Northeastern University
CS6650 Scalable Distributed Systems Fall 2022
Nisarg Patel - patel.nisargs@northeastern.edu

BookBuyBookSell - Project Proposal

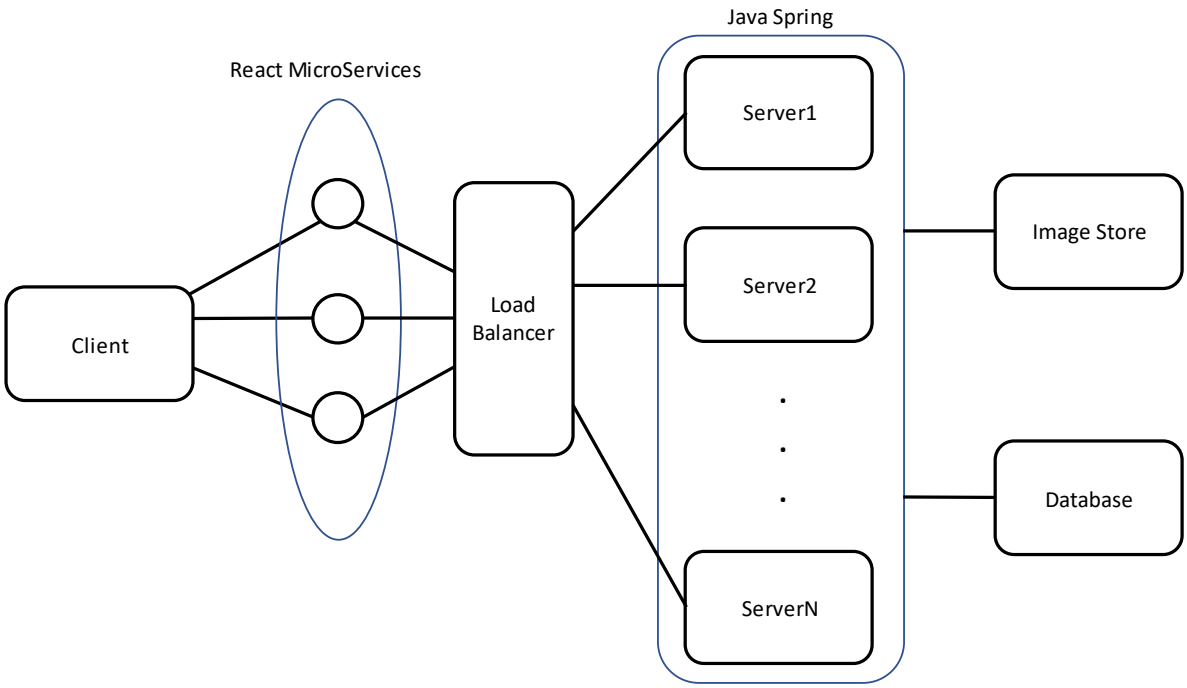
Summary:

For the final project, I propose a simple e-commerce application called BookBuyBookSell. It is a simple distributed application that would be a platform for user to lend or sell books. A user would have to create an account and login. User can maintain a collection of books on his name and put some of them to lend or sell. Interested users can lend/buy those books, sending request to the owner. Owner will then approve the request.

Since this project is focused mainly on the distributed aspect, the functionality is kept as simple but sufficient to be completed in the required time.

Architecture Overview and Design Description:

The image below shows the proposed architecture overview diagram of the application.



In this architecture, multiple clients can login at the same time. The graphical user interface part follows the micro frontend design. Each of these services communicate with server and perform real time updates. The server will be assigned based on a load balancer. The state

of the system is replicated among these servers. The server interacts with a database and an image store.

Implementation Approach:

The GUI part will be implemented using React and the backend will be implemented in Java and Spring.

Some requirements of this system include:

Functional Requirements:

1) A user should be able to signup/login.
2) A user can add a new book into collection
3) A user can put a buy to sell/lend
4) A user can buy a book.

Nonfunctional Requirements:

1) A request from a user should be executed within a specific period of time (5 seconds).
2) The system should not crash if one server crashes, thus the system should be robust.
3) A crashed node should be able to recover to the state it was before crashing.

From the functional requirements, there are 4 major services (use cases): LoginService, AddBookService, SellService/LendService and BuyService.

LoginService/SignupService:

This service would take the email and password as the arguments, and request server to check the credentials and fetch the account details of the user.

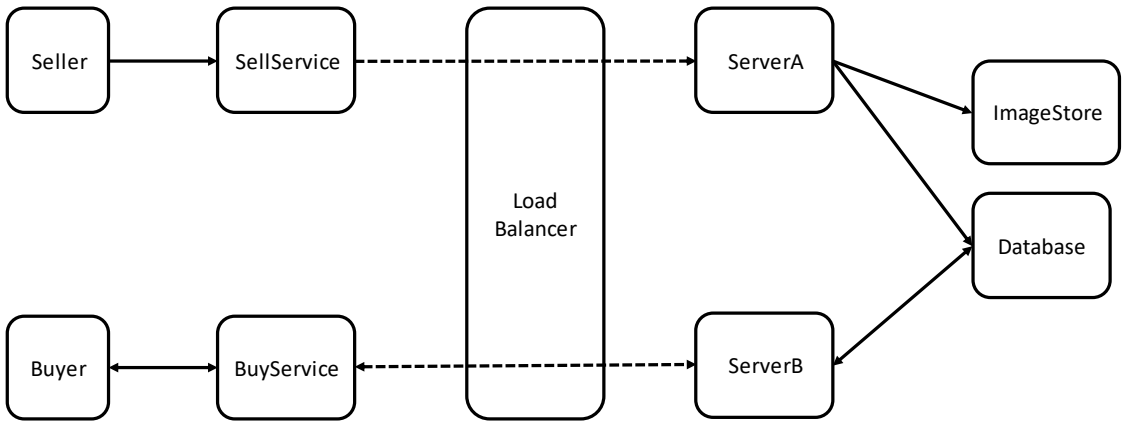
AddBookService:

It would take details of book as the argument along with some images of the book, and request server to add this book into the database. Since some images can also be requested to be stored, a separate dedicated image store is required.

SellService/LendService/BuyService:

This service can be called by a seller to make the book available for other users to buy/lend. A buyer will call BuyService to a specific book. The server checks if the book is still available and changes the status of the book accordingly. Seller will get updated upon the book being sold.

The below image shows the database interaction by a seller who calls SellService and a buyer who calls BuyService.



Based on these services, the following database schema seems to fit the criteria:

User: email, password, name, booklist

Book: id, name, status, imagelist

This will get updated once the system is actually implemented.

(Status could be one of IN_SHELF, SELLING, SOLD, LENDING, LENT)

Algorithms:

I will be implementing the following distributed systems algorithms:

1) **Time and Clocks:** Since a seller can get multiple concurrent requests to buy a book, it is required that requests should be timestamped and server handles the requests in the order of "happened before". For this, I would implement the Vector Clock timestamps for the requests and responses.
2) **Distributed Mutual Exclusion:** The updates on the database should occur in the order of the queries. To perform that, I will be implementing distributed mutual exclusion algorithm, specifically the Ricart-Agarwala algorithm.
3) **Distributed Transactions:** The changes in database should be atomic. Since multiple database calls can occur during a single buy/sell method, I will be implementing a distributed transaction algorithm.

4) **Fault Tolerance:** Since a server can crash anytime, I will be implementing a fault tolerance algorithm to make the system robust to single server failure and to restart the server with current start of the system.

The exact algorithm of Fault Tolerance and Distributed Transaction is yet to discuss during lectures and to study.

Expected Result:

The final application made will be a simple representation of a complex distributed system that will follow the basic principles of any such systems.

- The system will be heterogeneous, that it can be run in any machine.
- Scalability can be achieved such that more servers can be added and different micro services can be implemented to make the system perform complex operations.
- Failures on messages and servers will be handled by different mechanisms including timeouts and redundancy
- Different transparencies would be achieved based on the algorithms implemented and the properties of the system.

Finally, the expectations from this final project also include the learning about how to build and make a distributed system scalable and understand some of the distributed systems algorithms that help achieve the functionality similar to that of a simple system.