In the current web software climate it is more often than not that services become popular very quickly. Hardware resources have been made easy by technology such AWS but software needs to designed to be able to scale from the get go otherwise just on the ramp up of popularity it will crash and burn. The project was made scalable with five key concepts; caching, statelessness, data design, asynchronous messages and optimistic concurrency control.

With more than one client is likely that they will try to reserve seats at the same time. To avoid know when this happens the JPA implementation is set to stop concurrent writing on commits. Naive implementations will throw this error when anything on the same table is written to or even just when the table is read. By using optimistic concurrency it only throws this error when the same seat is being accessed. This means many users can reserve seats at once rather than having to wait for someone else reserving seats to another concert or the same concert but different seats. This vastly increases the seat reserving throughput especially when users are trying to all get seats to the same concert when tickets go on sale.

There is a seat entry per concert and per date-time so if two users are reserving the same seat number at the same time for different performances the optimistic con-currency will allow it because they're not on the same row.

The web service is stateless (REST-based) so any request by any client can be serviced by any server replica on any server. This provides great horizontal scalability because many servers can just be setup, run and work instantly to deal with load. This also allows for running multiple server replicas on a server to take advantage of when one might block and/or multithreaded processors which maximisses utility of individual servers.

Cookies help the stateless web service deal with individual users because any server replica can use the authentication token (cookie) to identify the user. The database stores the current token for the user so all data concerning the user is stored thus all server replicas can access that information.

Designing the domain model in a certain way lends it-self to more efficient storage usage, efficient updates and faster look-ups. Using an inherently unique value of an entry such as a performer id allows for faster look ups about performers from the concerts table. Databases sort along id values so finding entries is very fast. Ids also allow to store data that would other-wise be duplicated in one table to be stored just once in another leading to much less storage space being used and updates are faster because only one location has to have information updated.

The JPA implementation used is Hibernate which lazy-loads as much as possible (doesn't load objects from the database until used/needed) which saves quite a bit of memory. As the database populates with data from being used eventually it will be bigger than the memory of the server. Lazy-loading means the only a fraction of the domain model is loaded at a time; the servers will need less memory. Relatively to eager-loading there can be more requests serviced at a time because memory will be less of a restriction.

The asynchronous news-item system both has a positive an negative effect on scalability. It does mean that only on subscription and on new news that the server tries to send the news item reducing load by not having to check on every other interaction with the server whether the user needs to be sent news. The negative side is that there can only be server replica for the news items as it requires holding a list of subscribers (in the form of AsyncResponse) so it doesn't have any of the benefits of the stateless design of *ConcertResource*. Async means the server can push to the client priority scheduling

The client caches images which eliminates the server load from having to service requests for images that have already been serviced to a client. This is especially useful in a GUI application where images are likely to be reused. This increases scalability by reducing load on the processor, memory and network of the server.

Caching is enabled for two basic *GET* methods; retrieving performers and concerts. The cache reduces server load because if the request was serviced recently then the server replica will not have to go through the process of loading all the objects then having to send them. Instead the client when it gets a special message from the server replica will just use the most recent copy it received.

Overall the scability of this project is quite good, the only short fall is the news item sending as it isn't currently stateless; this is future work.