Swathi Shree

SOFTENG 325 Assignment 01 Report

1. Improving the Scalability of the web service:

Web services are becoming integral as they gain more popularity over time. To handle scalability issues, REST principles, such as statelessness and caching, have been implemented. Scalability is the ability to overcome performance limits by adding resources. In REST web services, the server end is stateless, resulting in nothing being stored across requests and making it horizontally scalable. Statelessness is used to authenticate a user through the use of cookies to store a sessionId. If login is successful, a sessionId is created and returned in a cookie and stored in a database. Caching is another way to improve scalability where the server's response is stored in the client to prevent the client from requesting the same resource again. Caching is achieved implicitly by optimistic locking repeatable reads without trading away scalability.

2. Lazy Loading and Eager Fetching:

Lazy loading occurs when the initialization of objects is delayed until they are needed. This improves startup and overall performance. An example of explicit lazy loading is User in the Booking class, where Booking data would not be loaded and initialized unless an explicit call is made to it. Lazy loading is used as users may browse various concerts and dates, and not necessarily book any shows yet.

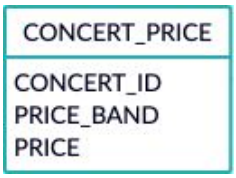
In contrast, eager loading means collections are fetched fully at the same time as the parent. The Concert class has Set<LocalDateTime>, which would fetch all the dates simultaneously with the Concert. Eager loading is required in this case as it allows the users to see all the available dates to make a decision.

3. Concurrency Handling:

Optimistic locking is used to address the issues that arise from concurrent access. Using optimistic locking, the client is able to make local changes to a resource. When an update request is attempted the client is notified if those changes are rejected by the server. This is usually done by adding the current version or last modification timestamp attribute. By using optimistic concurrency, it only throws an error when the same seat is accessed. This is primarily done with the use of OPTIMISTIC locks. This is especially useful when dealing with the high load experienced when tickets go on sale.

4. Extend Web Service for New Features:

<u>Ticket Prices Per Concert</u>
Create a persistent/database table called CONCERT_PRICE in which ticket prices for various concerts can be maintained. E.g.




```
INSERT INTO  CONCERT_PRICE(CONCERT_ID,PRICE_BAND,PRICE)VALUES(1,'Platinum','$100.00');
INSERT INTO  CONCERT_PRICE(CONCERT_ID,PRICE_BAND,PRICE)VALUES(1,'Gold','$90.00');
INSERT INTO  CONCERT_PRICE(CONCERT_ID,PRICE_BAND,PRICE)VALUES(1,'Silver','$80.00');
```

Multiple Venues

Create a persistent/database VENUE table where VENUE_ID is the primary key and consists of a list of venues and other relevant information (e.g. address). E.g.

```
INSERT INTO VENUE (VENUE_ID,NAME,ADDRESS,CAPACITY) VALUES (1,'Spark Arena','123 Queen St',500);
```

The CONCERT_DATES table can be expanded by adding a new column VENUE_ID. E.g.

```
INSERT INTO CONCERT_DATES (CONCERT_ID, DATE,VENUE_ID) VALUES (1, '2020-02-15 20:00:00',1);
```

Hold Seats

Create a class called SeatHold, which contains a method called holdSeats. This method takes in the number of seats available(numSeats) and the user's ID (userId) and returns the SeatHold_Id and the available seats, including the seats that were previously on "hold" (user cancels their payment/time period elapses results in the seat to return to available status).To ensure that the seat returns to an available seat once the time period has elapsed, a timer can be implemented by deducting a predefined amount of time from the CURRENT_TIMESTAMP. To release hold seats, compute the seat holding time which is the difference between the CURRENT_TIMESTAMP and the time the user locked the seat(which is always less than the CURRENT_TIMESTAMP). If the holding time is greater than the predefined time, the seat can be released.

predefinedHoldingTime = 600 seconds
SeatHoldingTime = CURRENT_TIMESTAMP - SeatLockedTime (in seconds)
if (SeatHoldingTime >= predefinedHoldingTime), then release the seat

The Timer can be implemented in two places :
At client level : a client side javascript timer can be implemented. When the Client-side Timer expires, it calls the web service to release the seat.
At server level: If the browser/client session is closed, the "held" seat needs to be released by implementing a server-side timer. The server timer starts when the application starts, and would run indefinitely/till application closes. As the seat holding status is maintained in the database, the server timer will clean up the expired holding seats.