CMPE:273 LAB-2 SPLITWISE

INTRODUCTION:

- The purpose of this lab is to build a Splitwise clone.
- Splitwise is a free application used to track and share expenses between friends.
- The application has a single user persona.
- Every User is provided with a Dashboard to monitor the expenses.
- User can create a group

Youtube Link:https://www.youtube.com/watch?v=1ybxveN3Qr8

SYSTEM DESIGN:

- **Database:**
- **Mongo database is used:**
- **Mongoose is used for storage.**
- **User**

  UserId:EmailId of User-Primary Key-
  UserName:Name of the user.can be changes
  Profile:Link to the AWS S3 store where profile picture is uploaded
  Currency:User Default currency
  TimeZone:Time zone user operates.

- **Group**

  GroupId:GroupId unique;y assigned to each group on group creation.
  GroupName:Name of the Group.
  CreatorEmail:Email Id of the creator.
  Url:Link to the AWS S3 store where profile picture is uploaded.

- **UserGroup(Connecting User and Group Table)**

  GroupUserId:Unique systematically generated for each group-user pair.
  UserId:User's EmailId
  GroupId:Groupid
  GroupName:GroupName
  Flag: Default values 0. Set when the user accepts an invitation.

- **User-User Group**
  - UserId1 owes UserId2.
  - For every group unique set of User1-User2 values.

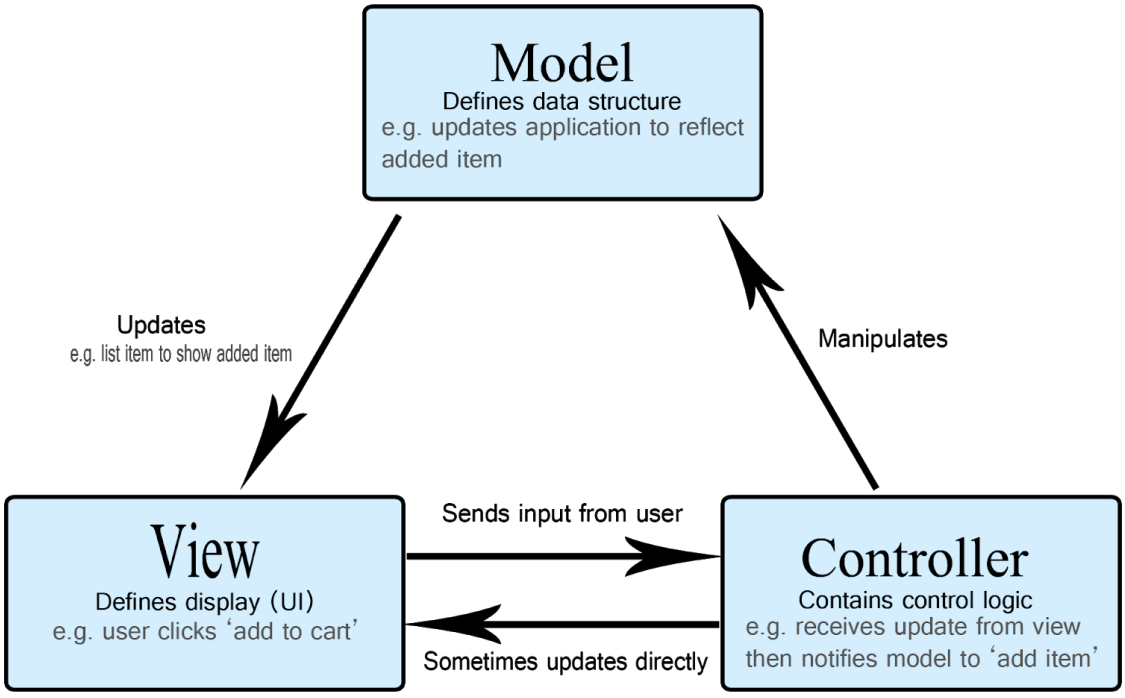
  UUID
  UserId1
  UserId2
  GroupId
  Owes

- **User-Group-Transaction**
    UUID
    EmailId
    Description
    Amount
    GroupId

- **Recent-activity**
    activityId
    OperationType
    GroupId
    GroupName

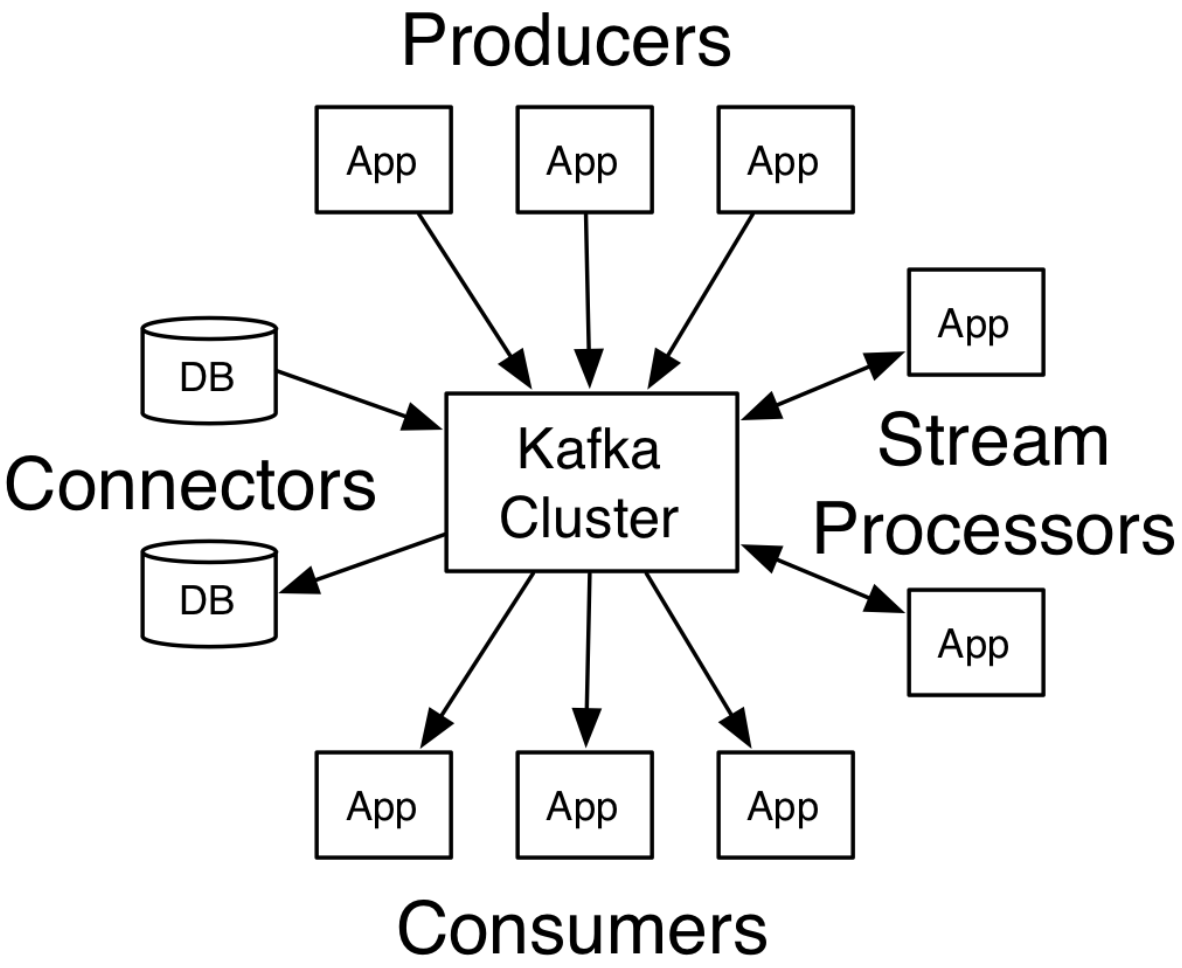
- **ORM(Object Relational Model Mongoose)**
    1. ORM converts a mongo into objects
    2. Mongoose is a promise based node based ORM.

- **Backend Design:**



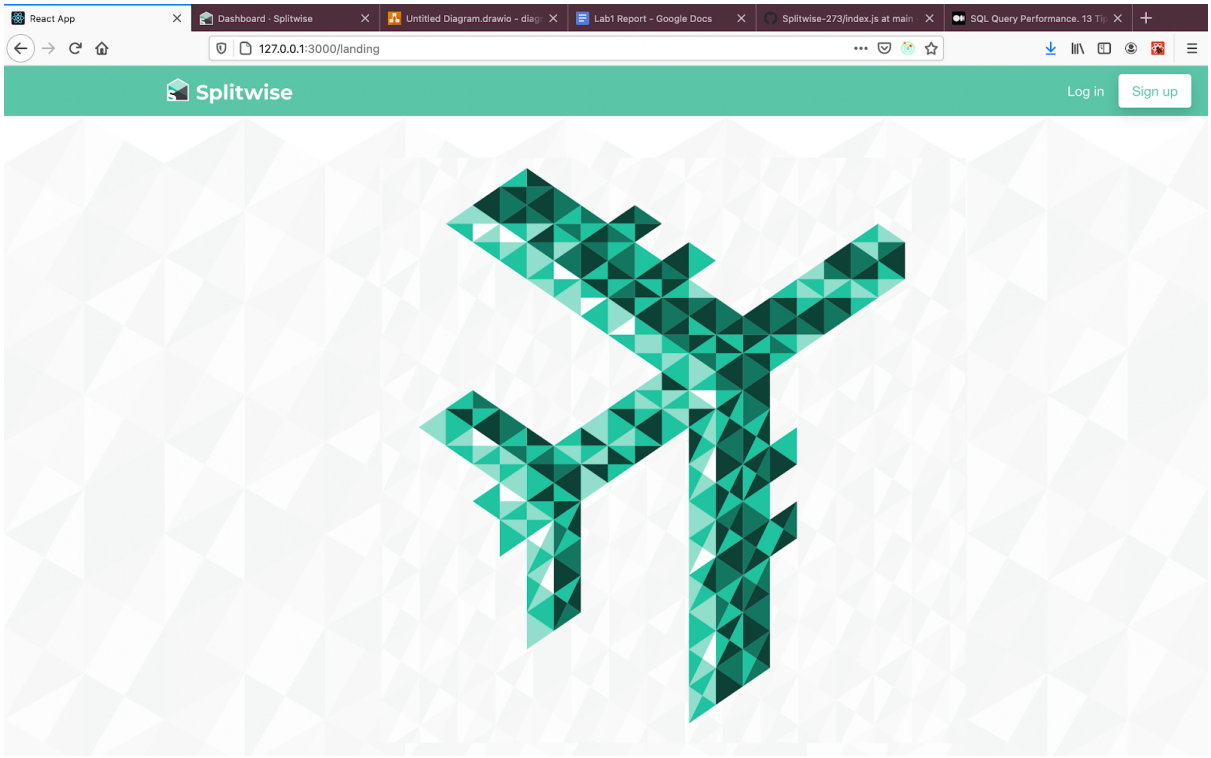
- A popular application architecture pattern MVC(Model Controller View) is used in this project.
- Model accesses the database and serves the data in object format.

KAFKA

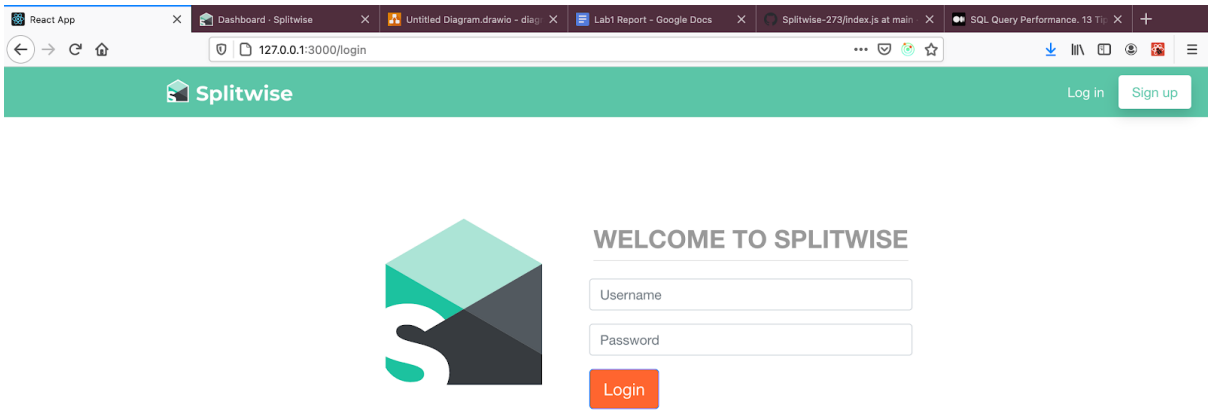


- **Frontend UI:**

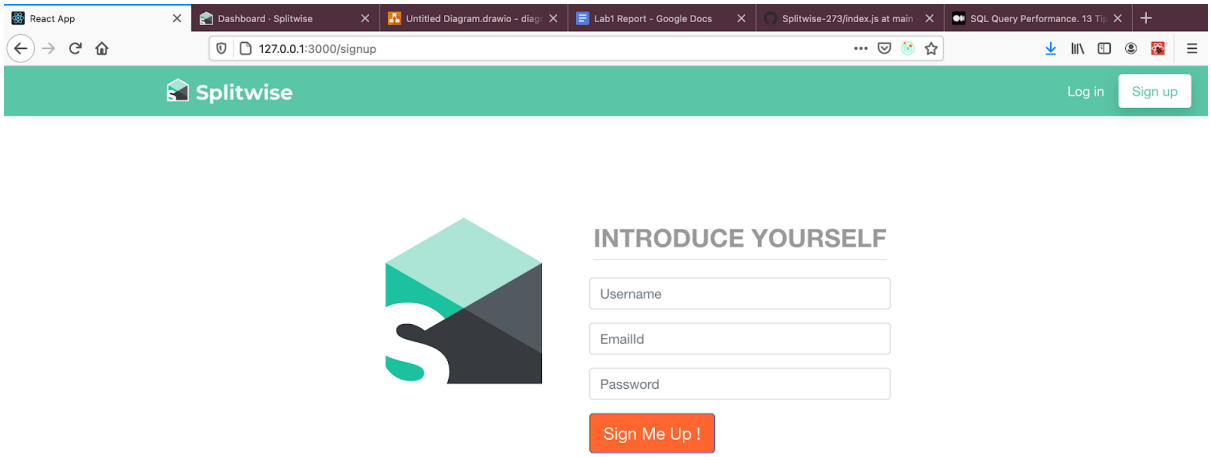
Landing page:


React App
Dashboard - Splitwise
Untitled Diagram.drawio - diag
Lab1 Report - Google Docs
Splitwise-273/index.js at main
SQL Query Performance. 13 Tip
127.0.0.1:3000/landing
Splitwise
Log in
Sign up

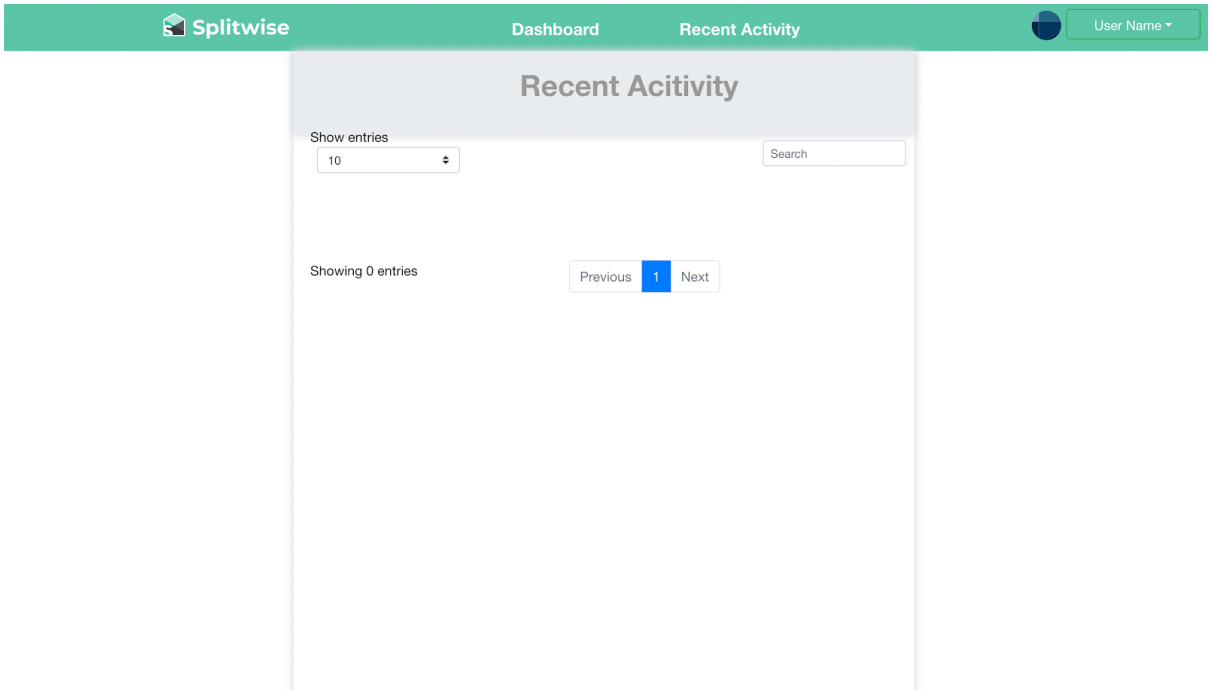
Login:


React App
Dashboard - Splitwise
Untitled Diagram.drawio - diag
Lab1 Report - Google Docs
Splitwise-273/index.js at main
SQL Query Performance. 13 Tip
127.0.0.1:3000/login
Splitwise
Log in
Sign up
WELCOME TO SPLITWISE
Username
Password
Login

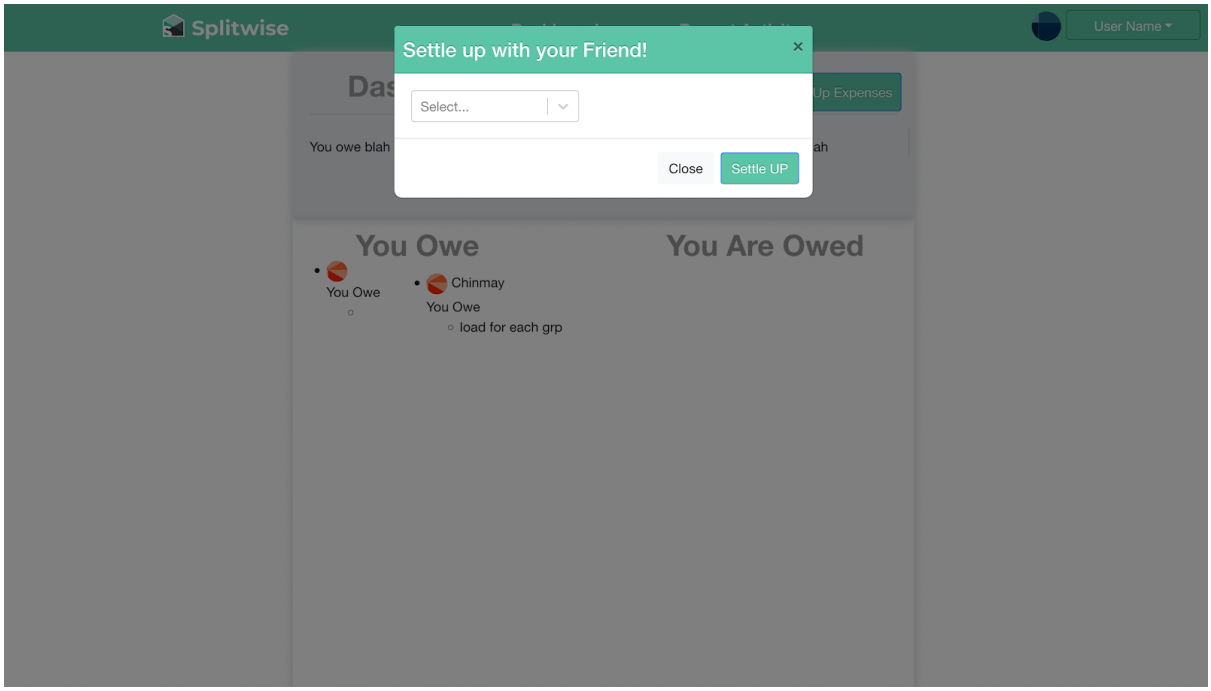
Register:



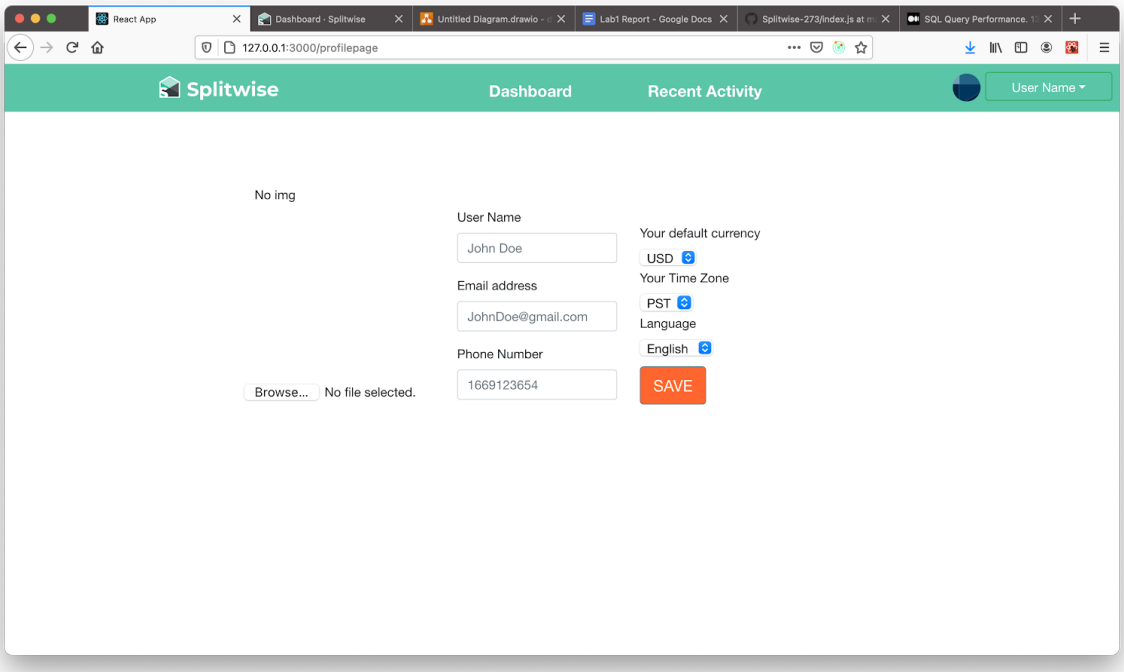
- **Recent Activity:**


Splitwise
Dashboard
Recent Activity
User Name
Recent Acitivity
Show entries
10
Search
Showing 0 entries
Previous
1
Next

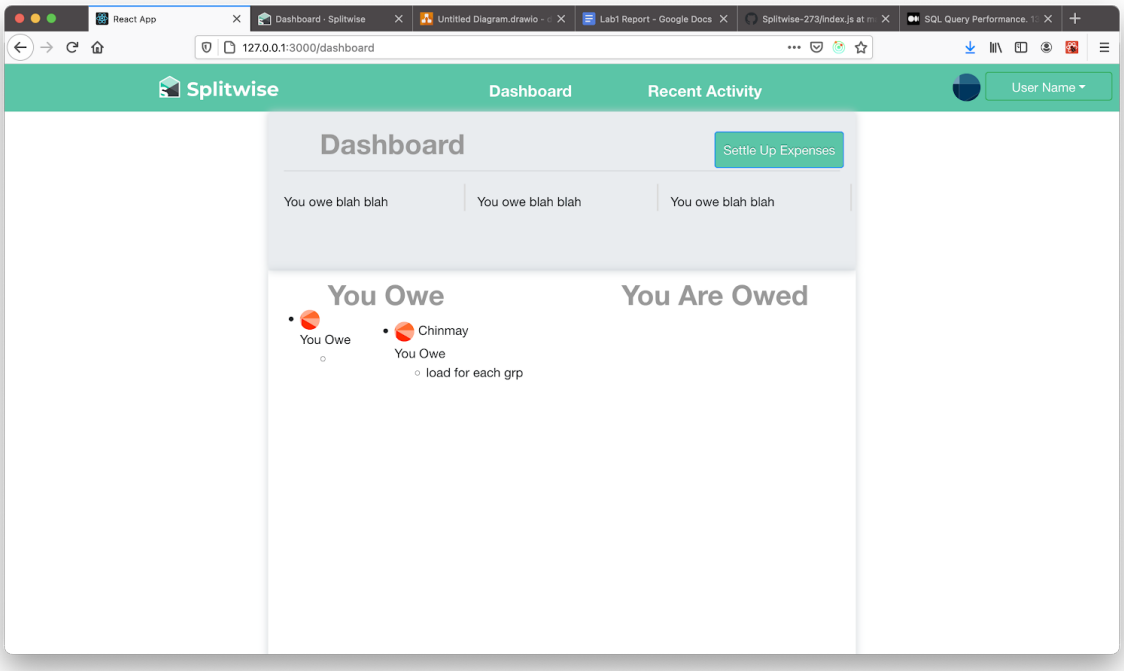
Settle up:


Splitwise
User Name
Settle up with your Friend!
Select...
Close
Settle UP
You Owe
You Are Owed
You Owe
Chinmay
You Owe
load for each grp

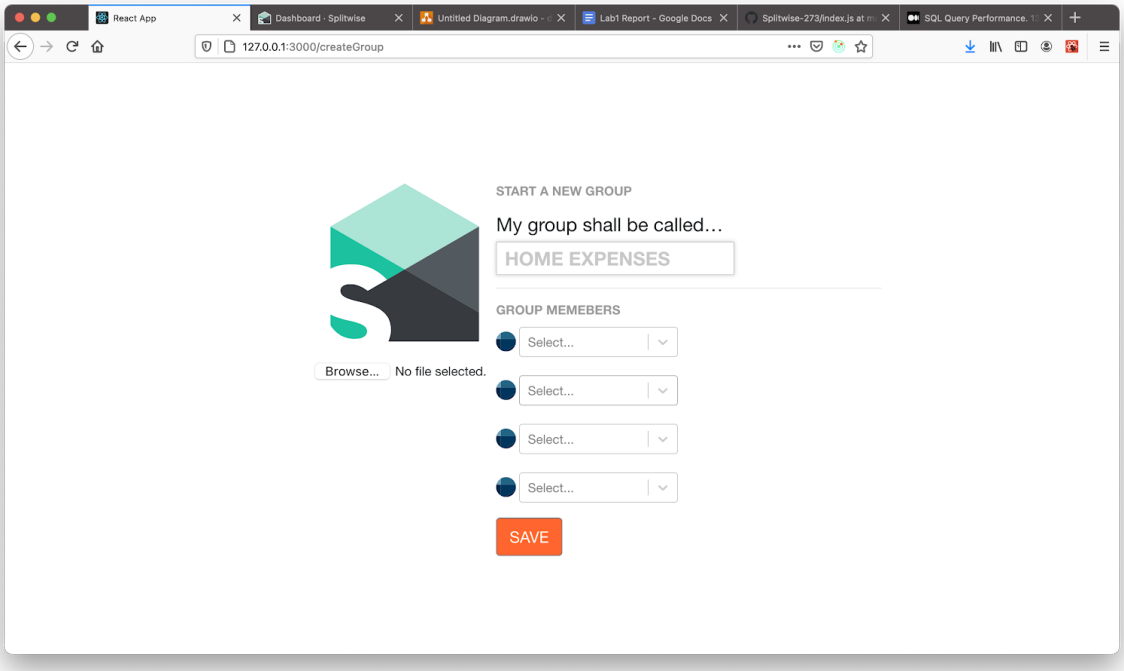
Profile page:



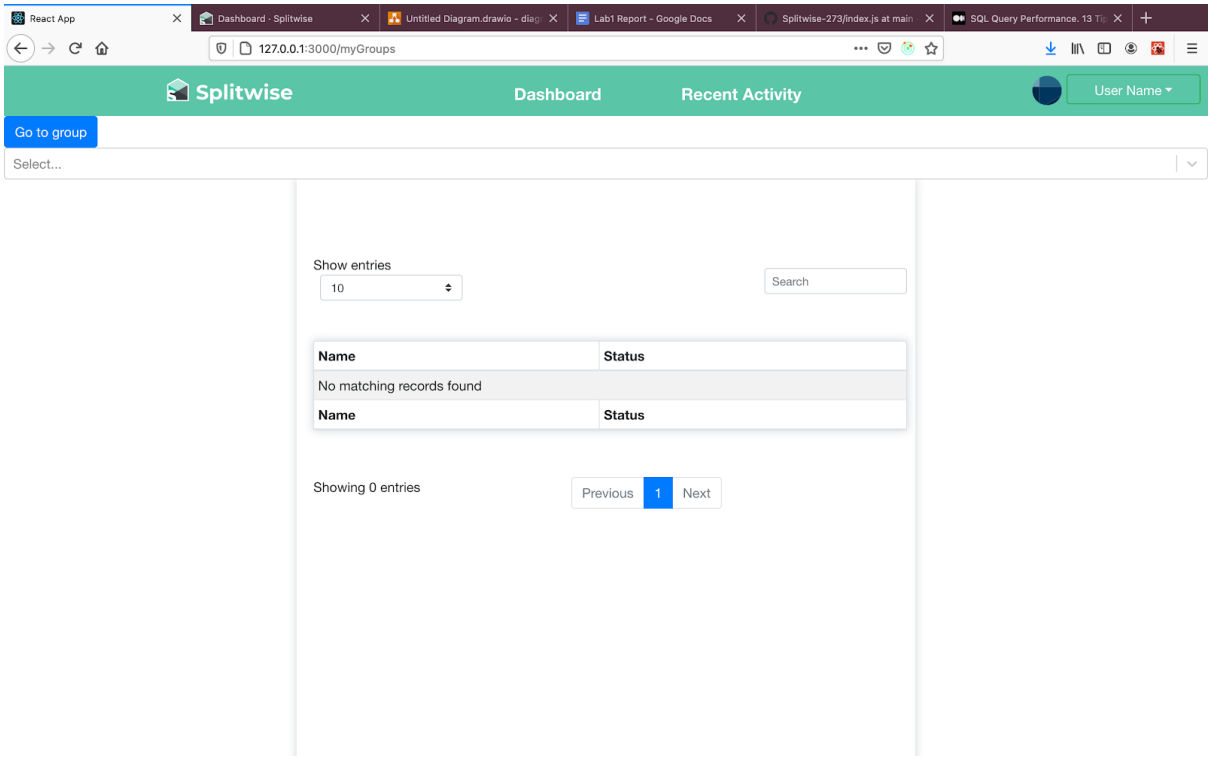
Dashboard:



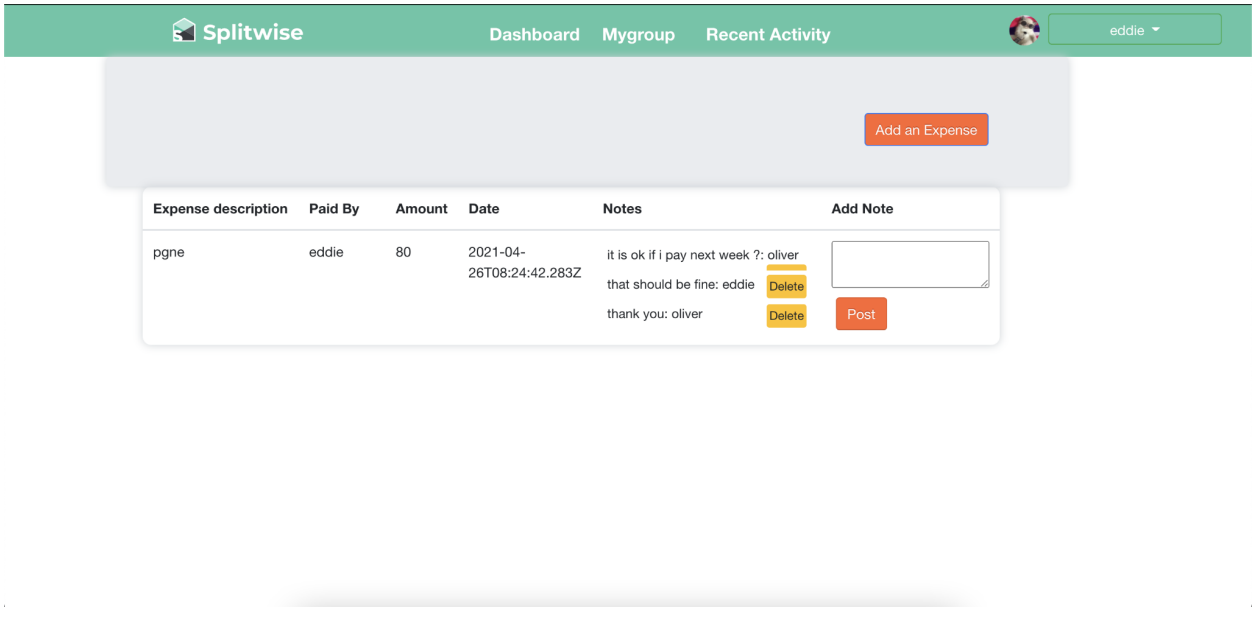
Create group:



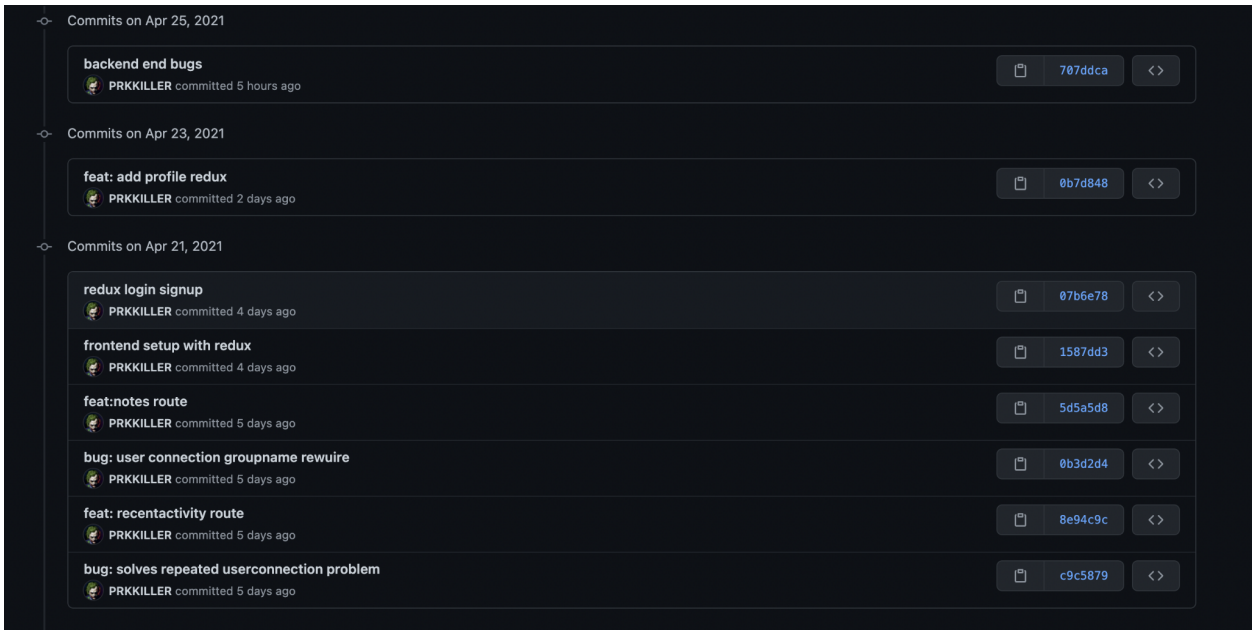
My groups:

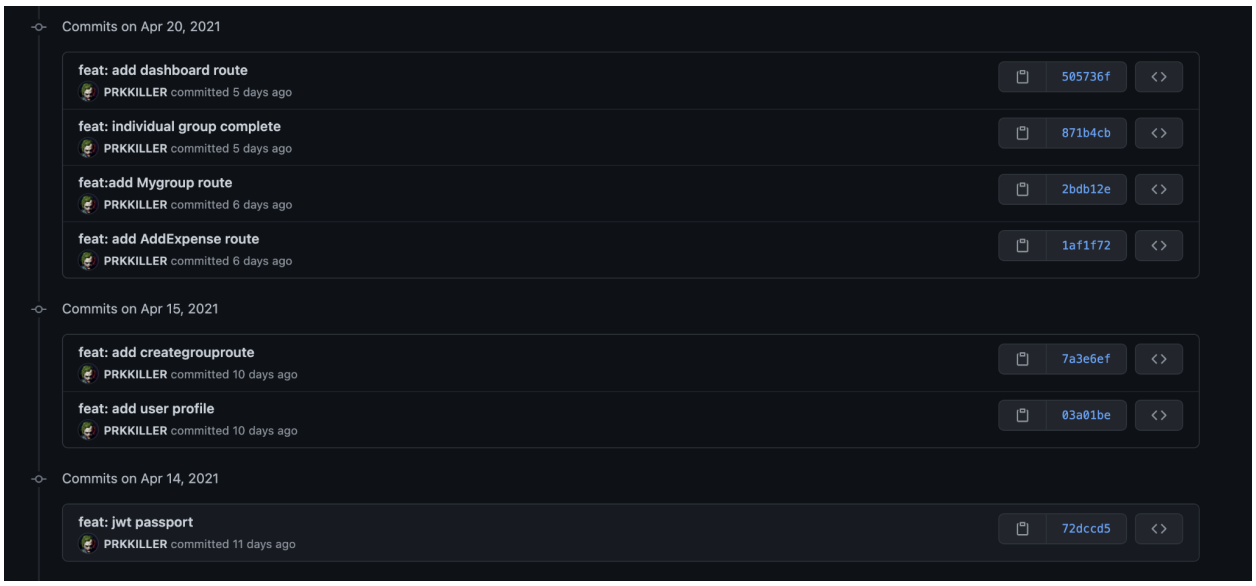

React App
Dashboard - Splitwise
Untitled Diagram.drawio - diag
Lab1 Report - Google Docs
Splitwise-273/index.js at main
SQL Query Performance. 13 Tip
127.0.0.1:3000/myGroups
Splitwise
Dashboard
Recent Activity
User Name
Go to group
Select...
Show entries
10
Search
Name
Status
No matching records found
Name
Status
Showing 0 entries
Previous
1
Next

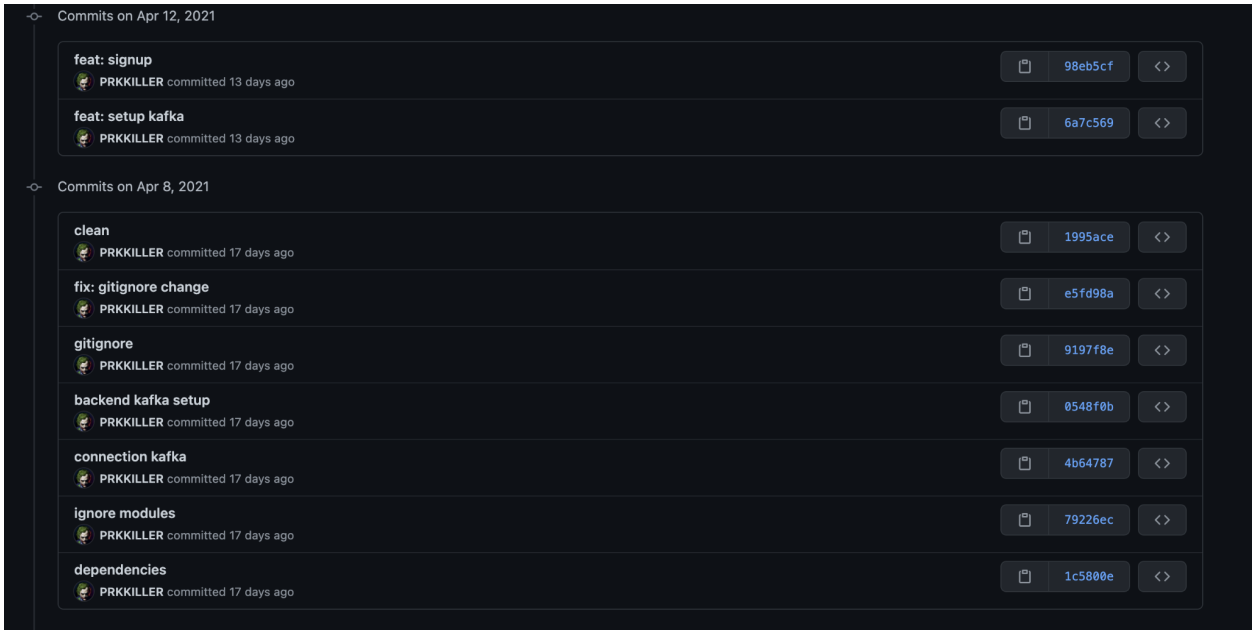
Group Page:

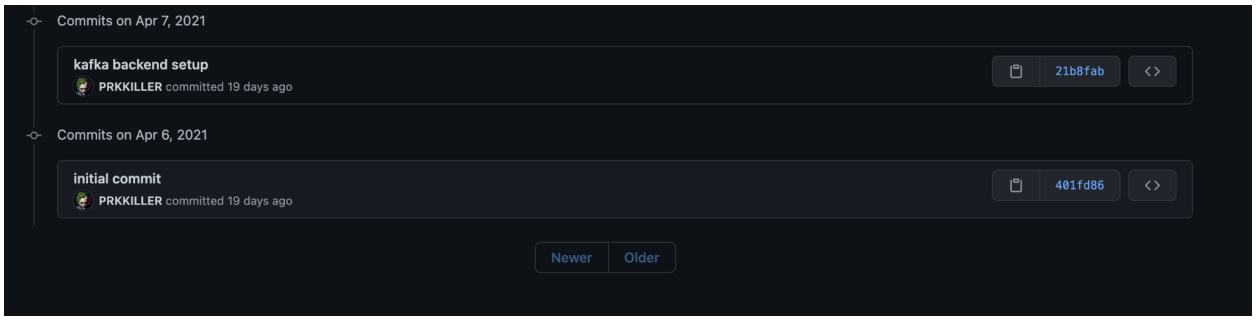

Splitwise
Dashboard
Mygroup
Recent Activity
eddie
Add an Expense
Expense description
Paid By
Amount
Date
Notes
Add Note
pgne
eddie
80
2021-04-26T08:24:42.283Z
it is ok if i pay next week ?: oliver
that should be fine: eddie
Delete
thank you: oliver
Delete
Post

Commit history:

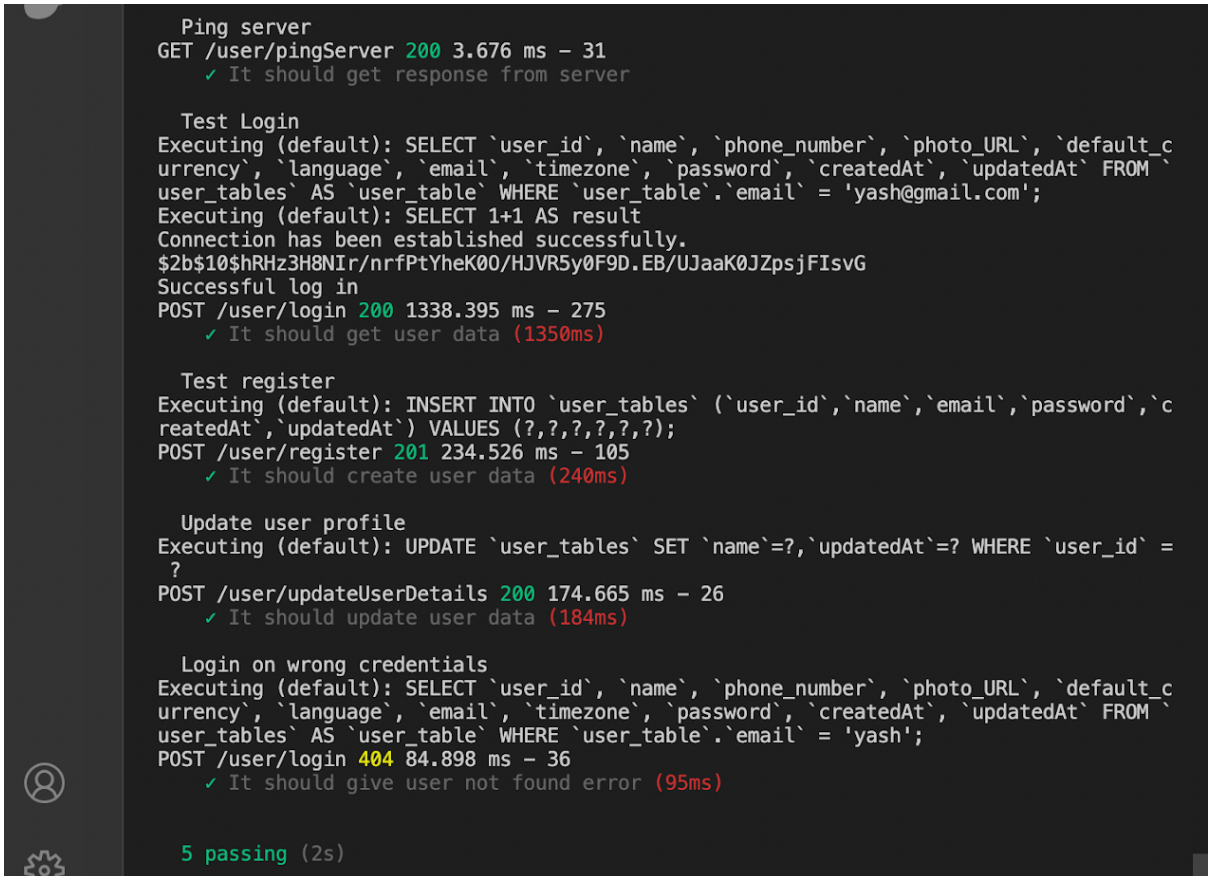






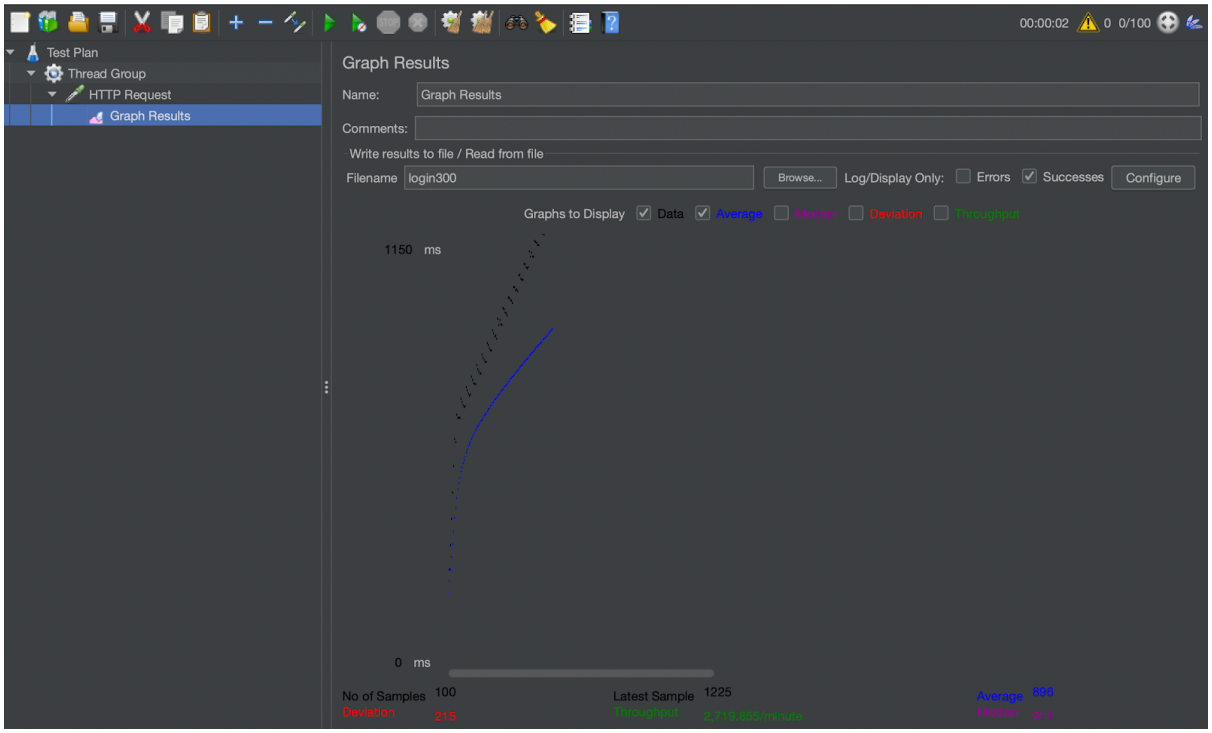


Mocha testing:

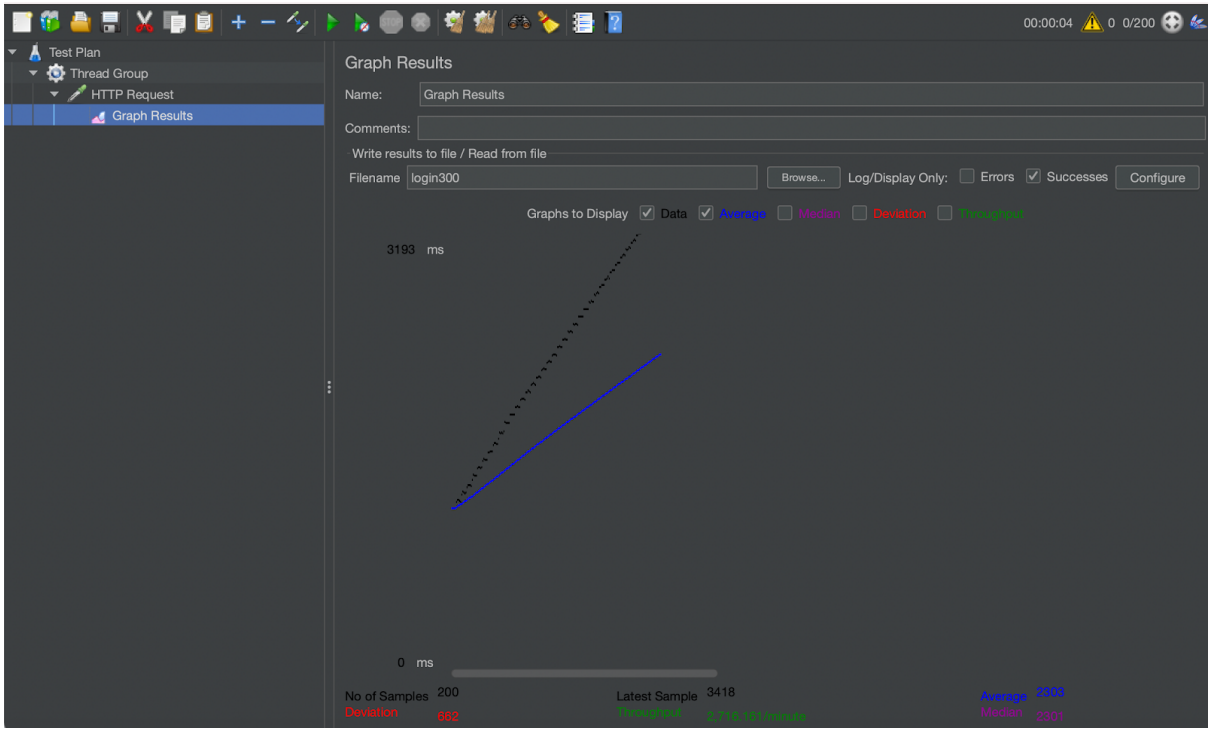

Ping server
GET /user/pingServer 200 3.676 ms - 31
✓ It should get response from server
Test Login
Executing (default): SELECT `user_id`, `name`, `phone_number`, `photo_URL`, `default_currency`, `language`, `email`, `timezone`, `password`, `createdAt`, `updatedAt` FROM `user_tables` AS `user_table` WHERE `user_table`.`email` = 'yash@gmail.com';
Executing (default): SELECT 1+1 AS result
Connection has been established successfully.
$2b$10$hRHz3H8NIr/nrfPtYheK0O/HJVR5y0F9D.EB/UJaaK0JZpsjFIsvG
Successful log in
POST /user/login 200 1338.395 ms - 275
✓ It should get user data (1350ms)
Test register
Executing (default): INSERT INTO `user_tables` (`user_id`,`name`,`email`,`password`,`createdAt`,`updatedAt`) VALUES (?,?,?,?,?,?);
POST /user/register 201 234.526 ms - 105
✓ It should create user data (240ms)
Update user profile
Executing (default): UPDATE `user_tables` SET `name`=?,`updatedAt`=? WHERE `user_id` = ?
POST /user/updateUserDetails 200 174.665 ms - 26
✓ It should update user data (184ms)
Login on wrong credentials
Executing (default): SELECT `user_id`, `name`, `phone_number`, `photo_URL`, `default_currency`, `language`, `email`, `timezone`, `password`, `createdAt`, `updatedAt` FROM `user_tables` AS `user_table` WHERE `user_table`.`email` = 'yash';
POST /user/login 404 84.898 ms - 36
✓ It should give user not found error (95ms)
5 passing (2s)

- **Jmeter Testing**

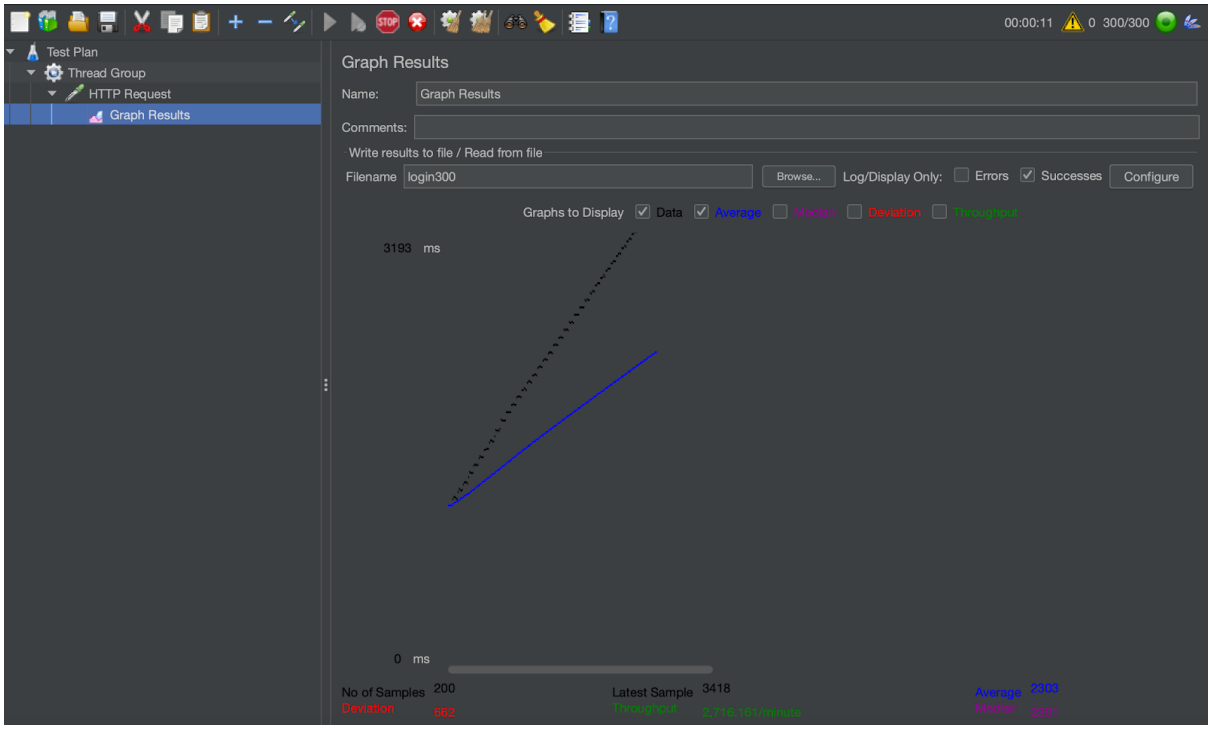
  100 Users Without Pooling



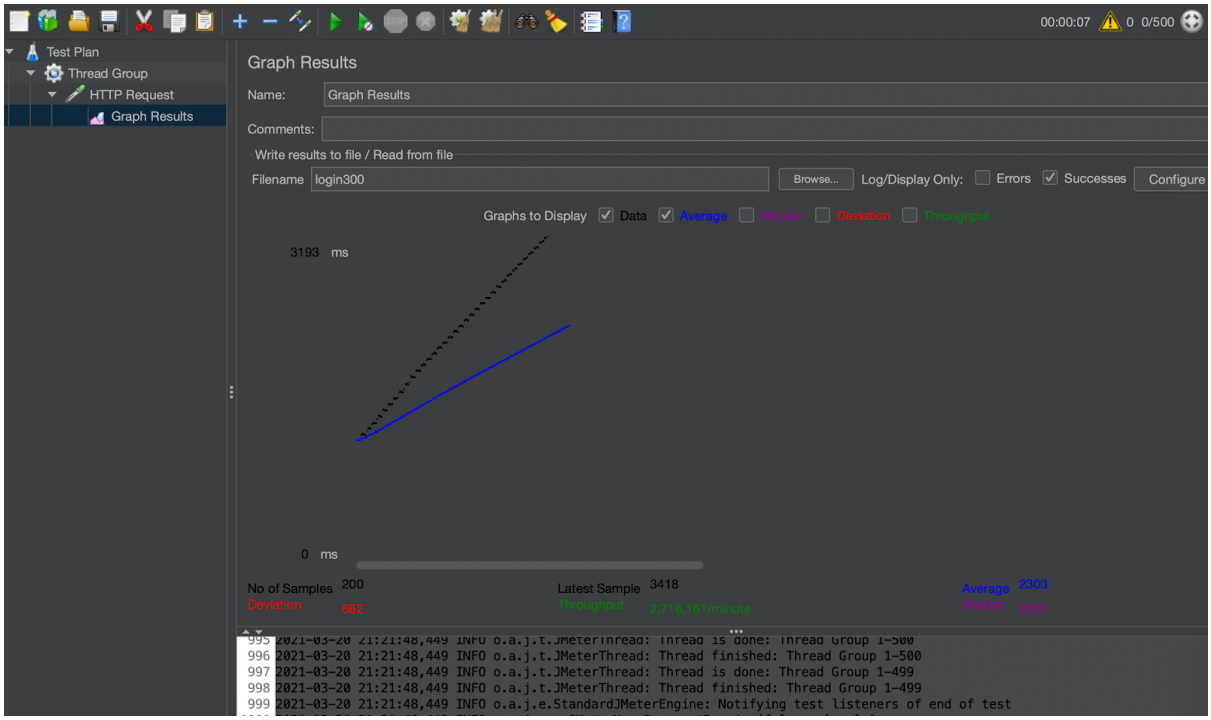
  200 Users without pooling



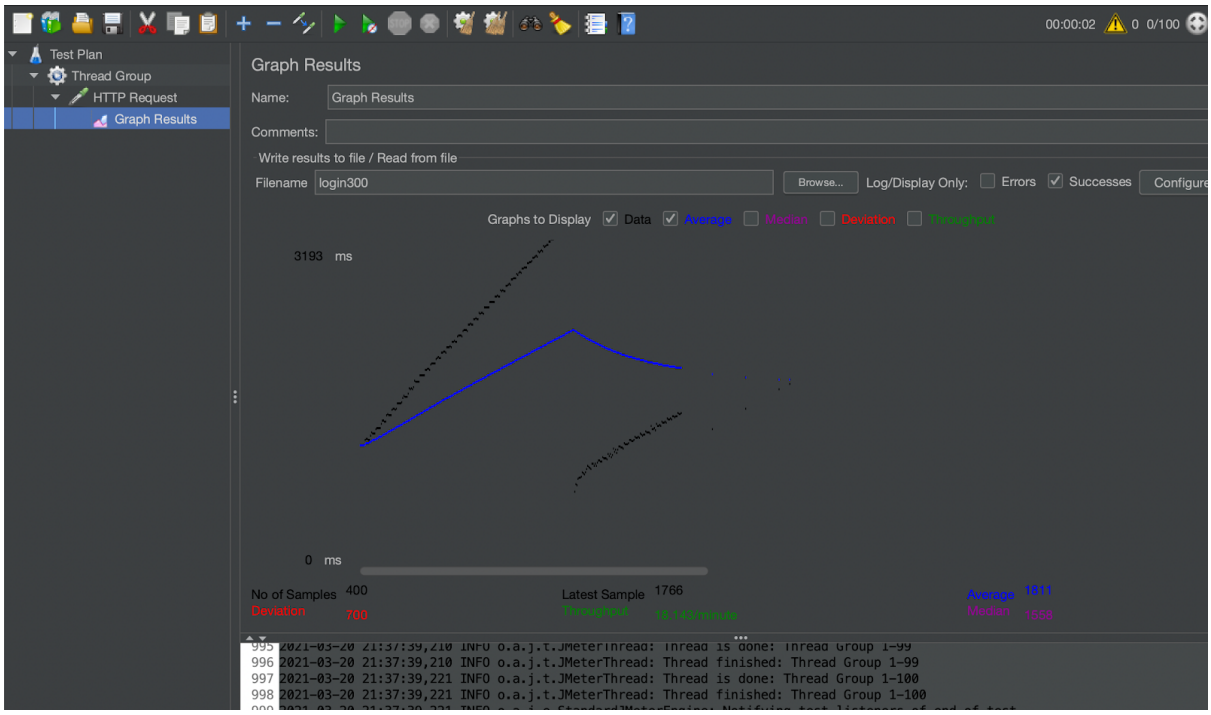
300 Users without pooling



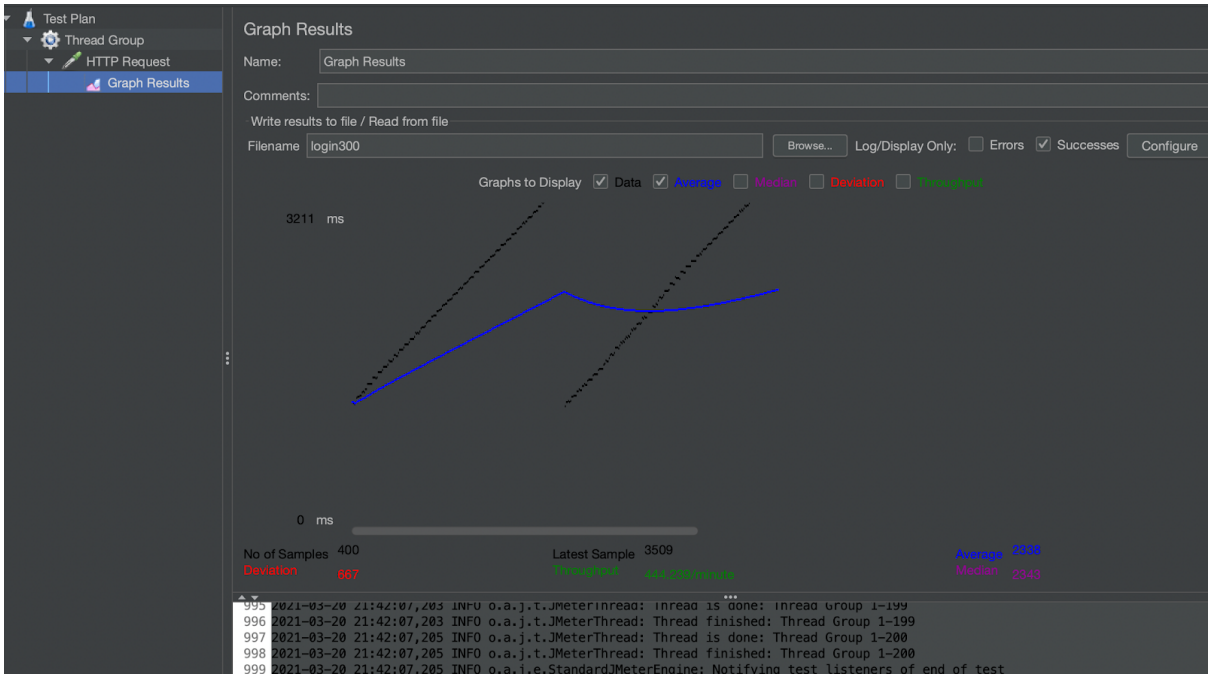
400 user without pooling



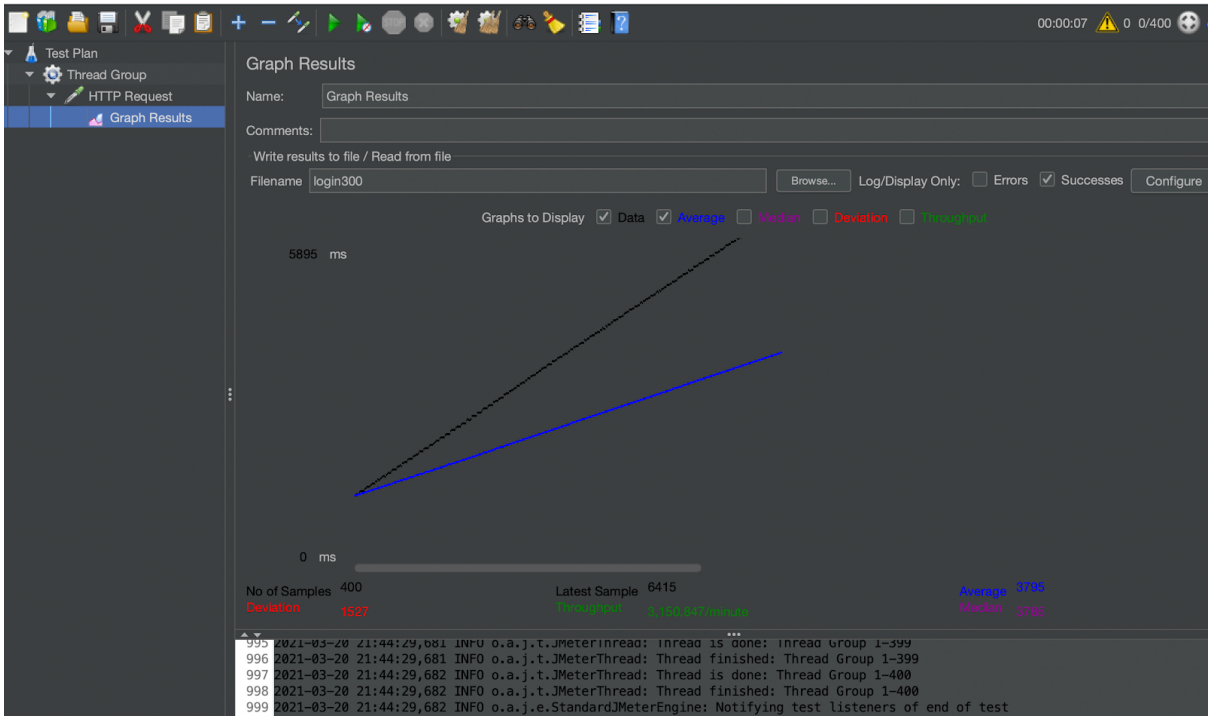
100 Users with pooling



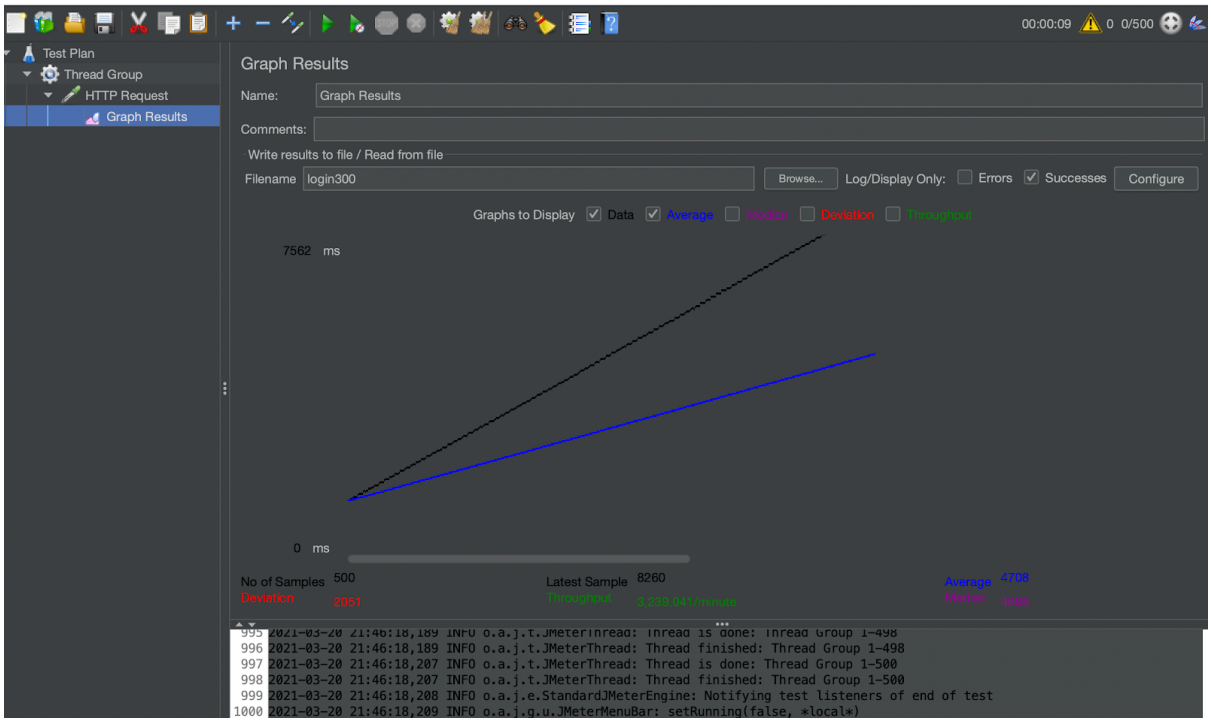
200 user with pooling



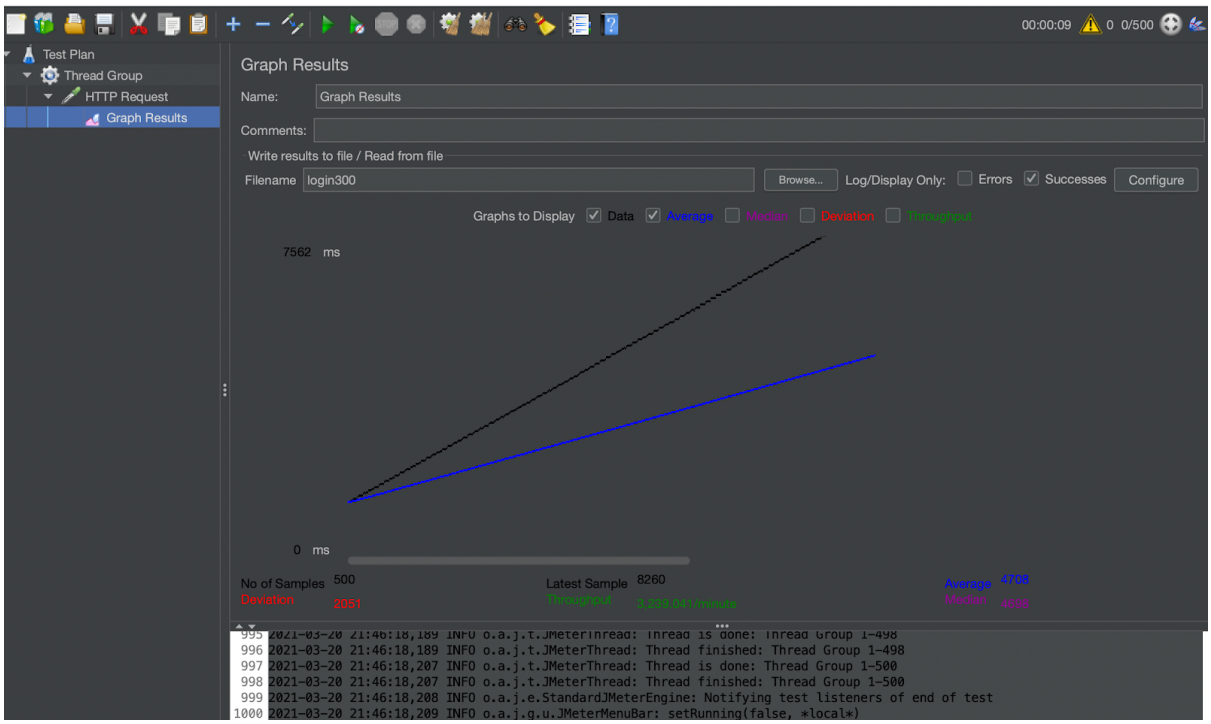
300 users with pooling



400 users with pooling



500 users with pooling



Answers to the questions:

1) For authentication in the first lab I used the local storage to persist user information that was necessary for all the routes. This time around I incorporated JWT tokens using passportJS and protected all the routes with these tokens. These tokens had user information encrypted in them and they also have a limited expiry period. This means that every resource is protected using these stateless tokens and also they will always only have the fresh data. Hence, it can deduced that using JWT tokens is the more scalable and secure technique of authentication
2) The kafka framework is stream-processing bus software which means it is a high end message queue. Given the extent of the splitwise clone there were not too many concurrent requests and the number of services was limited. But using kafka in this project ensured that the services in use are all parallelized. For example, when the dashboard data is requested by two users both the requests are sent to Kafka as two atomic messages and these messages get data from the Database and send response as another set of atomic messages. This divides these two requests into four parallel units as opposed to two. Now when the replication factor and partitions are increased the parallelism also increases. In this way the application is exponentially more scaled than it was initially. Although the performance difference will really show when there will be multiple concurrent users requesting the API service
3) Given my implementation, the user and the group tables are such that they're almost mutually exclusive and don't have cascading effects on one another. But the transactions table is heavily related to the both of them which increases the groupby functions on these tables. These tables if stored in the MySQL database will preserve their relational attributes thus making groups and joins easier whereas the user and group tables will be stored in the MongoDB database making it easier to write and read from as a lot of information can be embedded in them making reading easy. Thus the dashboard or group summaries will be fetched very fast.