**Lab 1 Report**

**Demo video** -> <https://youtu.be/hwmtcHzptpQ>

**Introduction**

Purpose –

To create an e-commerce platform for buyers to search and purchase custom made items, and for sellers to sell customized items

Goal –

Build the application using MySQL as the primary database with a Node.js/Express server, and use React to develop the front end of the website.

**System Design**

The database used is MySQL and the tables are not normalized. We have the following entities-

* User
* Shop
* Product
* Favourite
* Cart
* Order

The ORM library Sequelize is used to execute CRUD operations efficiently. The backend is built using Node.js/Express, and REST API routes are implemented to allow the client to interact with the server. A few of these routes are-

* POST /auth/login, POST /auth/register, POST /auth/logout
* GET /product/id/:product\_id, POST /product, PUT /product/:product\_id
* GET /shop/available, POST /shop, GET /shop

The client is developed using React, with Redux in the background managing the user and application workflow state. React-bootstrap is used for styling the elements and creating a responsive UI.

**Results**

Mocha API testing

Text

Description automatically generated

JMeter Testing – Average Response Time

Chart, line chart

Description automatically generated

Chart, line chart

Description automatically generated

Chart, line chart

Description automatically generated

Chart

Description automatically generated with medium confidence

Chart, line chart

Description automatically generated

**Git history**

A screenshot of a computer

Description automatically generated with medium confidence

**Questions**

1. The efficiency of executing query increases by connection pooling. When connection pooling is not used a thread is created for every request. But when connection pooling is used instead of creating a new thread for every request a pool of thread is maintained which execute the requests. When more requests come to the backend then they are maintained in a queue. Due to this the calls to the backend database are reduced. When pooling is not used the NodeJS server crashes intermittently which lower the performance of the application. When pooling is used it does not crash.
2. Ways to improve SQL performance –  
   1. Creation and use of indexes – An index can reduce the data retrieval time of a record
   2. Good schema definitions – Availability of the right constraint on the right place always helps to improve the query performance, like FORIGEN KEY constraint helps to simplify joins by converting some outer or semi-joins to inner joins and CHECK constraint also helps a bit by removing unnecessary or redundant predicates.
   3. Avoid Running Queries in a Loop – Running queries in a loop can significantly slow down runtime. In some cases, we can make use of bulk inserts and updates to the data, which is more efficient than using loops.