SE 325 A1 Report

elee353, 840454023

At the networking application level, HTTP is used by REST for all four CRUD operations. The HTTP protocol is a stateless protocol. Stateless protocols reduce memory and eases the replication concerns, as the state itself doesn't need to be stored on the server.

REST is an architectural style that utilises standards like HTTP and XML. This promotes flexibility of my design as support for additional standards such as JSON can be easily added. REST defines the Web as a distributed hypermedia application. Hyperlinks are implemented within hypertext and are used for exchanging representations of the resource state. New hyperlinks can be easily implemented.

There are several aspects that improve the scalability of my design.

In general, the less database calls, the better the performance. Caching has been used to achieve this objective. When the first time the user tries to download the performer images, the images are downloaded and cached in the DefaultService. The cached images are retrieved when needed later, instead of downloading them from the AWS Bucket again. Another example of caching is where some domaining class objects are cached in DefaultService. This promotes scalability because the database calls are reduced by retrieving the cached objects instead.

Fine grained access control promotes scalability. In my design, the Performer and Concert domain classes do not store references to the bookings and reservations. This achieves fine grained access control and prevents the lockout problem. The lockout problem happens in a coarse grain access model. An entire table can be potentially blocked out by an increased number of users trying to book and reserve concerts. An example of this is a model where a

concert class contains seats.

My concert application utilised a three-tier architecture. In a three-tier system, the three main components involved are the client, server, and database. A three-tier architecture is more scalable than a 2-tier architecture because the web-tier and middle-tier can be scaled differently if necessary. The DTO pattern is also applied. DTOs enforce *separation* of the web tier from the business logic tier.

An application server can be used to cache persistent data to increase performance and scalability. There is no direct accessible links between the client and the database and the database is exclusively accessed by the server. This reduces the need for a pessimistic lock.

Optimistic concurrency control (OCC) is used to promote scalability. It allows fast performance and high concurrency (access by multiple users), at the cost of occasional conflicts. The ‘first commit wins‘ characteristic matches the ‘first in first served’ nature of booking a seat in a concert. In contrast, a pessimistic lock normally requires a direct connection to the database, as would typically be the case in a two-tier client server application.

Pessimistic locking requires overhead for every operation, whether or not multiple users are actually trying to access the same record. The overhead is small, but adds up because every row that is updated requires a lock. Furthermore, every time that a user tries to access a row, the system must also check whether the requested rows are already locked by another user or connection. In a highly concurrent system such as our concert application, these locks can prevent scalability.

Many aspects of the program are stateless, but the news item component has to be stateful. Asynchronous response is utilised to increase scalability of this particular module. With asynchronous response, the client is not blocked while waiting for a response. The server threads are not devoted for maintaining a synchronous communication. They can therefore be used for other computing tasks.

Lazy loading is used in my design. In most situations when booking a concert, the user is likely to view the list or overview of the concert attributes. The user is unlikely to view a nested set of concert attributes. In addition, it may not to be decided yet whether the user would like to view the data in question, e.g. having a list of concert attributes in memory and the request for some particular attributes depends on future actions.

Lazy loading also prevents the Cartesian product problem.

Eager /Lazy fetching

At the moment, there is a thread that times the 5 second session in the Default Service class. In the future, this can be done by logging the start and end sessions of booking and comparing if the difference is within 5 seconds. This will reduce the use of the thread resources, hence increasing scalability.