SOFTENG 325: Assignment #1

September 27, 2020

Aiden Burgess
abur970 - 600280511

Scalability - How have you made an effort to improve the scalability of your web service?

The web service does not hold any information about client sessions. Instead, it provides cookies to the client which they may use for authentication. These cookies are obtained through the login service, and used when retrieiving user-specific information, or performing changes to the database. This allows the web service to be easily replicated. Each replicated server acts identically to handling the requests.

The web service uses a RESTful architecture, which allows us to perform optimisations due to the use of HTTP methods.

Resources are split into different classes, which allows more functionality can be added later on. It is also easier to expand functionality of existing resources.

Identify (implicit and explicit) uses of lazy loading and eager fetching within your web service, and justify why those uses are appropriate in the context of this web service?

Concert lazy loads performers and dates, as when concert summaries are retrieved, all the performers and dates will be retrieved, without being used. These have been further optimised to user FetchMode.SUBSELECT, avoiding the n+1 problem, as when retrieving performers and dates, we need all of them.

Booking also uses FetchMode.SUBSELECT to retrieve the seats for a booking, as currently, when retrieving a booking, we always need all the seats. The ConcertDate and User are lazy loaded, as these are not always needed, and there is no n+1 problem as these are single values.

ConcertDate lazy loads Concert, as often we only need to retrieve the date, which increases our performance.

How have you made an effort to remove the possibility of issues arising from concurrent access, such as double-bookings?

A single transaction is used to book all seats at once, so if an error is found - such as a seat already being booked - the entire transaction is rolled back. This conforms to the requirement that either all seats are booked or no seats are booked.

The Seat domain class has a @Version attribute. By default, JPA will perform optimistic locking when a change to an entity with a version attribute is detected. This ensures seat versions have not changed at commit time. If they have changed then the full transaction is rolled back.

Subscriptions are iterated through a synchronized() block, which prevents issues from concurrent access of bookings.

How would you extend your web service to add support for the following new features?

Support for different ticket prices per concert (currently ticket prices are identical for each concert)

Add ticket price field to Seat, which is determined when the seats are initialised for a new concert date. This field should also be added to the seatDTO class, as the client might need such information.

Support for multiple venues (currently all concerts are assumed to be at the same venue with an identical seat layout)

Currently the web service relies on only one concert allowed per date. With multiple venues, it is assumed that there may be multiple concerts per day. To accomodate this change, a Venue class should be introduced, which stores information about a venue, such as the id, name, and description. A venue field will be added to the Concert class. Concert will be made an id attribute of ConcertDate, as there can be more than one concert per date. A venue DTO class will be added, and concertDTO will be modified. Some retrieval queries will be modified in the ConcertUtils class.

A VenueResource class will be added to support endpoints regarding venues.

Support for "held" seats (where, after a user selects their seats, they are reserved for a period of time to allow the user time to pay. If the user cancels payment, or the time period elapses, the seats are automatically released, able to be booked again by other users).

The bookingStatus enum will be extended with a HELD value. Endpoints will be added to the BookingResource to support reserving and paying for a booking. In order to ensure no double bookings, the seats will be labeled as booked, until either a user cancels or the time is exceeded. To support the periodic removal of reservations, a timer can be set up to trigger at a set interval, checking a list of reservations to confirm if their time period has elapsed.