>

<mediaType>application/json</mediaType>

<method>GET</method>

</link>

</links>

<bookId>11112</bookId>

<title>Title11112</title>

<price>19.99</price>

<isbn>ISBN11112</isbn>

<author>Author11112</author>

</book>

<quantity>1</quantity>

</orderDetails></orderDetails>

<orderState>Processing</orderState>

<orderTotal>60.91</orderTotal>

<defaultBillingAddress>

<addressId>123</addressId>

<street>1100 Milwaukee Ave</street>

<city>Chicago</city>

<state>IL</state>

<zip>60111</zip>

</defaultBillingAddress>

<defaultShippingAddress>

<addressId>123</addressId>

<street>1100 Milwaukee Ave</street>

<city>Chicago</city>

<state>IL</state>

<zip>60111</zip>

</defaultShippingAddress>

</Order>

## USING THE WEBSERVICE:

### CONNECTING TO THE WEBSERVICE:

To webservice is deployed on the Amazon’s ec2 web server. To connect to the BookStore webservice the clients can access the following URI: [http://ec2-54-148-67-53.us-west-2.compute.amazonaws.com](http://ec2-54-148-67-53.us-west-2.compute.amazonaws.com/)

This URI is the base URI for all the requests sent to the webserver.

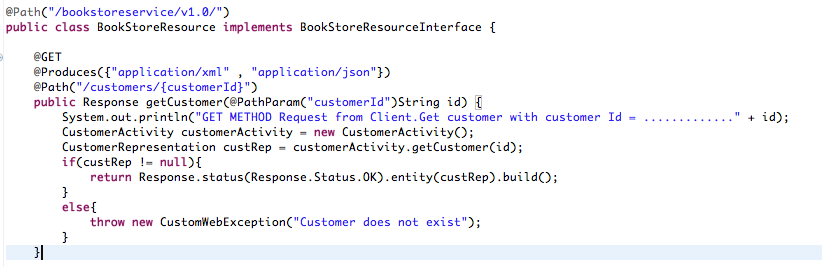
### EXAMPLES ON USING THE RESOURCE CLASS:

* Response getCustomer(String id);

This method is for retrieving a customer, the following URI can be used along with the GET method and the content-type received will be application json/application xml:

<http://ec2-54-148-67-53.us-west-2.compute.amazonaws.com>/bookstoreservice/v1.0/customers/{someCustomerId}

Based on the annotations, the webservice maps the URI to the particular method. For Example:



@Path : “/bookstoreservice/v1.0/ “: represents the root path for all the service methods in **BookStoreResource** class. v1.0 represents the version number.

@GET : is HTTP method

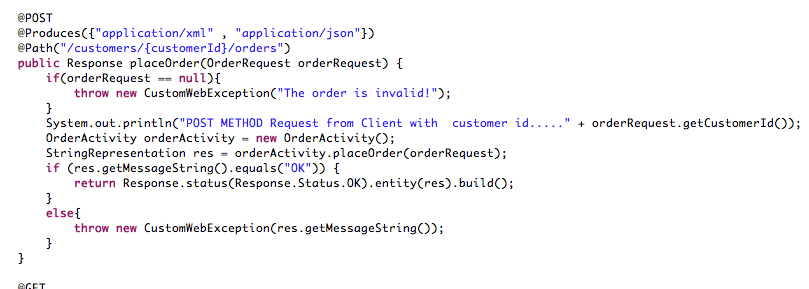
@Produces: The values along with this annotation state that the method can produce both xml and json representations.

@Path : “/customers/{customerId}” represents the path to reach the getCustomer method. For example: “/bookstore/v1.0/customers/”1234”” gets customer representation of customer with id : 1234

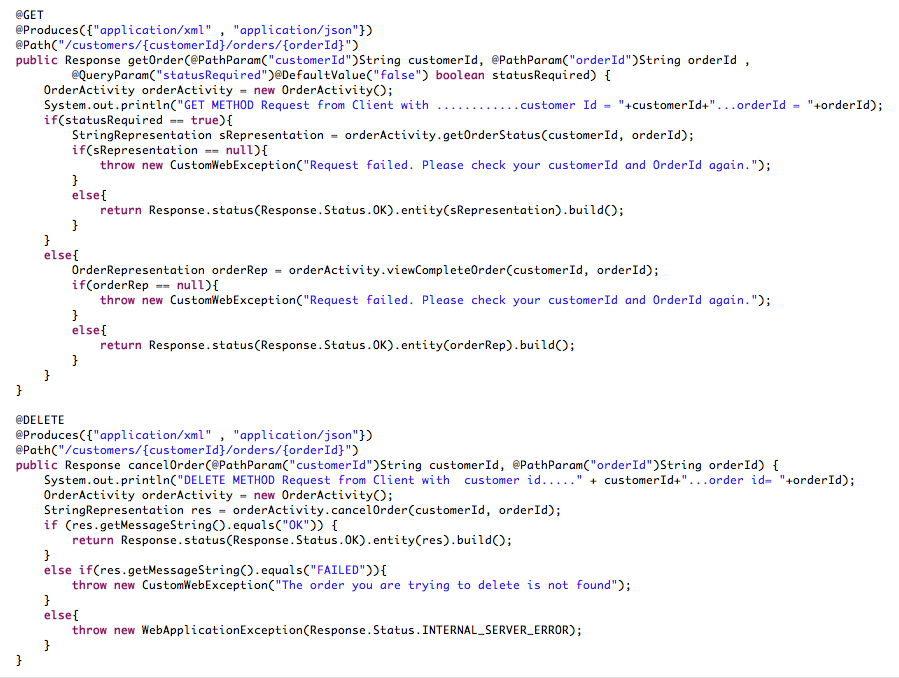
@PathParam: {customerId}. This maps the customer Id in the path to the customer id parameter of the java method.

* Response placeOrder(OrderRequest orderRequest);

This method is used to place orders for a particular customer.



* Below is an example of 2 methods with same path but different HTTP methods. One is to ‘get’ an order by a customer with specific id, another is to ‘delete’ the order placed by customer. They differ in the HTTP method used. The getOrder’s method can be invoked when a GET request is specified with the URI, and the cancelOrder’s URI can be invoked when a DELETE request is specified along with the URI.



The method used @PathParam for ‘customerId’ and ‘orderId’ where as @QueryParam for ‘statusRequired’ with a @DefaultValue : false for the later.

So the path looks like: “/bookstore/v1.0/customers/”1234”/orders/”123”? statusRequired=true”.

The other methods in the resource class can be invoked in a similar manner by calling the URI stated by the path along with the defined methods. The section below describes some of the requested URI’s and their returned responses. The links in the responses state how the other URI’s can be called.

### RETURNED RESPONSES: **Book:** Request: <http://ec2-54-148-67-53.us-west-2.compute.amazonaws.com/bookstoreservice/v1.0/books?title=Title11111&author=Author11111&ISBN=ISBN11111>

**Sample JSON Response:** [{"links":[{"use":"View all details","url":"http://ec2-54-148-67-53.us-west-2.compute.amazonaws.com/bookstoreservice/v1.0/books?title=Title11111&author=Author11111&ISBN=ISBN11111","mediaType":"application/json","method":"GET"}],"bookId":"11111","title":"Title11111","price":20.46,"isbn":"ISBN11111","author":"Author11111"}]  
  
  
Customer:  
[**Request:** http://ec2-54-148-67-53.us-west-2.compute.amazonaws.com/bookstoreservice/v1.0/customers?username=Customer1112&password=Password1112](http://ec2-54-148-67-53.us-west-2.compute.amazonaws.com/bookstoreservice/v1.0/customers?username=Customer1112&password=Password1112)  
  
**Sample JSON Response:** {"links":[{"use":"View all orders","url":"http://ec2-54-148-67-53.us-west-2.compute.amazonaws.com/bookstoreservice/v1.0/customers/12345/orders","mediaType":"application/json","method":"GET"},{"use":"View the customer's detailed information","url":"http://ec2-54-148-67-53.us-west-2.compute.amazonaws.com/bookstoreservice/v1.0/customers/12345","mediaType":"application/json","method":"GET"},{"use":"View Books","url":"http://ec2-54-148-67-53.us-west-2.compute.amazonaws.com/bookstoreservice/v1.0/books/","mediaType":"application/json","method":"GET"}],"customerId":"12345","lastName":"Bednarczyk","firstName":"Daniel"}  
  
  
  
Order:  
**Request:**  <http://ec2-54-148-67-53.us-west-2.compute.amazonaws.com/bookstoreservice/v1.0/customers/12345/orders/1234>  
  
  
**Sample XML Response:** <?xml version="1.0" encoding="UTF-8" standalone="yes"?><Order><links><link><use>Cancel Order</use><url>http://ec2-54-148-67-53.us-west-2.compute.amazonaws.com/bookstoreservice/v1.0/customers/12345/orders/1234</url><mediaType>application/json</mediaType><method>DELETE</method></link></links><customerId>12345</customerId><orderDetails><orderDetails><book><links><link><use>View all the details of the book</use><url>http://ec2-54-148-67-53.us-west-2.compute.amazonaws.com/bookstoreservice/v1.0/books?title=Title11111&amp;author=Author11111&amp;ISBN=ISBN11111</url><mediaType>application/json</mediaType><method>GET</method></link></links><bookId>11111</bookId><title>Title11111</title><price>20.46</price><isbn>ISBN11111</isbn><author>Author11111</author></book><quantity>2</quantity></orderDetails><orderDetails><book><links><link><use>View all the details of the book</use><url>http://ec2-54-148-67-53.us-west-2.compute.amazonaws.com/bookstoreservice/v1.0/books?title=Title11112&amp;author=Author11112&amp;ISBN=ISBN11112</url><mediaType>application/json</mediaType><method>GET</method></link></links><bookId>11112</bookId><title>Title11112</title><price>19.99</price><isbn>ISBN11112</isbn><author>Author11112</author></book><quantity>1</quantity></orderDetails></orderDetails><orderState>Processing</orderState><orderTotal>60.91</orderTotal><defaultBillingAddress><addressId>123</addressId><street>1100 Milwaukee Ave</street><city>Chicago</city><state>IL</state><zip>60111</zip></defaultBillingAddress><defaultShippingAddress><addressId>123</addressId><street>1100 Milwaukee Ave</street><city>Chicago</city><state>IL</state><zip>60111</zip></defaultShippingAddress></Order>