**Functional requirement**

We are going to develop a commercial website for selling products where:

* Each product has a number of available items in stock.
* The system should be able to process at least **N = 6000** concurrent requests for **viewing** or **purchasing** products.
* The system is only allowed to have at most **S = 6** servers, where up to **3** servers can be used as relational databases.
* One relational database can serve at most **C = 300** concurrent connections.
* The system **must**: (main requirements)
  + ensure **data consistency**, which means that there will be never such a case where the number of successful purchased items is more than the number of available items in stock.
  + provide **real-time feedback** to its users with the **lowest possible latency**.

**Non-Functional requirement**

* have the **highest availability** with no point of failure.
* be **easily scalable** in the future when the number of products and users increase.

**High-level System Design**

When around **6000** requests update and read the entity concurrently and larger than the numbers of DB server can be able to serve (**300** connections). We can queue all the requests in a high throughput message queue. Then we can process them one by one in a FIFO sequentially

For this approach, we enable **highly available** as message queue guarantee the messages are always consumed by consumers.

**The load balancer and no point of failure**

To utilize **scalability**, we need a load balancer between client and app server so this enables highly available as the load balancer will simply route to healthy servers in case of server failure thus prevents any of application servers become single of failure

Another single point of failure is the DB server, to solve this problem we need the redundant copies of the DB. We can go with master-slave replication as **1** DB master, **1** DB slave synchronous and **1** DB slave asynchronous.

If DB master suddenly failure the data still available as DB synchronous can take place to become a master and if DB synchronous slave becomes unavailable the DB asynchronous is made synchronous. So at a time, we ensure the data always available to be retrieved.

**Data Consistency**

To ensure **consistency** data, the Optimistic Locking in JPA, if any concurrent update happen any update or delete need to compare the version and if the version does not match an exception is thrown indicate the items in stock sold out and warning to all the other users that item is no longer available.

**Lowest possible latency**

For real-time feedback and utilize latency, we will take the help of the message queue as a Push-Based Approach. Once the user provides feedback, the other transaction will update the payload into the database and other transaction send the payload to the message queue. Then the message queue will asynchronous push the feedback message to the user.

**The diagram**

Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings


Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
  
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings


**Technologies**:

Java

Web logic application server

Apache HTTP Server load balancer

JMS message queue

JPA