Enterprise Middleware

COURSEWORK REPORT

ALEXANDROS

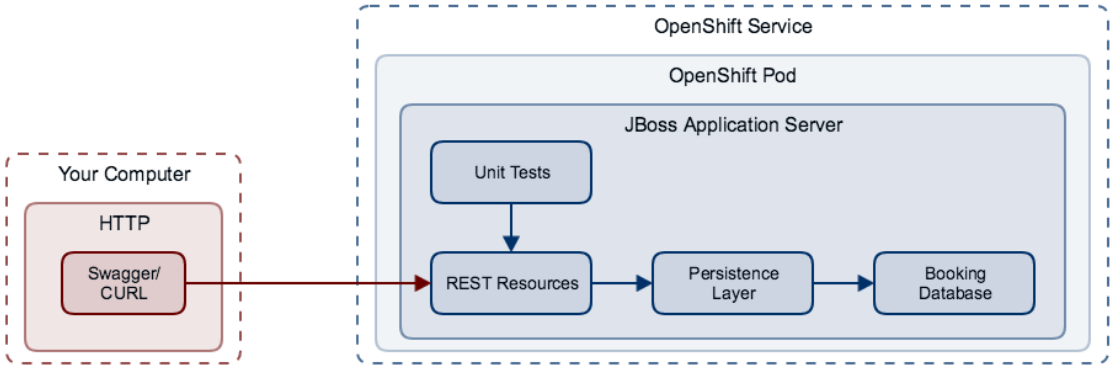
DATE

20/11/2019

DESIGN EXPLANATION

Part 1

As it is depicted in the coursework instructions this is the architecture which I implemented for each of my Entities.

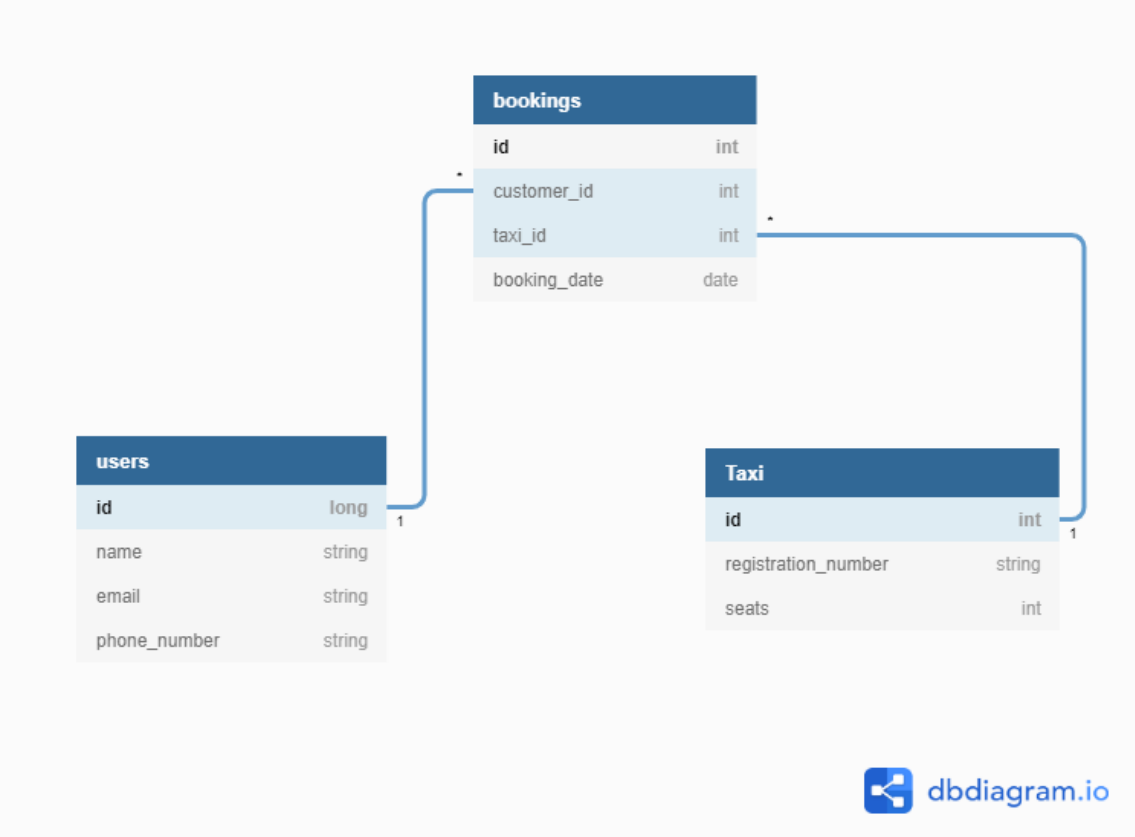


The client sends a HTTP Request to the server which responds with the appropriate response code and message depending on whether the Request was successful or not. Requests includes operations for creating, reading and deleting persistent objects from the database. Each request fires the corresponding Rest Service which communicates with the database via the Java Persistence API. More specifically, each Entity handles each Request on *EntityName*RestService file and then after checking in some cases the JSON input it calls the *EntityName*Service in which if the validator(if it exists) does not throw an exception the Repository is called in order to persist the object in the database.

Functionality of the RESTful Resources has been implemented as was instructed in conjunction with the validation of all parameters. Responses and code in general are well documented.

Part 2

In order to specify the relationship between a user and a commodity I used Bidirectional Relationship so both entities know about its related object. This is the database diagram :



Also cascade deletion was added to both database relationships and as a result whenever a user or a commodity is deleted the corresponding booking record is deleted as well. As long as fetching is considered EAGER was used since LAZY was returning null id's sometimes.

The ability of registering a Customer to the database and also booking a Taxi at a specific date was also implemented in this coursework via the GuestBooking Post request. The use of UserTransactions was a necessity for this task since both objects should be persisted in the database or none.

Part 3

For Part 3, a travelAgent was created with default customer id's for each commodity. This service is checking if the given customer_id and taxi_id exist in the database and if so continues with a processing a transaction which involves creating a booking in my database and then tries to remotely add a flight and a hotel resource on other's endpoint. Only if all requests were successful, a travelAgent object is created and that was achieved again using Transactions like Part 2. The main difference is that the UserTransaction.rollback() functions only deletes the object on the local database so it is needed to remotely delete any created objects by sending a delete request on the other's API. That is done in org.jboss.quickstarts.wfk.travelagent.TravelAgentRestService.roolback(), in which I check if I have a booking_id for either flight or hotel booking and call the corresponding delete function.

Many obstacles had to be overcome for this part since we had no communication with each other and not everybody followed the same principles or had the same data structures. So even though my code is working with Jackson Abraham's & Alexander G. McClelland's APIs it doesn't mean that it will not work without any tweaks for any other Flight & Hotel API respectively.

FEEDBACK

Overall, this coursework felt quite frustrating. The main problem was that the lectures covered the theoretical part of the coursework without any code examples and we were not taught the logic behind the frameworks we used. For example, I strongly believe devoting one hour of explaining in depth the example code we were given would be ideal.

Also, this coursework was primarily about researching how to use these frameworks and did not require any specific critical thinking thus was boring. Students needed to taught themselves a particular framework in which I found no motivation whatsoever. As far as part 3 is concerned, relying on others while you are not in the same team with them felt very uncertain and worrying. Maybe you should have provided us with your own APIs or put a separate deadline for Part 1 & 2 so we are all on the same page.

On the other hand, I acknowledge the fact that this was a project was a realistic representation of a real work project and represents the most common problems that are occurred in using a REST API.