Messaging (MSG)

Exercise MSG –Messaging with RabbitMQ

The Setup :

Download and install RabbitMQ from: https://www.rabbitmq.com/download.html

From Sakai download the projects W3D4-MSG-Sender and W3D4-MSG-Receiver. Make sure that RabbitMQ is running, and then run the Sender project to send a message. Next run the Receiver project and check that the message is received.

Create a copy of your W3D3-REST project and call it **W3D4-MSG**, and add the following dependency to the pom:

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-amqp</artifactId>
</dependency>
```

The Exercise:

The purpose of this exercise is to add the ability to receive AMQP messages into into our already fully featured Book project.

To start, create a BookListener.java file, whose purpose will be similar to what our Controllers do, but then for RabbitMQ messages. The simplest is adding a Book.

Part A: adding a book

- In your App.java add a @Bean to create a 'book' queue similar to the 'hello' queue in the sender and receiver project. Make sure the type of your queue is: org.springframework.amqp.core.Queue;

- Annotate your book listener as @Component and @RabbitListener(queues = "book"), and autowire it to receive the bookService

- Make sure that your Book class implements Serializable

- Make a add method that receives a Book object and annotate it as @RabbitHandler, the method should simply pass it on the the bookService.add() method. It's probably wise to put it inside a try / catch block to catch any ConstraintViolation exceptions that validation may throw.

- Update the W3D4-Sender project to send a Book object instead of a Person object

- Remember to update the the Queue in the sender project to "book" both inside the sender method and inside the Spring Boot @Bean configuration of the project.

- Test to see if it works (you can look either on the web page or web-service)

Part B: updating a Book

- We can use the same trick that JPA uses to check if we should update or delete an object – by checking if the ID is null.

- Update your @RabbitHandler method to check the id of the book being sent. If it is null the method should add it, otherwise it should update it.

- Again test to see if it works

Part C: deleting a Book

- You can have multiple @RabbitHandler methods in a single @RabbitListner class. It decides which method to call based on the Type of the payload (the data type of the message you are sending).

- Add an additional @RabbitHandler method that receives an Integer id, and deletes the book with the given id through the BookService

- Test by sending a integer from the sender and check to see if that book was deleted

Part D: Using headers

- Although our trick with checking for null works, a it may be clearer to send an additional header saying if we want to update or delete.

- To receive a header in our @RabbitHandler we can add an additional String parameter into the method that is annotated with @Header("operation"). In other words our method may look like:

```
public void save(Book book, @Header("operation") String operation) {
```

- Then in order to specify the operation header on the sender we can add a 3rd method to the .convertAndSend() method of type function (taking a Message and returning a Message). This function then gets the headers from the message properties and adds a value to it Similar to what is shown below:

```
template.convertAndSend(queue, msg, m -> {
    m.getMessageProperties().getHeaders().put("operation", "add");
    return m;
});
```

- Using this we can specify that the operation should be 'add' or 'update'.

- Once again test to make sure your code works