CMPE-273 Lab 1 Splitwise

Yusuf Juzar Soni

Link on YouTube: https://youtu.be/q6yrZbcMcJU

Introduction:

Problem Statement:

Mocking the popular bill splitting application "Splitwise" using a diverse technology stack. Splitwise is a mobile app and web platform that helps users share expenses with others. The technologies used in this project are as follows:

- React JS was used for frontend coding.
- Node was used as to implement the API layer etc.
- My SQL was used to implement the databases.

Functional Requirements/ Goals of the System covered:

- A new user would be able to sign up and will be redirected to his dashboard which shows a summary of his transactions (How much he owes, how much he is owed etc.)
- Existing users can log in and would be redirected to their respective dashboards.
- Form based validations have been implemented to check proper inputs
- The user can see a list of groups he is part of, he can also search within that list of groups if he wishes.
- The left navbar also contains links to the recent activity page where the user can view a history of who has added bills into the group.
- The same navbar also contains a link to the invite list page which displays a list of groups the user has been invited to. The user can accept the invitation, only after accepting the invitation will the group be visible in the users group list.
- The members list also changes based on the invite status.
- A member can create a group by selecting all the users registered in the app.
- A member can settle up the amount he is owed and the amount he owes. (Slightly buggy)
- A basic profile page is visible that gets the data from the database and displays the data stored in the backend.

System Design and Database Overview

- ReactJS makes get, post, put calls to express backend using routes.
- NodeJS receives the requests and performs MySQL queries to update the database.
- Session is assigned to a user when he signs up or logs in.
- MySQL database receives the requests from NodeJS and performs the operations to its tables.
- Backend Sends the response back to React JS to display.

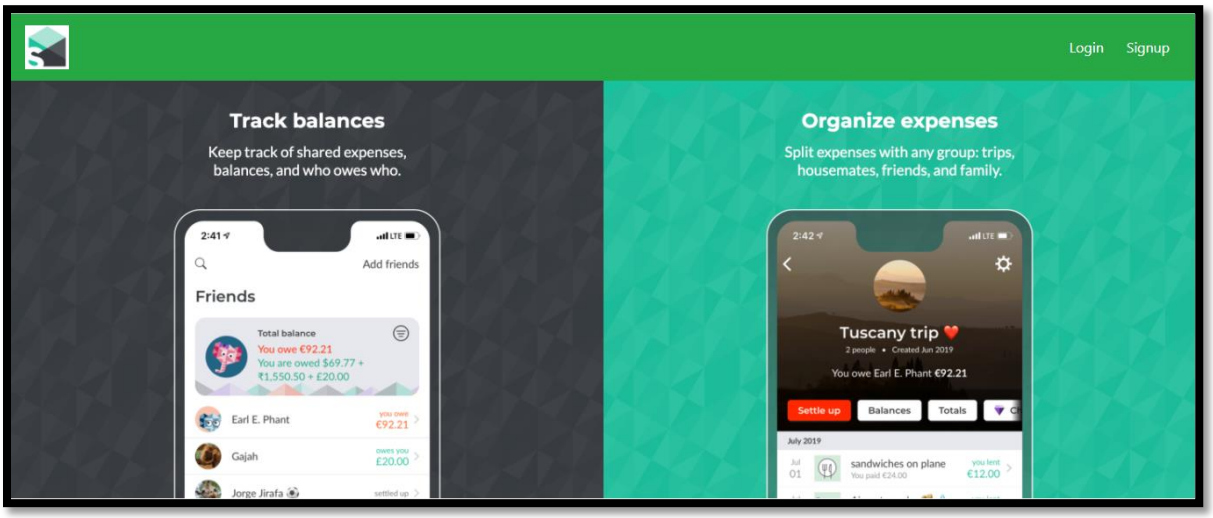
Database Schema:

- users (stores user elated information like username, email etc.) **Email is used as primary key and hence is used in all operations.**
- groups (stores group name, group description ang group photo)
- user_group table (serves as link between user and group table) also stores the users invite status in various groups.
- Bill table (store bill related information, bill id, bill name, amount description etc.) primarily used in displaying recent activity.
- Transcation_table(used to store details of transactions and also keeps record of splits etc.)'

Basic operations using simple SQL Queries are mode to retrieve data as required. Effort has been made to try and keep backend as simple as possible. (also owing to time constraints 😊).

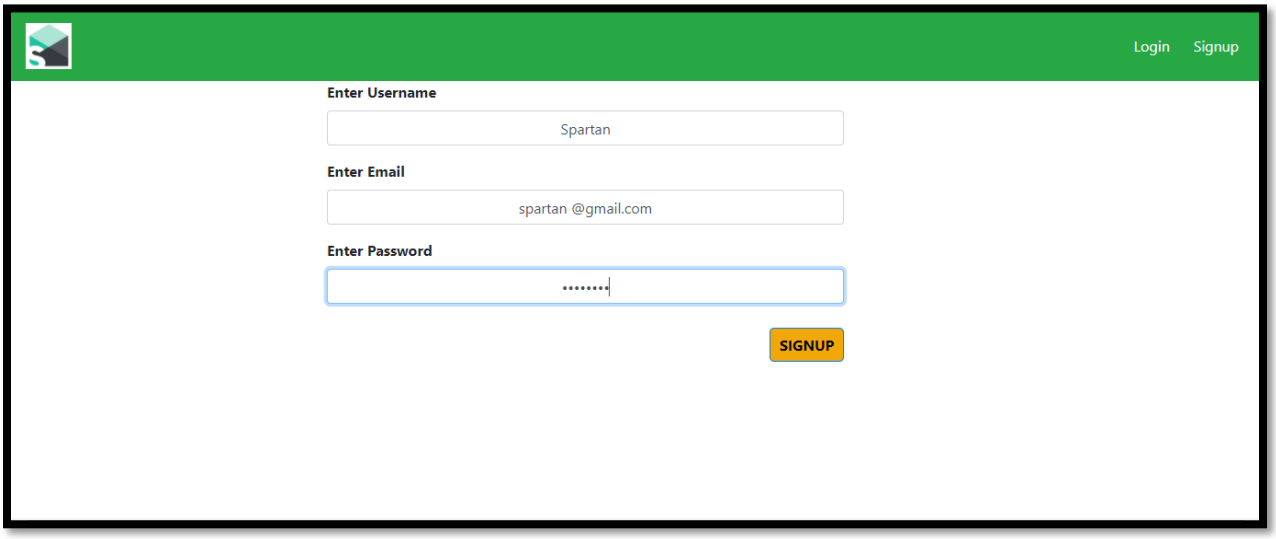
Screenshots of various workflows are added below

Landing Page:

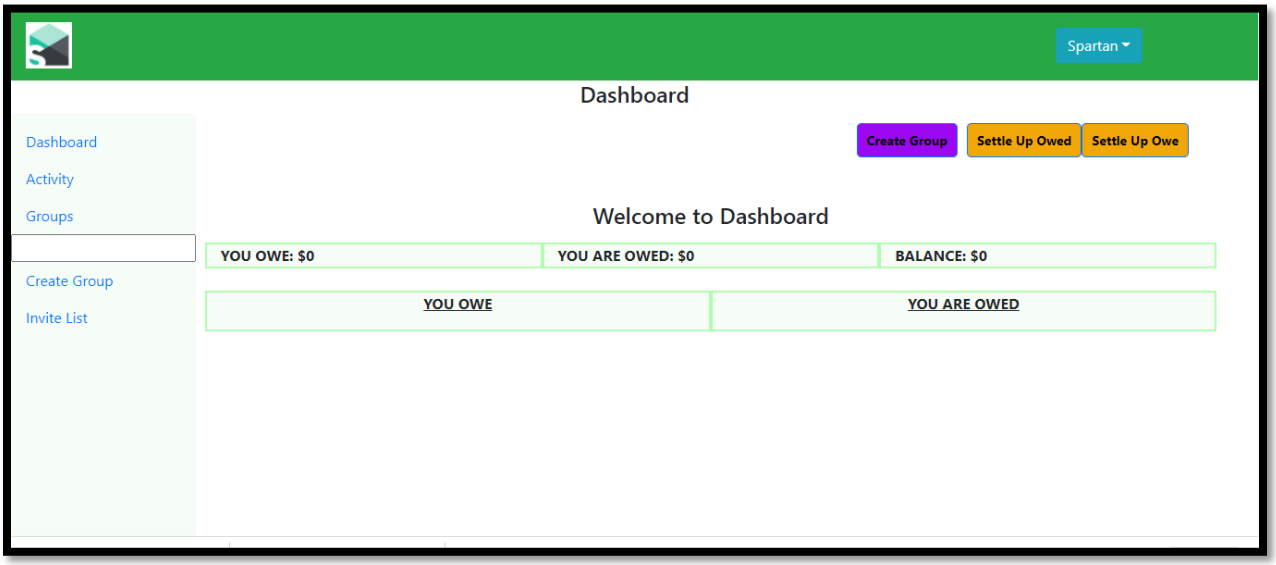


\

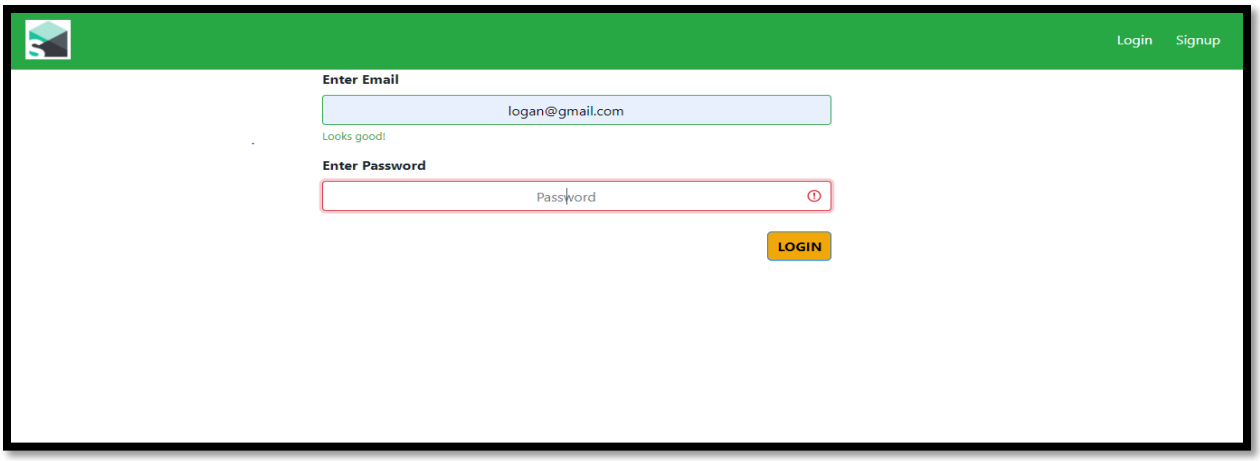
Signup Page:



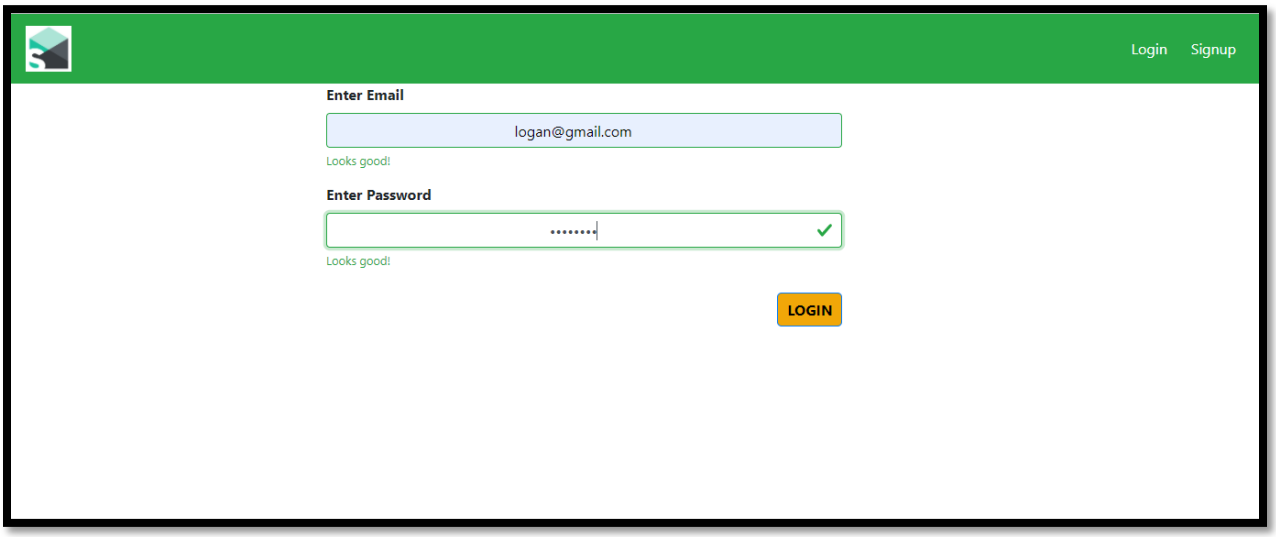
Dashboard After Signup:



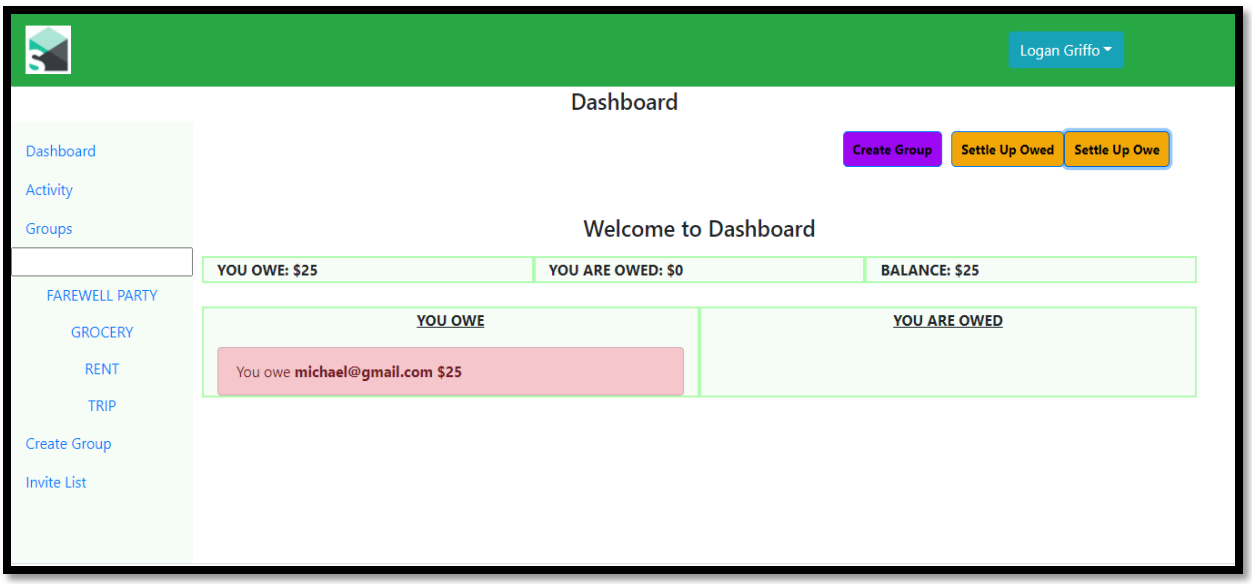
Form Based Validations



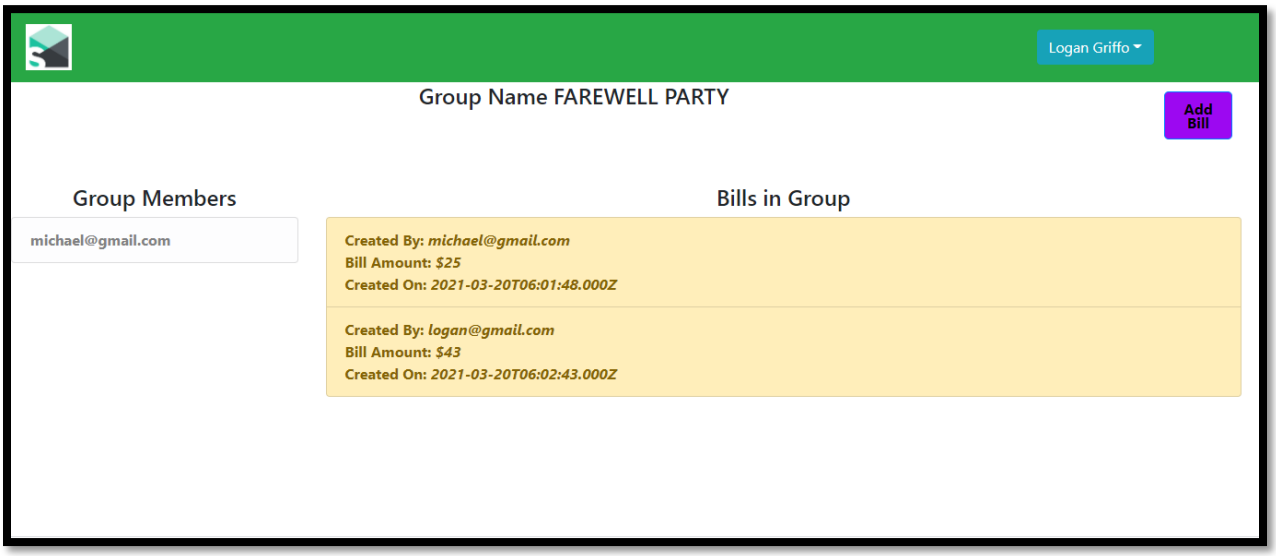
Login of Existing Users



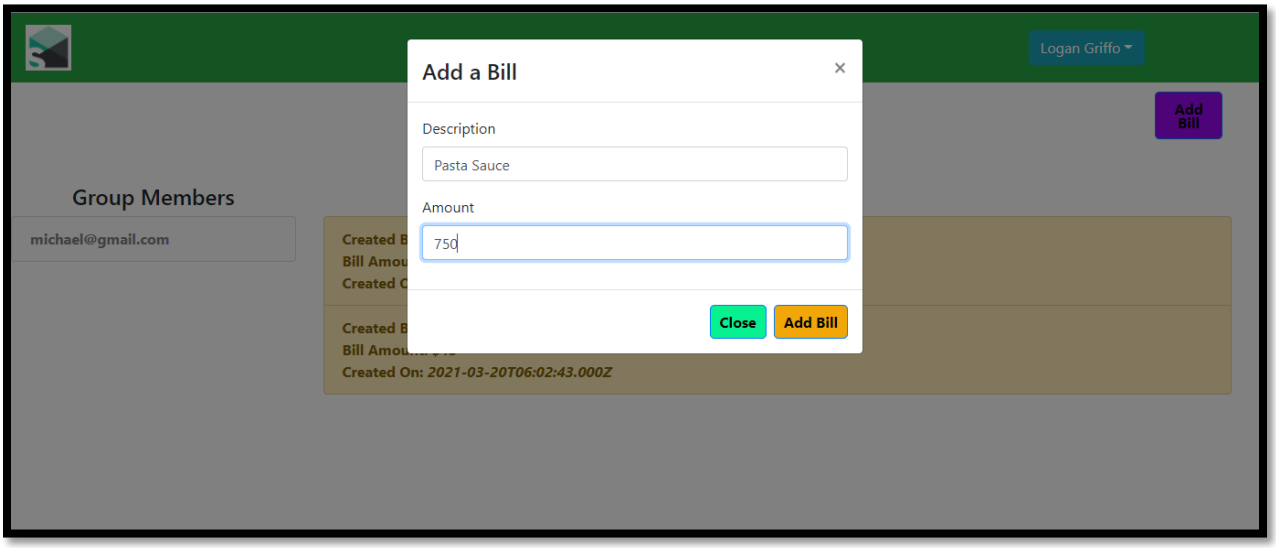
Dashboard after existing user logs in:



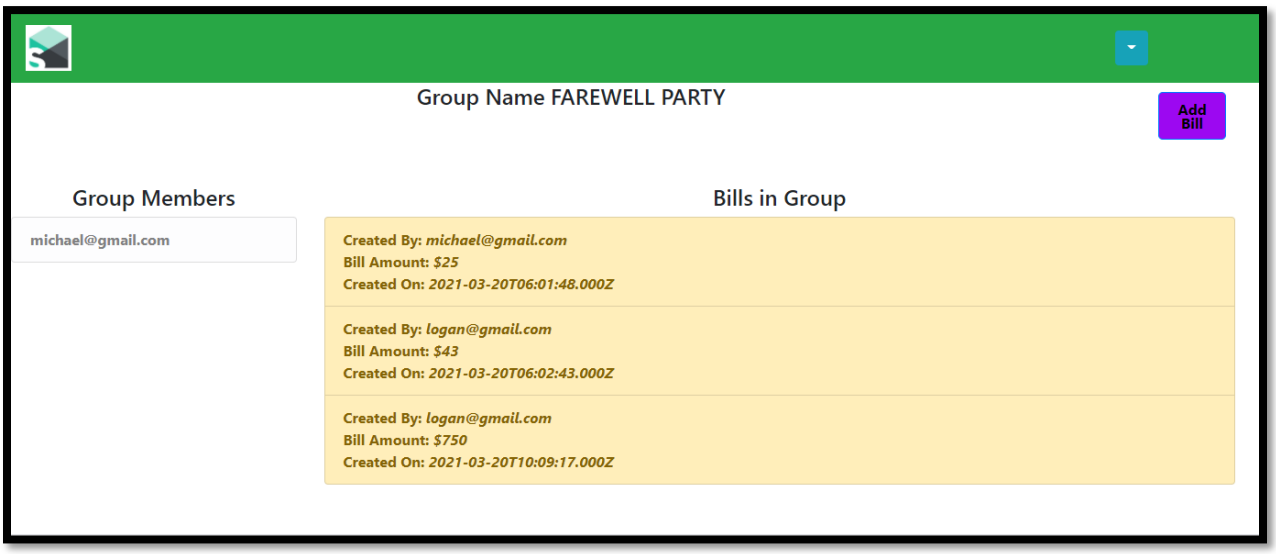
Group Details Page



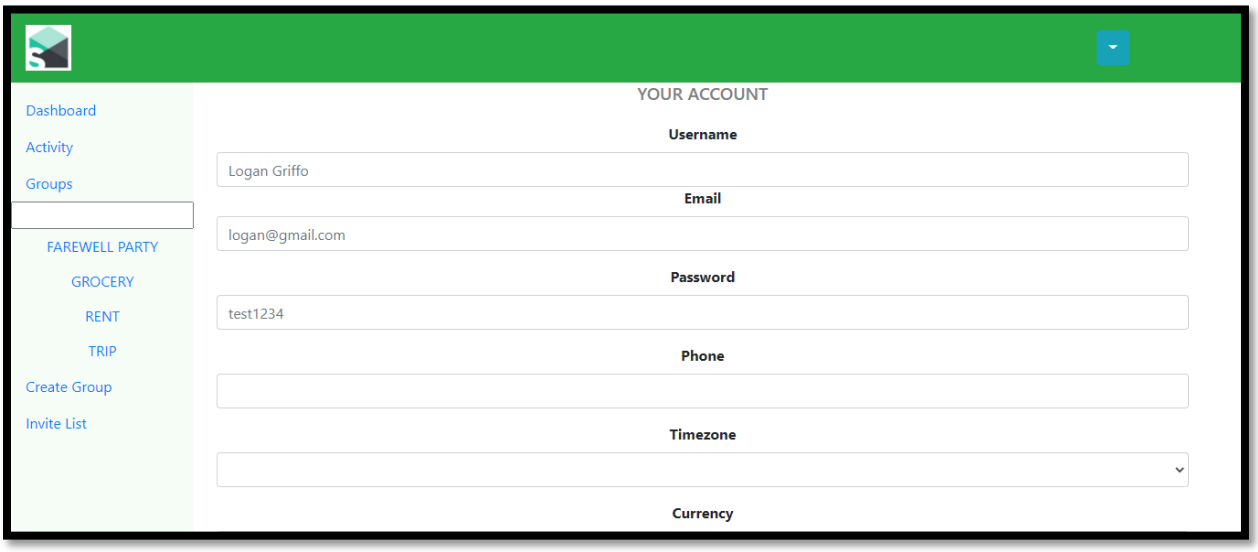
Adding a Bill



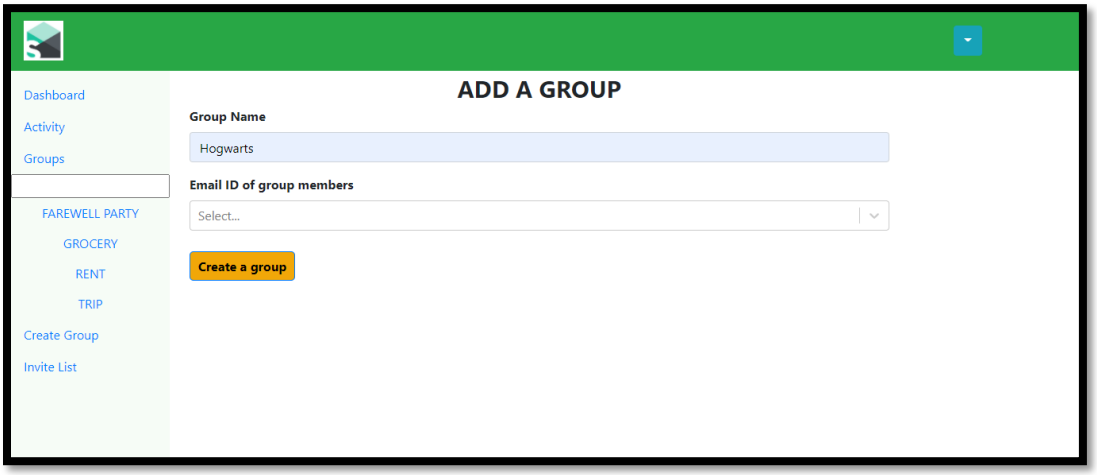
Displaying added Bill



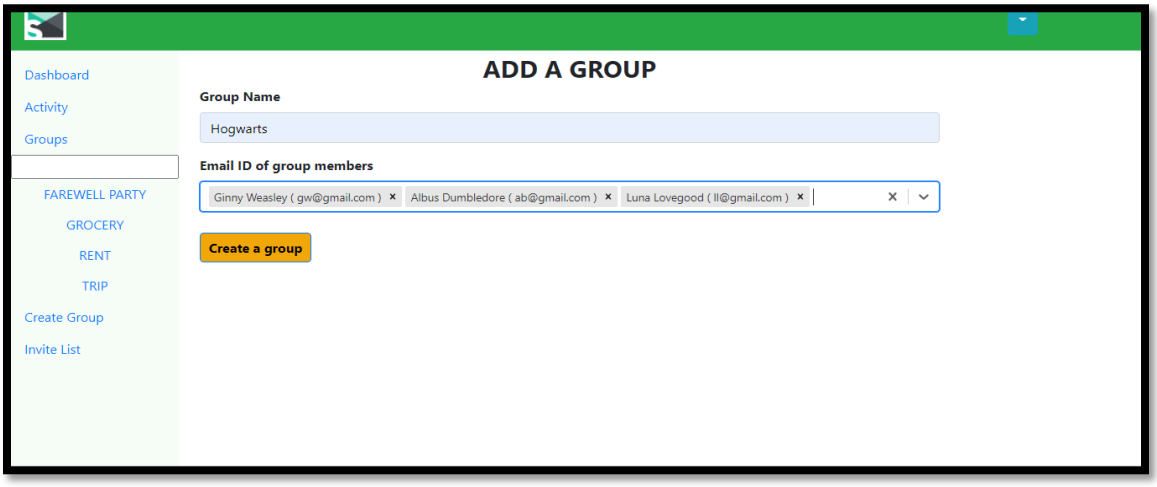
Basic Profile Page:



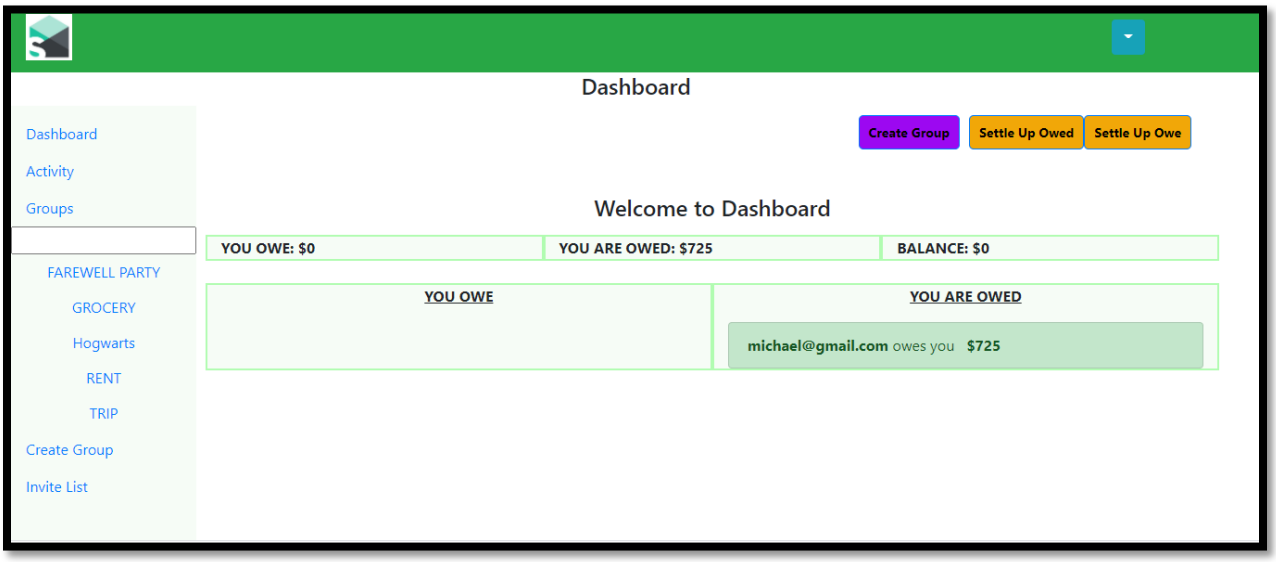
Add a group page:



Adding Members:

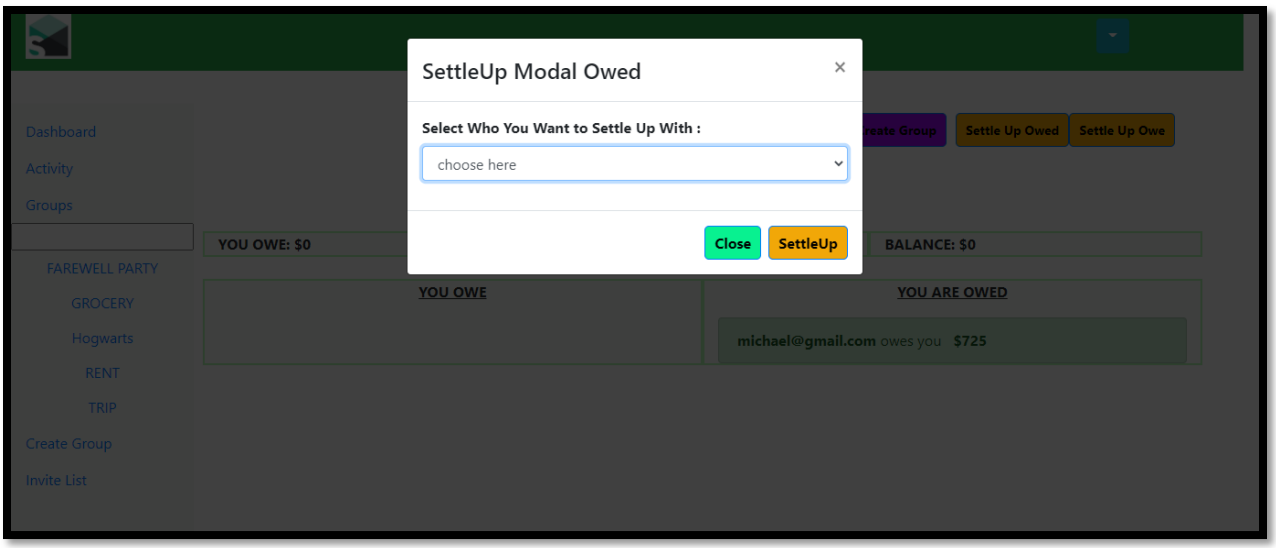


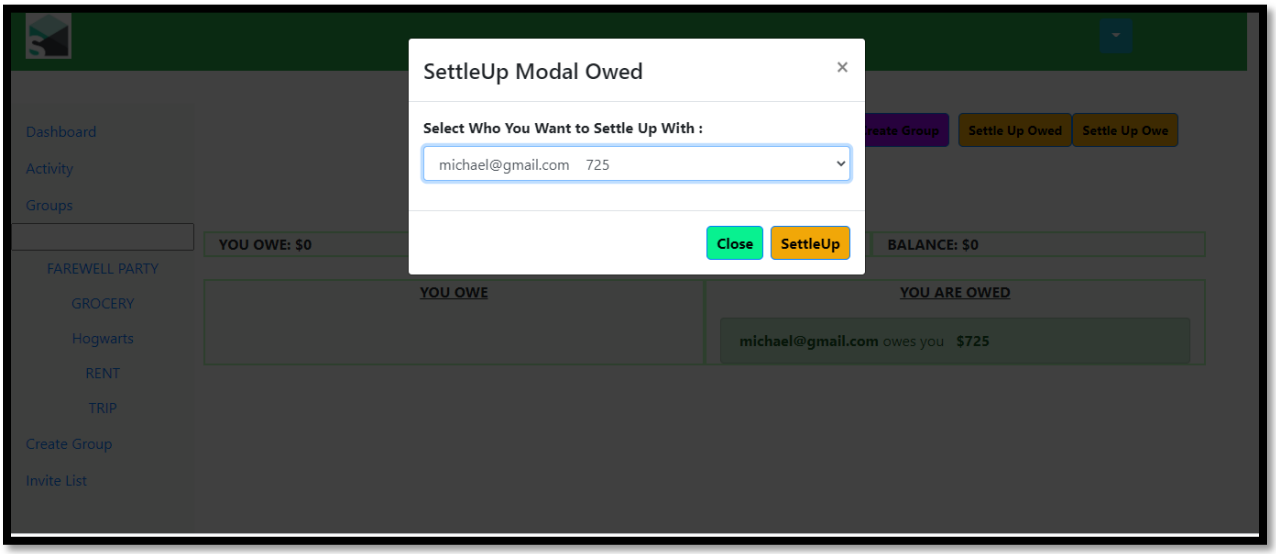
Displaying Added Bill:



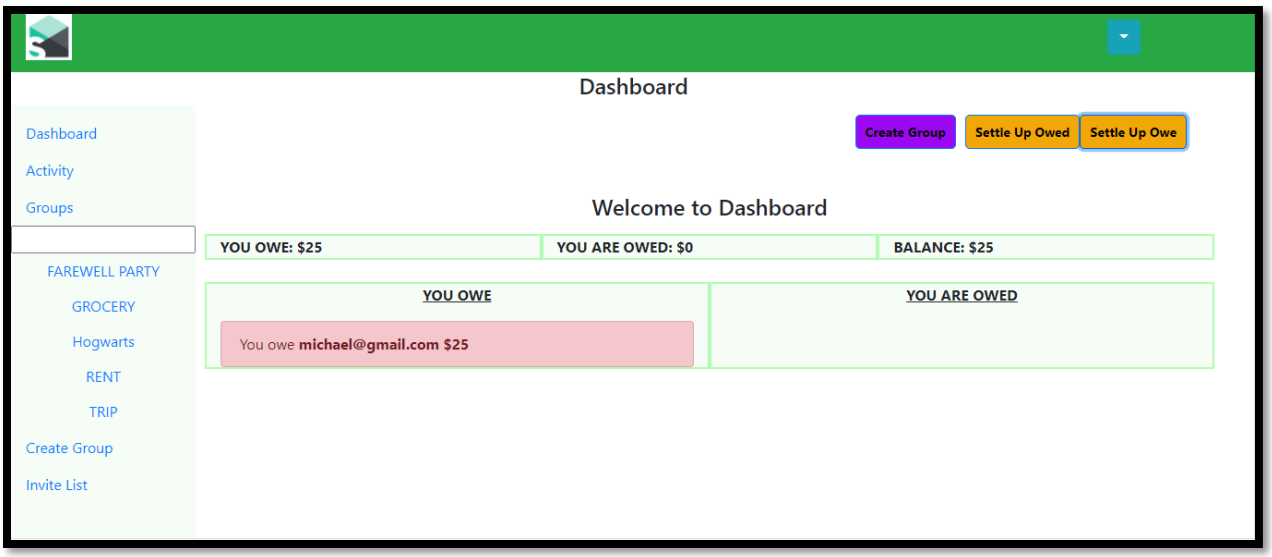
Recent Activity Page:

Settle Up Modal:

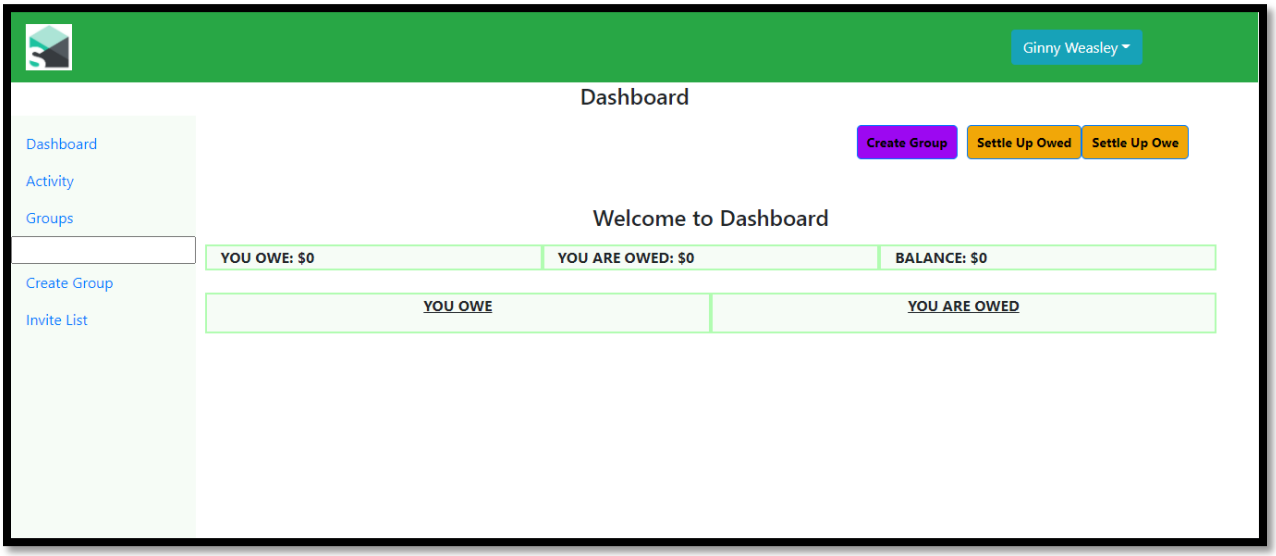




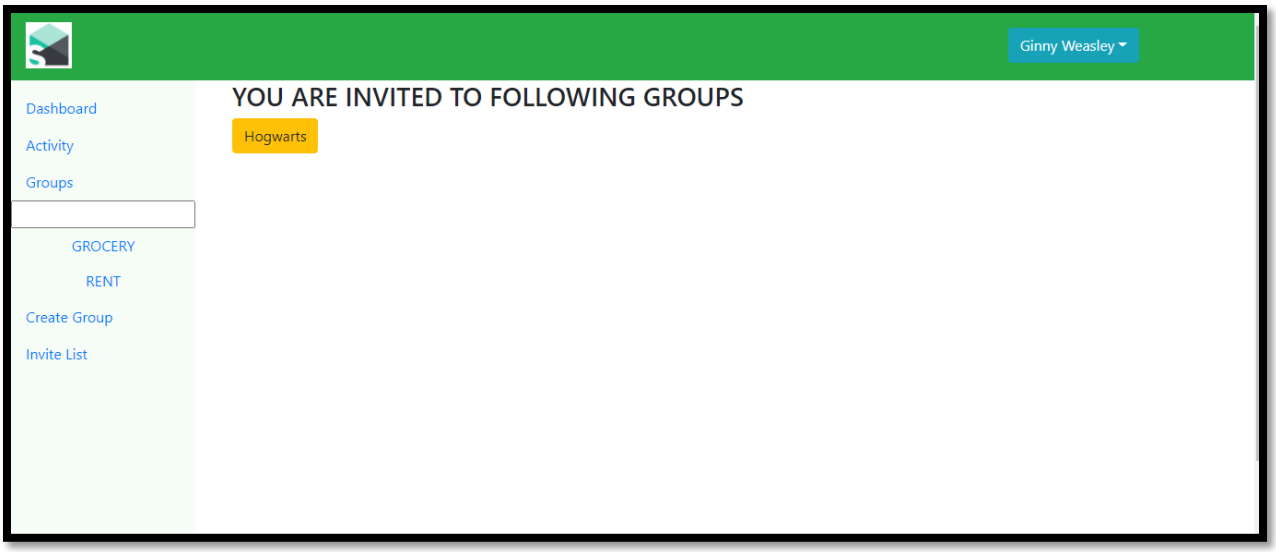
Settling Up of Transaction (Buggy Code Fault in code):

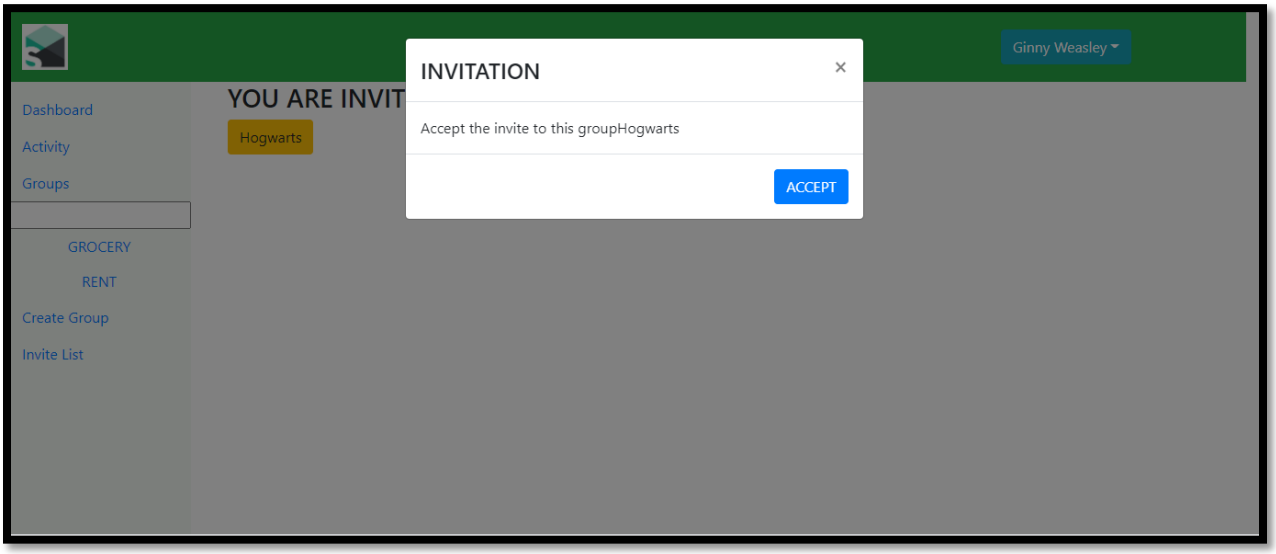


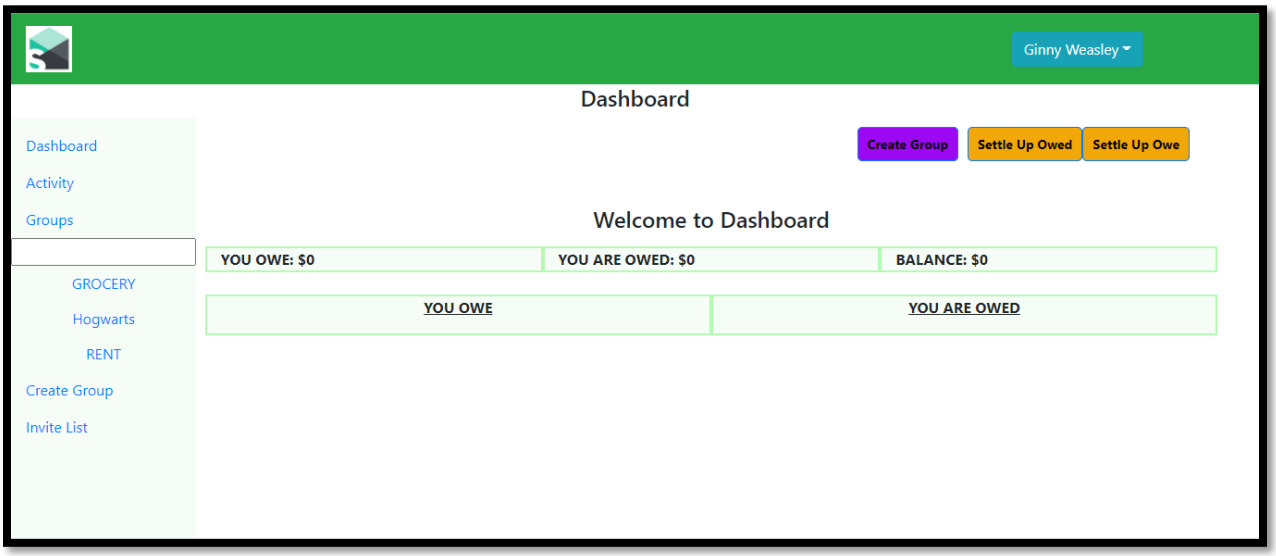
Demonstrating Accept invite :



Modal opens up and the user can choose to accept the invitation. Only then will the name be displayed in the groups list of the user, demonstrated in below screenshots.


Ginny Weasley
Dashboard
Activity
Groups
GROCERY
RENT
Create Group
Invite List
YOU ARE INVITED TO FOLLOWING GROUPS
Hogwarts


Ginny Weasley
Dashboard
Activity
Groups
GROCERY
RENT
Create Group
Invite List
YOU ARE INVIT
Hogwarts
INVITATION
Accept the invite to this groupHogwarts
ACCEPT


Ginny Weasley
Dashboard
Dashboard
Activity
Groups
GROCERY
Hogwarts
RENT
Create Group
Invite List
Create Group
Settle Up Owed
Settle Up Owe
Welcome to Dashboard
YOU OWE: $0
YOU ARE OWED: $0
BALANCE: $0
YOU OWE
YOU ARE OWED

SAMPLE SCREEN SHOTS OF DATABASE TABLES (user_group,transaction_table,bill_table)

user_email	group_name	invite_status
gw@gmail.com	RENT	1
hp@gmail.com	TRIP	0
logan@gmail.com	FAREWELL PARTY	1
logan@gmail.com	GROCERY	1
logan@gmail.com	RENT	1
logan@gmail.com	TRIP	1
michael@gmail.com	FAREWELL PARTY	1

28	mj@gmail.com	logan@gmail.com	25	2021-03-19 22:29:11	FAREWELL PARTY
29	logan@gmail.com	mj@gmail.com	0	2021-03-19 22:29:54	FAREWELL PARTY
30	michael@gmail.com	logan@gmail.com	25	2021-03-19 23:01:48	FAREWELL PARTY
31	logan@gmail.com	michael@gmail.com	0	2021-03-19 23:02:43	FAREWELL PARTY
32	logan@gmail.com	michael@gmail.com	0	2021-03-20 03:09:17	FAREWELL PARTY
NULL	NULL	NULL	NULL	NULL	NULL

bill_id	bill_amount	bill_desc	bill_timestamp	created_by	split_amount	bill_group
45	25	Food Arrangement	2021-03-19 23:01:48	michael@gmail.com	25	FAREWELL PARTY
46	43	Gifts	2021-03-19 23:02:43	logan@gmail.com	43	FAREWELL PARTY
47	750	Pasta Sauce	2021-03-20 03:09:17	logan@gmail.com	750	FAREWELL PARTY
NULL	NULL	NULL	NULL	NULL	NULL	NULL

TESTING:

1: Frontend Testing using React Testing Library (Summary of Tests)

```
PASS src/Components/Landing/Landing.test.js

Snapshot Summary
 › 2 snapshots written from 2 test suites.

Test Suites: 5 passed, 5 total
Tests:       10 passed, 10 total
Snapshots:   2 written, 3 passed, 5 total
Time:        5.724 s, estimated 14 s
Ran all test suites related to changed files.

Watch Usage
 › Press a to run all tests.
 › Press f to run only failed tests.
 › Press q to quit watch mode.
 › Press p to filter by a filename regex pattern.
 › Press t to filter by a test name regex pattern.
 › Press Enter to trigger a test run.
```

2: Mocha and Chai Backend Testing:

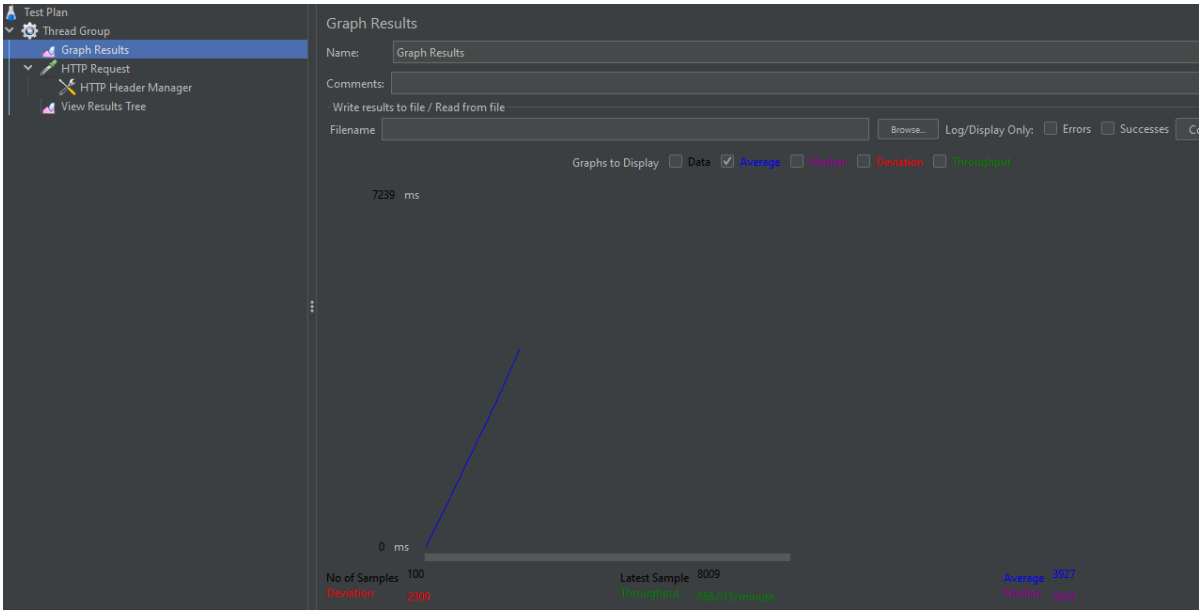
```
  Splitwise
    Login Test
      √ Incorrect Password
    Signup
      √ Signup
{
  name: 'Logan Griffo',
  email: 'logan@gmailmail.com',
  password: 'test1234'
}
    Display groups
      √ Groups
    Get invites
      √ Get Invites
    Get Activity
      √ Name for Dashboard


[]
[]
User already present!!
```

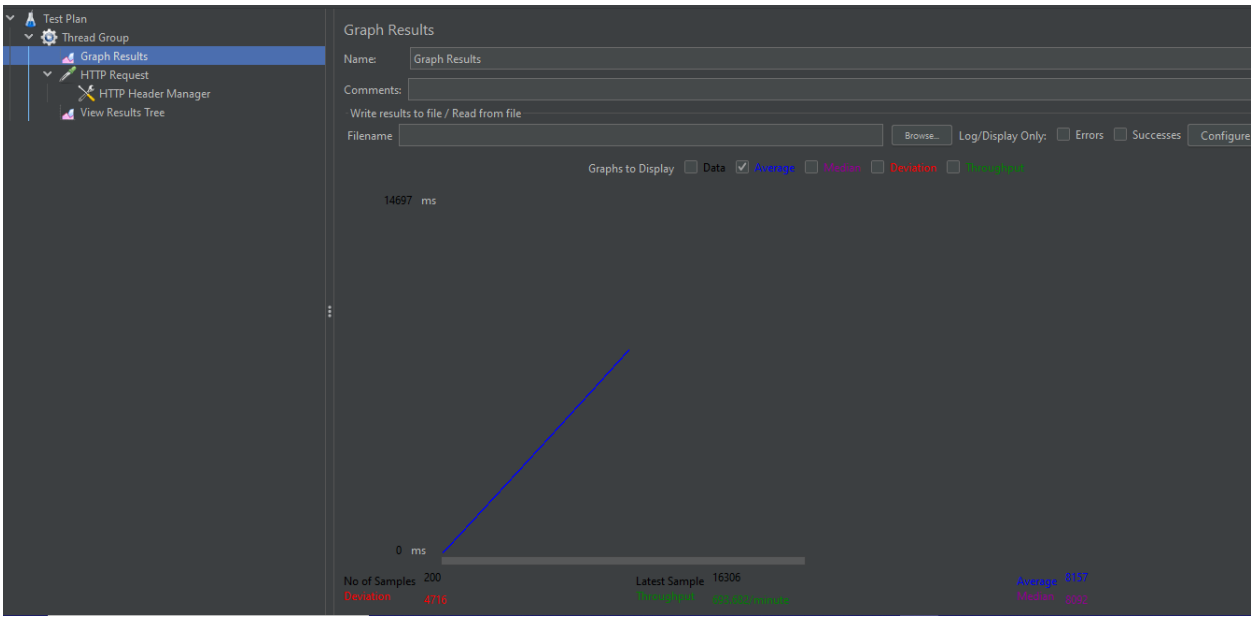
3: JMeter testing.

<u>BEFORE POOLING:</u>

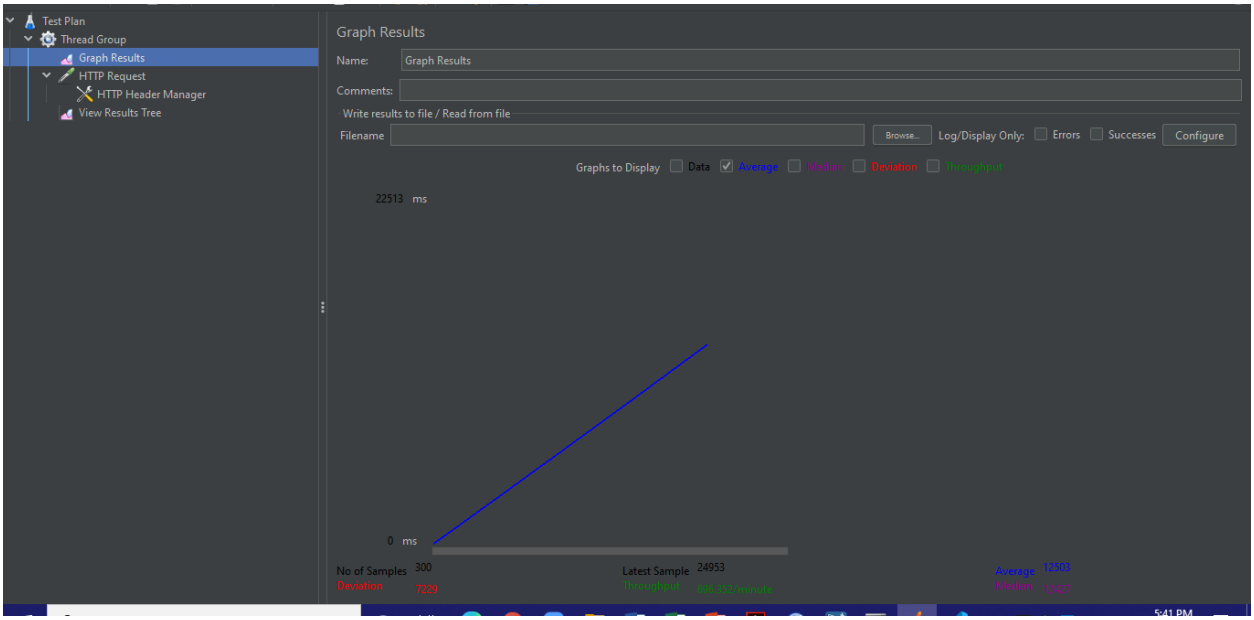
100 Requests



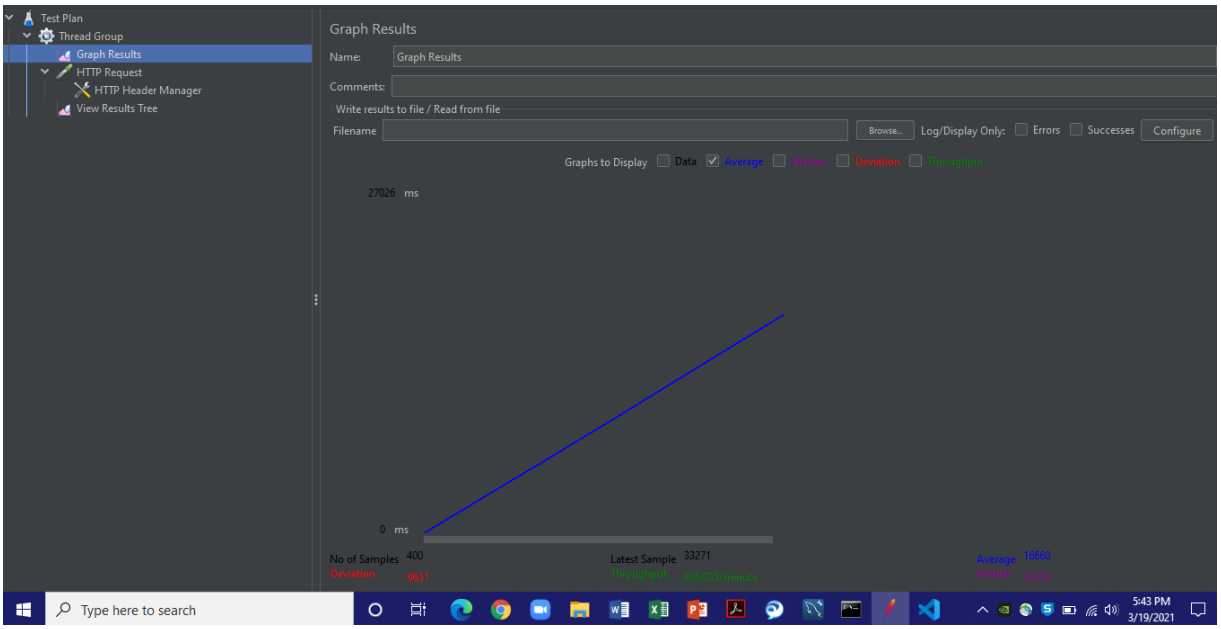
2: 200 Requests



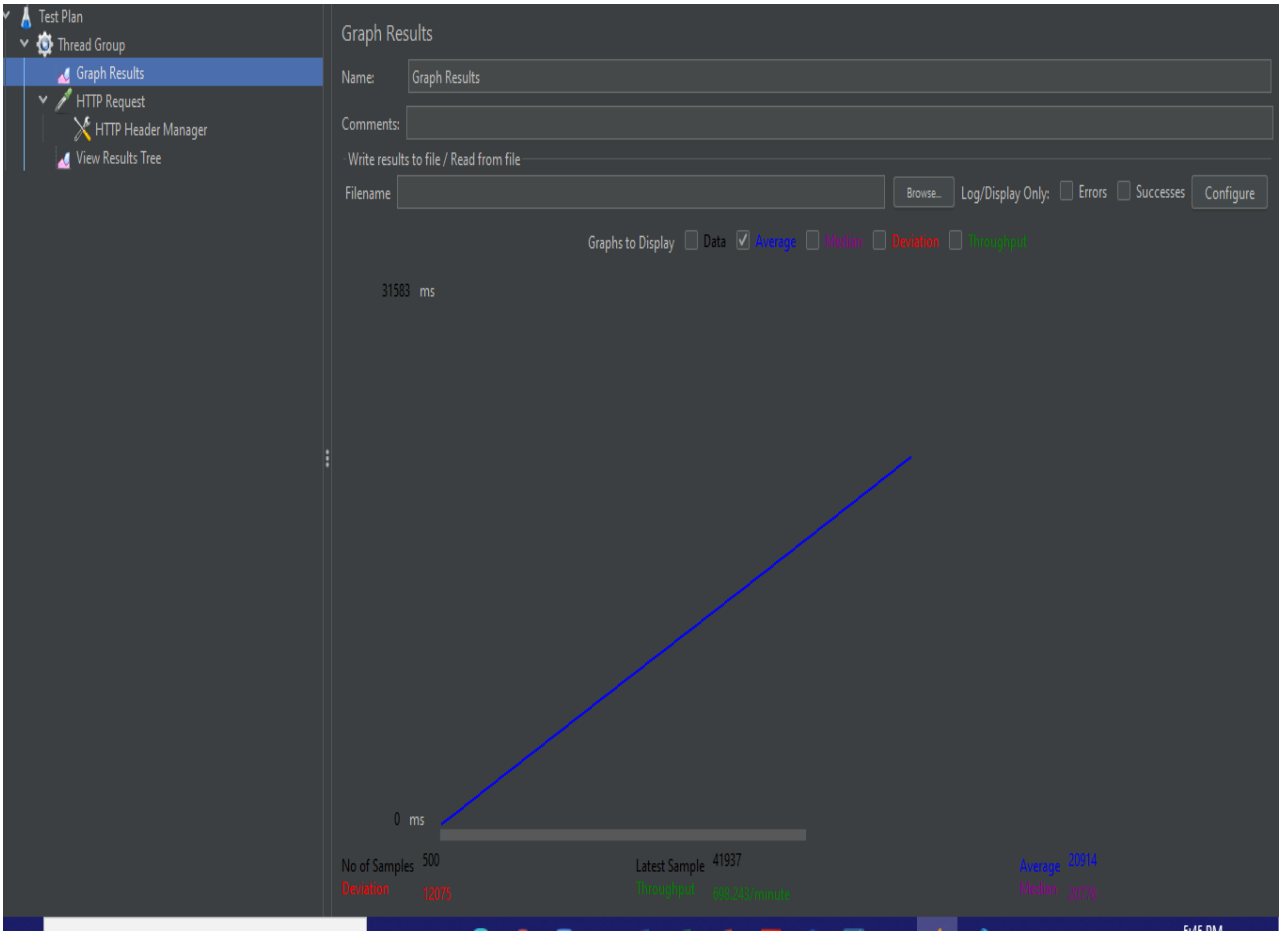
3: 300 Requests



4: 400 Requests



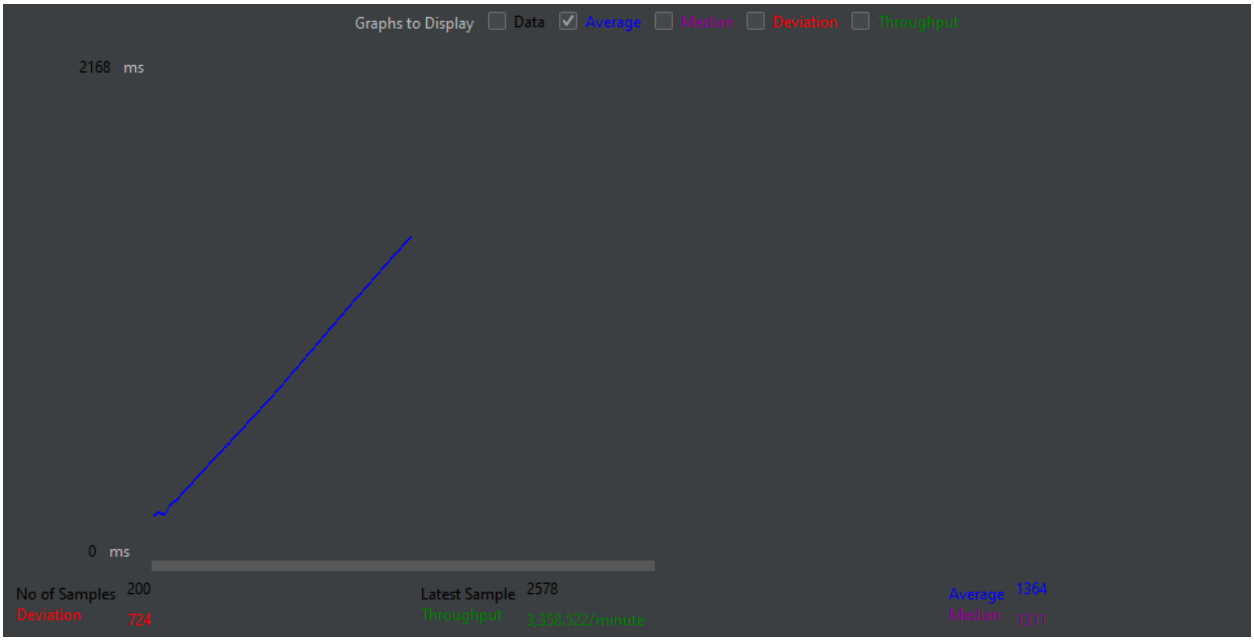
5: 500 Requests



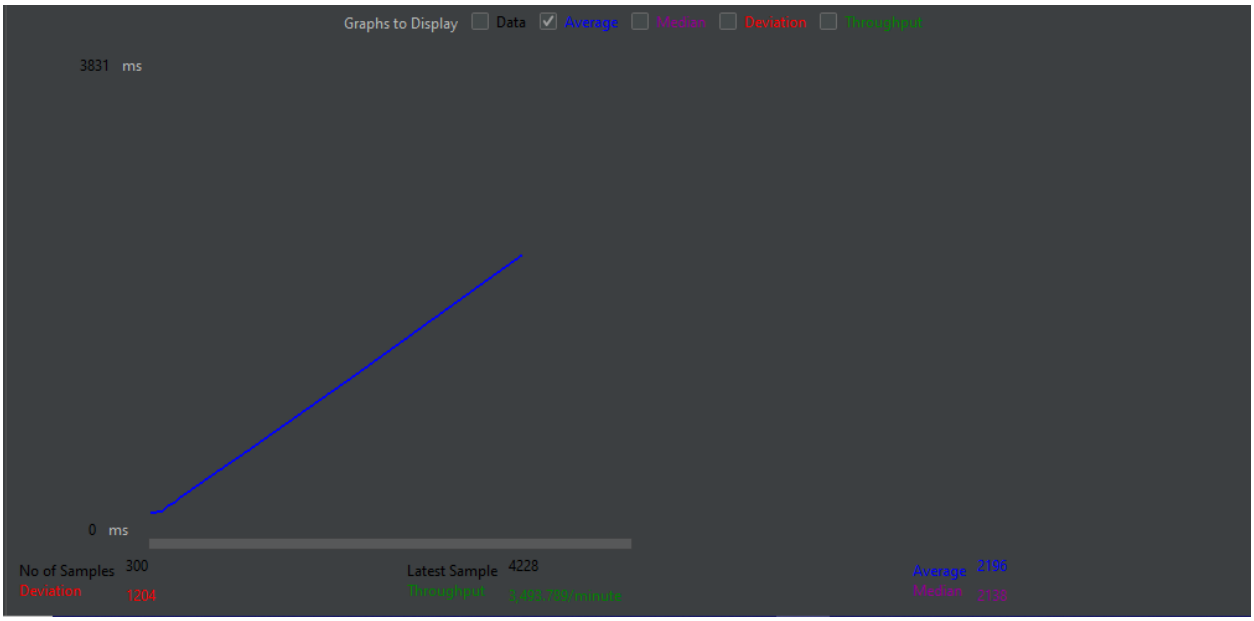
AFTER POOLING:

1:100 Requests

2:200 Requests

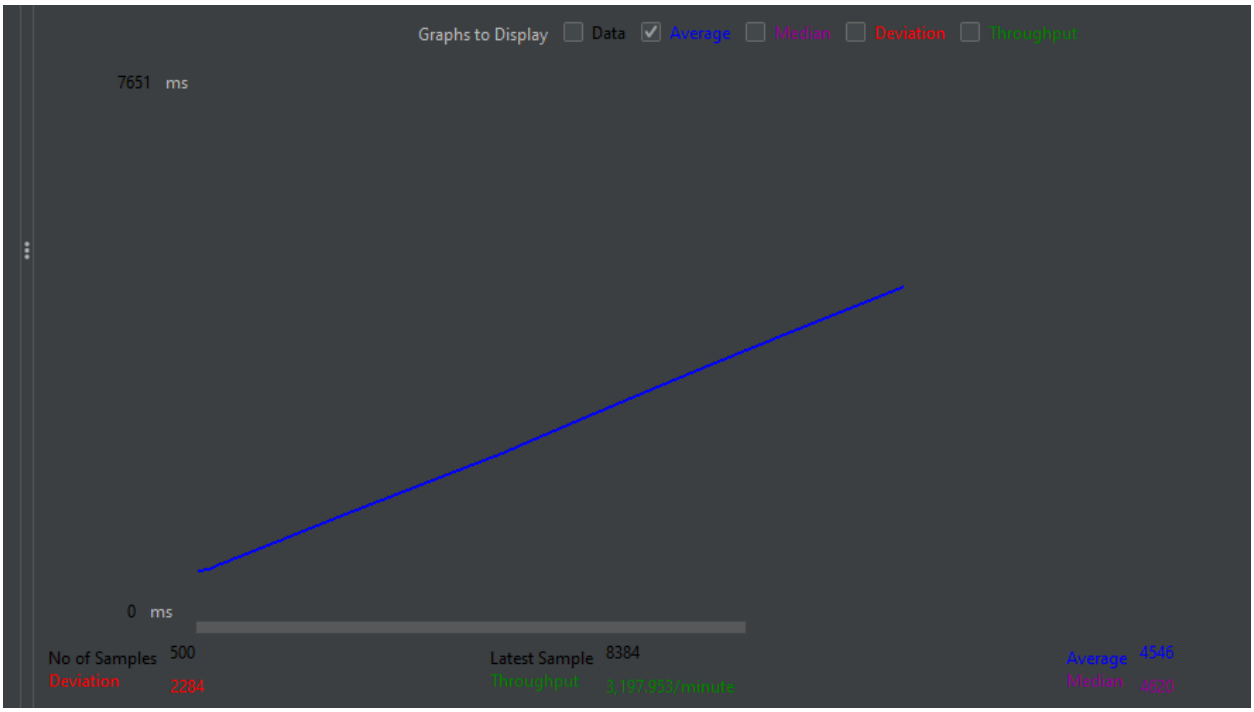


3: 300 Requests



4: 400 Requests

5:500 Requests



QUESTIONS:

1. Compare the results of graphs with and without in-built mysql connection pooling of database. Explain the result in detail and describe the connection pooling algorithm if you need to implement connection pooling on your own.

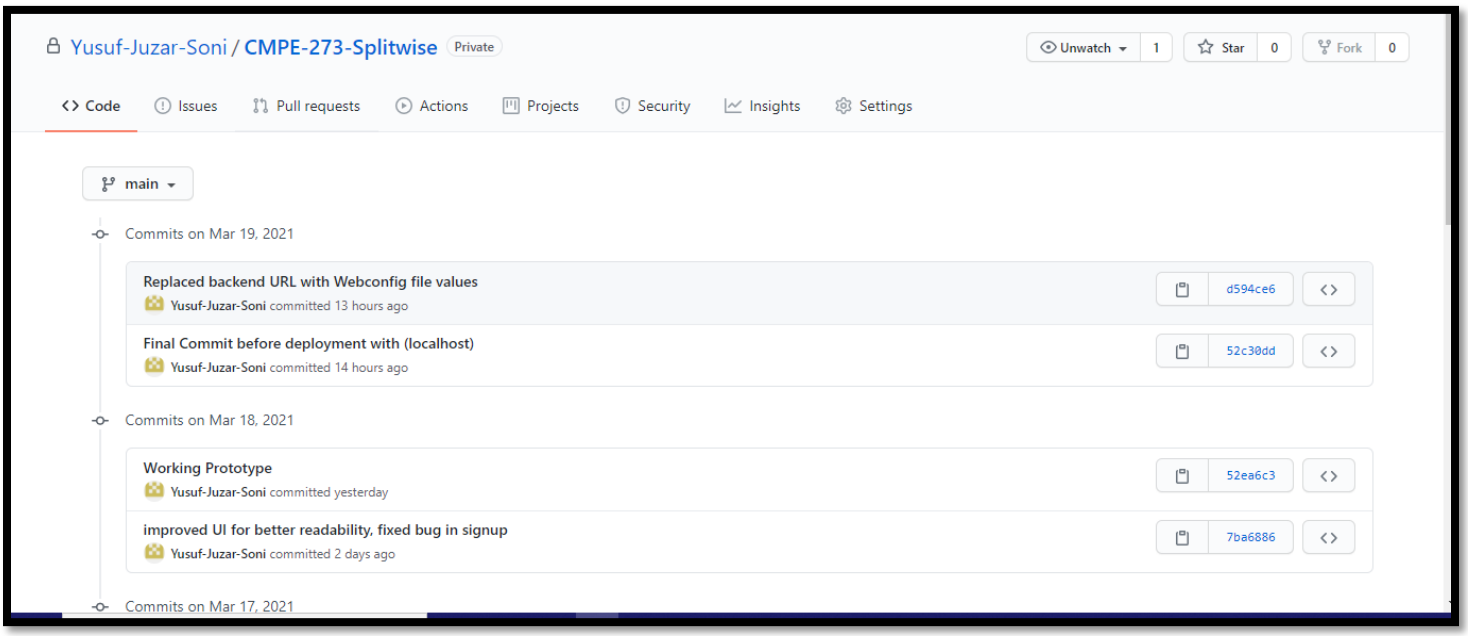
The most common methods to handle a large number of requests to the database is caching. However, we can also rely on connection pooling to improve the database performance in large scale systems.
Clients need to connect to a web service to perform basic CRUD operations. This connection can sometimes take up to 1.3 MB per connection. In a real-world production environment, where millions of queries are being fired to the database, response time drastically reduces.
Instead of opening and closing connections for every request, connection pooling uses a cache of database connections that can be reused when future requests to the database are required. This helps in scaling the database as traffic grows. As traffic is variable, so pooling can better manage traffic peaks without causing outage, thus avoiding bottlenecks.
It is evident from the above graphs that connection pooling improves throughput by quite a significant amount. This is indicative of increased performance.
Any data structure/system that is implemented base on a FIFO approach can be used to implement connection pooling. Let us take thread pooling as an analogy, say there are 15 threads in the thread pool, that are all currently running tasks. We need to maintain a queue of tasks in a first in first out fashion, that will be picked up by the thread in the pool once it completes its own task. As there are a number of threads/connections already created, no overhead is spent in establishing or destroying a connection.

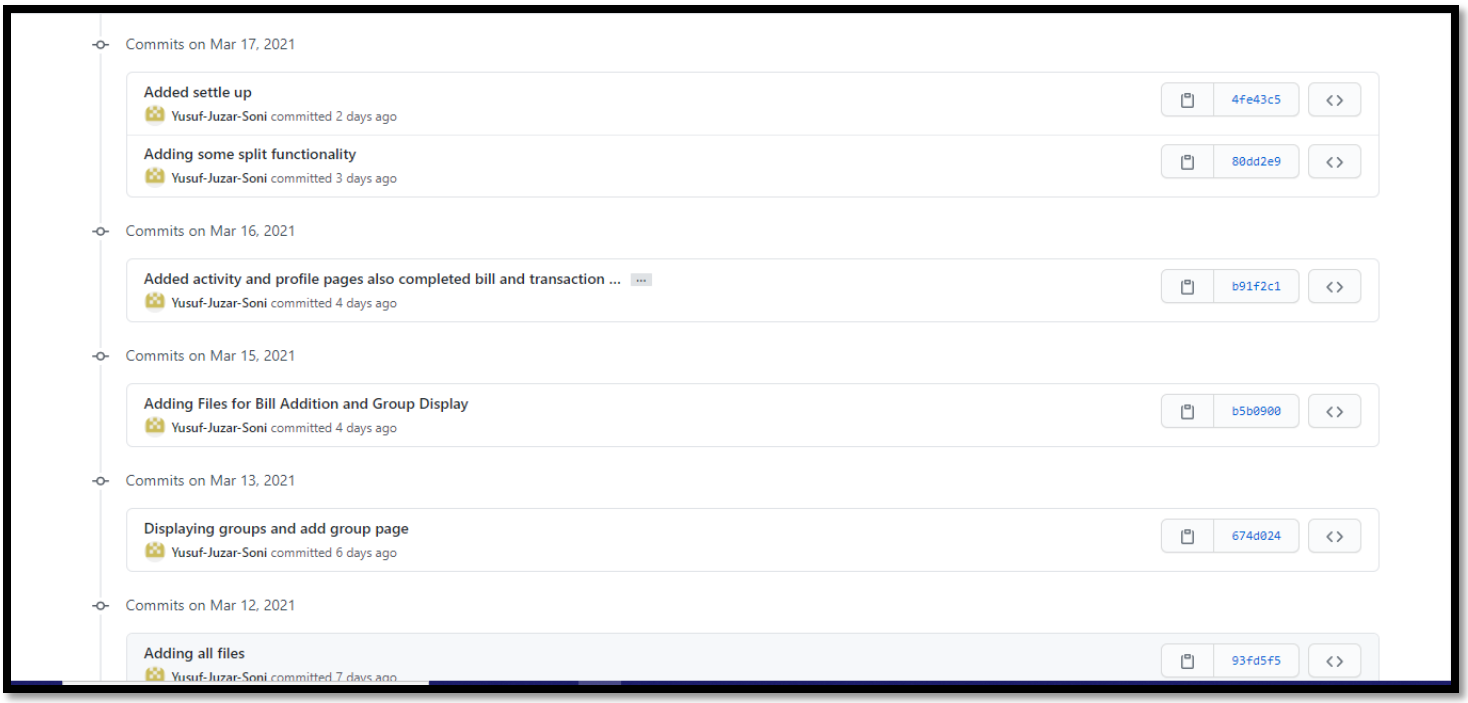
2. What are the ways to improve SQL Performance? List at least 3 strategies. Explain why those strategies improve the performance. (2 pts)

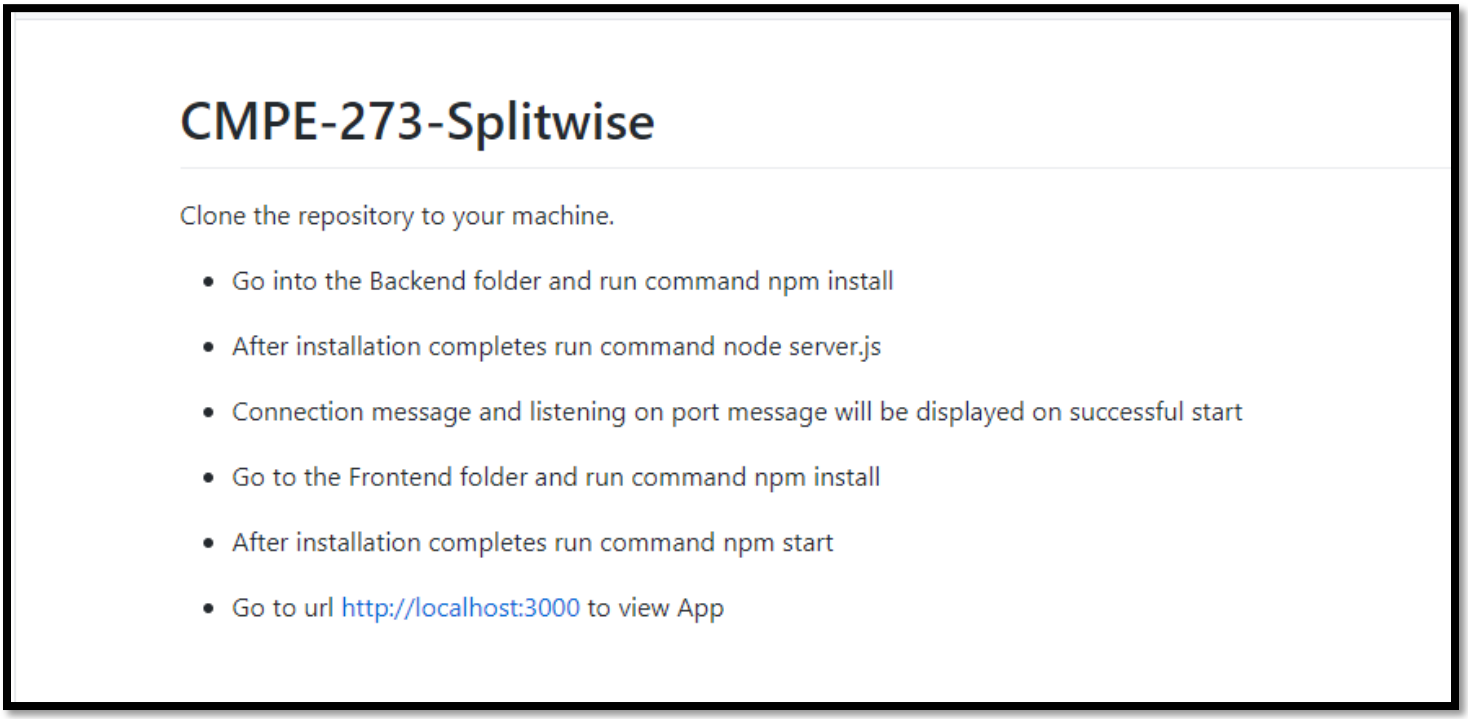
SQL Performance is greatly dependent on the way one writes/structures one's queries. The entire query optimization subspace within databases deals with this aspect of SQL performance. A few strategies can be employed to improve SQL performance.

- **Using actual column names in the SQL query instead of selecting all columns (using SELECT *) FROM a table, so that only necessary columns are selected. This, although trivial greatly reduces the data that needs to be fetched and evaluated every time the query is fired.**
- **If we are running a query that involves scanning a table, we must implement indexing on the desired attribute through which the table is accessed. Indexing the table helps to locate a row quickly. This however should be used with caution. This is best only for large tables.**
- **A good query is one that can pull only the required records from the database. While writing a query one must keep the order of operations in SQL in mind, For E.g. HAVING statements are evaluated after WHERE statements. If the intent is to filter a query based on conditions, a WHERE statement is more efficient, as per order of execution.**

GIT COMMIT HISTORY







==========XXXXXXXXXXXXXXXXXXXXXX-==========