325 Scalability of REST interface – mbea966

The implemented system helps to meet the scalability requirements in a number of ways, including caching, transferral of compact resources in requests, null checking, and locking.

Caching on the client side has also been initialised to improve the scalability of the system. A set of all concerts and all performers are stored by the client. When the client calls one the method to get all concerts or performers if the cache has not expired, instead of sending a request to the server, the client can return the cached entities. This reduces load on the server requests as for entities are only made when the cache is expired. As this is a large HTTP payload, this significantly improves the scalability of the system. The time after which the cache expires is defined on the server end, as the server should be controlling its data integrity. Caching concerts and performers is a smart choice as they are less likely to be frequently in need of update, so a relatively long caching timeout can be implemented by the server if appropriate. I chose not to cache bookings as these are constantly changing when users make and confirm reservations and out of date data can lead to invalid bookings and reservations. Caching means there is a compromise, when the cache is up to date the client is very quickly able to retrieve all their desired entities from the cache, however the data may not be up to date.

Images are also cached by the server. To retrieve an image, a request to AWS is required, this is inefficient to do frequently due to the relatively large size and number of images. In many GUI applications, it is likely that images are to be reused. Caching these images will reduce the load time of images and reduce network traffic to retrieve them.

From the client side, the number of requests to the server were reduced where possible. One method of reducing traffic to the server was checking the information supplied by the client for null values. It is straightforward to verify null data on the client side and does not require any interaction with the database. If any of the required fields are not set, the request will not go through. It is a waste of resources to send an HTTP request to the server to determine this. When the system is experiencing a high load, the server will in total receive less requests since those with uninitialized fields will not make it to the server.

To reduce load on the server, only bare-bones DTO objects were transmitted in requests as opposed to the more complex domain objects. Domain objects tend to carry more data than their corresponding DTO which means they generate more strain on the system as larger data payloads need to be transmitted across the network. Sending only DTO objects reduces some of this strain as only the necessary data is transmitted.

When experiencing a high load, it is likely that users will concurrently be attempting to reserve seats for a concert on a date. This can quickly lead to issues of users being misinformed of the state of reserved and available seats and invalid bookings and reservations can be generated. To combat this, optimistic locking was implemented. This means that if users are trying to book seats for a concert at the same time, available seats will be locked for the duration of the transaction which ensures that a seat cannot be double booked. This could be made more efficient by only locking the seats that are about to be booked, rather than all the available seats.

All classes are stateless except for the resource class associated to the subscription service. This further improves the scalability of the system as many servers can be set up and run in parallel to process a high load of requests. This can be particularly helpful in dealing with blocking requests, those requests that take a long time to process do not affect the requests that are being run on another server. By nature, HTTP requests are stateless, so a REST interface lends itself well to scalability. For further improvements in scalability, methods could be taken to make the news item resource class stateless as this means subscription services could be carried out concurrently.

The domain model has been designed for optimal speed of queries, in most cases domain classes contain all the information needed by the system in a request which means joins are not necessary to compute queries. This is a compromise as it does mean that size of the database is larger.

Overall, the system is effective in improving scalability although changes could be made for further improvements. Changes include, making subscription services stateless to allow parallel executions, minimising the amount of duplicate data in the domain classes for a more compact database and locking only the relevant seats when generating a reservation.