eCabs - Backend Technical Task

Dear candidate, we are thrilled you are interviewing with eCabs!
We'd like you to complete a coding exercise to take the interview process to the next level. We genuinely encourage you to ask lots of questions if you need clarification, if you have any ideas or if something isn't going as planned. This is for you as much as it is for us. We want you to get a sense of what it's like to work with us.

Challenge

Your task is to build a simple backend API layer to manage bookings. You are requested to use a message broker (specifically RabbitMQ) to manage the events produced and consumed by and from your application. Moreover, it is requested that your application exposes a REST layer to manage booking operations (CRUD).

What's expected

1. Use Spring Boot to create the artifact using the embedded Tomcat web container
2. Use RabbitMQ as message broker (more details below)
3. Your application should create/configure the RabbitMQ exchanges and queues on application startup and not using the RabbitMQ CLI
4. Use an embedded database like the H2 database
5. Use Spring Data to interact with the database and Hibernate as the JPA implementation

RabbitMQ Topology

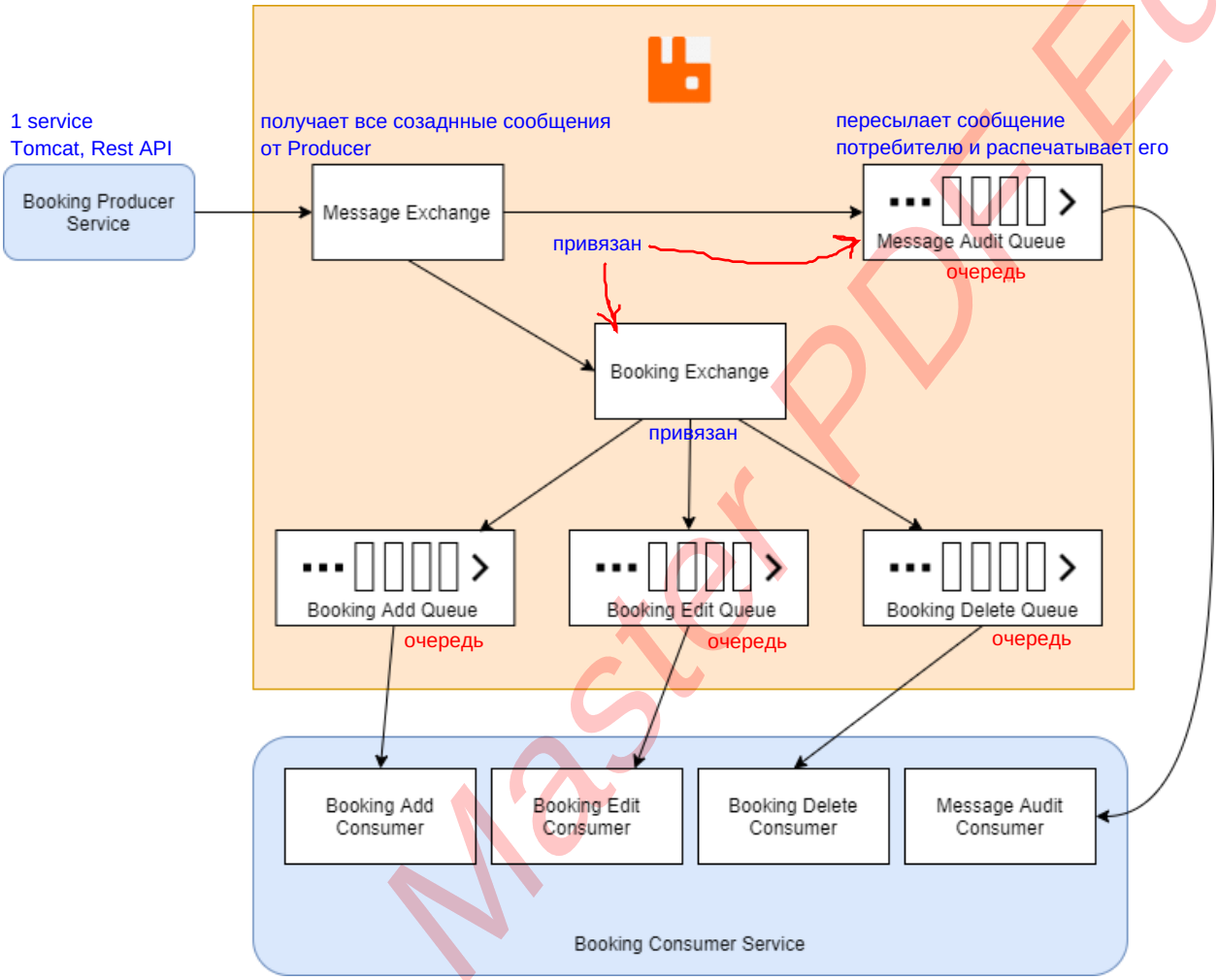
It is required that your application will configure the message broken as shown in the picture below. The configuration is up to you as far as you keep the defined topology composed as following:

Message Exchange: The **MessageExchange** is the one receiving all the messages created by the **BookingProducerService**. It has to be bound to the **MessageAuditQueue** and the **BookingExchange**.

Message Audit Queue: The only purpose of the **MessageAuditQueue** is to forward the message to its consumer and print the received message.

Booking Exchange: The **BookingExchange** will have the following queues bound to it:

1. **BookingAddQueue**: its consumer will create the booking into the database
2. **BookingEditQueue**: its consumer will update the booking into the database
3. **BookingDeleteQueue**: its consumer will delete the booking from the database



Your main aim here is to come up with a correct routing strategy that will achieve the previously described topology as well as making use of the correct type of exchanges.

RabbitMQ configuration

To avoid setting up a fully-fledged RabbitMQ installation, it is advised that you get rabbitMQ Official docker Image from here.

Download the image using the following docker command: **docker pull rabbitmq**

Create and start the docker container using the following: **docker run -d --hostname ecabs-rabbit --name rabbit-assignment -e RABBITMQ_DEFAULT_VHOST=/ -p 15672:15672 rabbitmq:3-management**

After starting the rabbitMQ container, you can then access the rabbitMQ web interface here: http://localhost:15672/
with credentials:
username: guest
password: guest

Model

The model classes you should use are highlighted next. The two classes you're going to use are, the **Booking** class modeling a booking entity and the **TripWaypoint** class that models a stopping point during a booking. A booking may have multiple stops.

Booking model:

- Id
- Passenger Name
- Passenger Contact Number
- Pickup Time
- Asap
- Waiting Time
- Number Of Passengers
- Price
- Rating
- Created On
- Last Modified On
- Trip Waypoints

Trip Waypoint model:

- Id
- Locality
- Latitude
- Longitude

Deliverable

You are requested to provide your code as a Maven multi-module project, both modules should be provided as a self-contained micro-service that can be started independently from the other. You are free to have as many modules as you see fit inside your project, as long as the producer and consumer modules are kept separate.

Please provide your code on GitHub and give access to the following email addresses:

- mandyzammit@ecabs.com.mt
- grzegorzgolowicz@ecabs.com.mt

If you have any queries, please do not hesitate to contact us at the email addresses above. Thanks and enjoy your challenge!