PRATIK KASLE

CMPE:273 LAB-1 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=0QJO7fE_87U

Github Link:https://github.com/PRKKILLER/React-

SYSTEM DESIGN:

- **Database:**
  - **MySQL database hosted on AWS RDS is used:**
  - **AWS S3 is used for storage.**
  - **MySQL Schema:**

    a. **User**

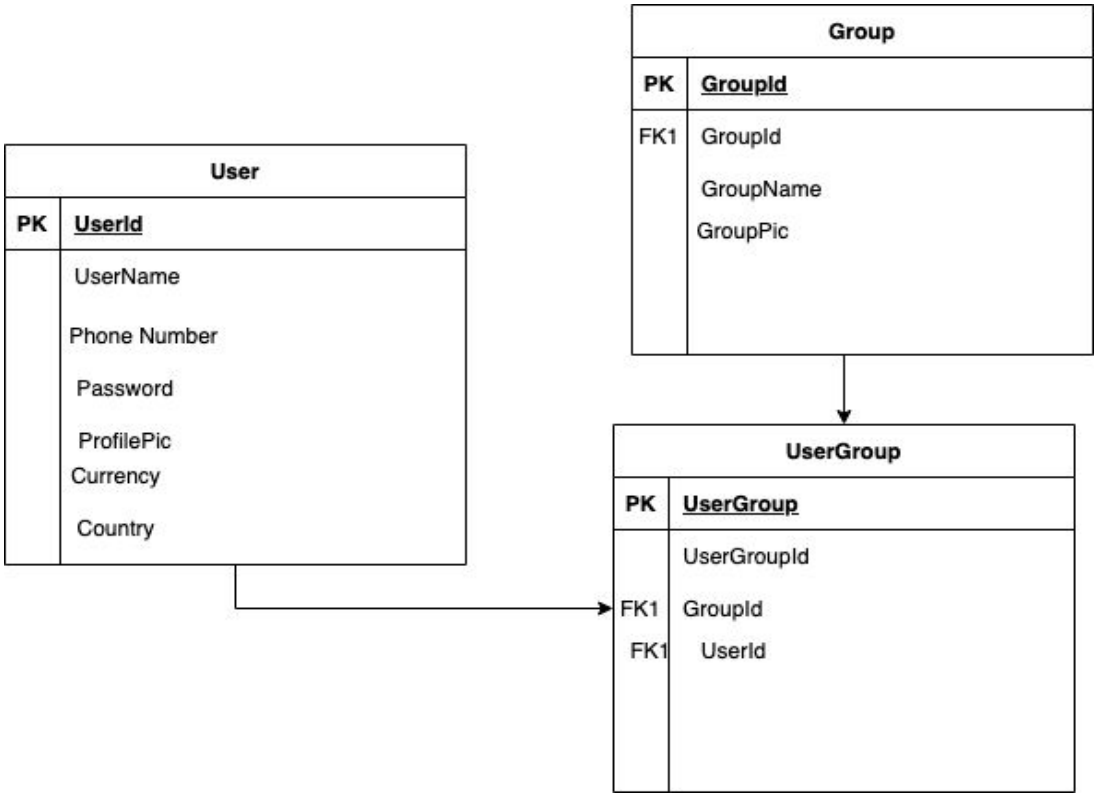
    UserId:EmailId of User-Primary Key-
    UserName:Name of the user.can be changes
    Profile:Link to the AWS S3 store where profile picture is uploaded
    Currency:User Default currency
    TimeZone:Time zone user operates.

    - **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.

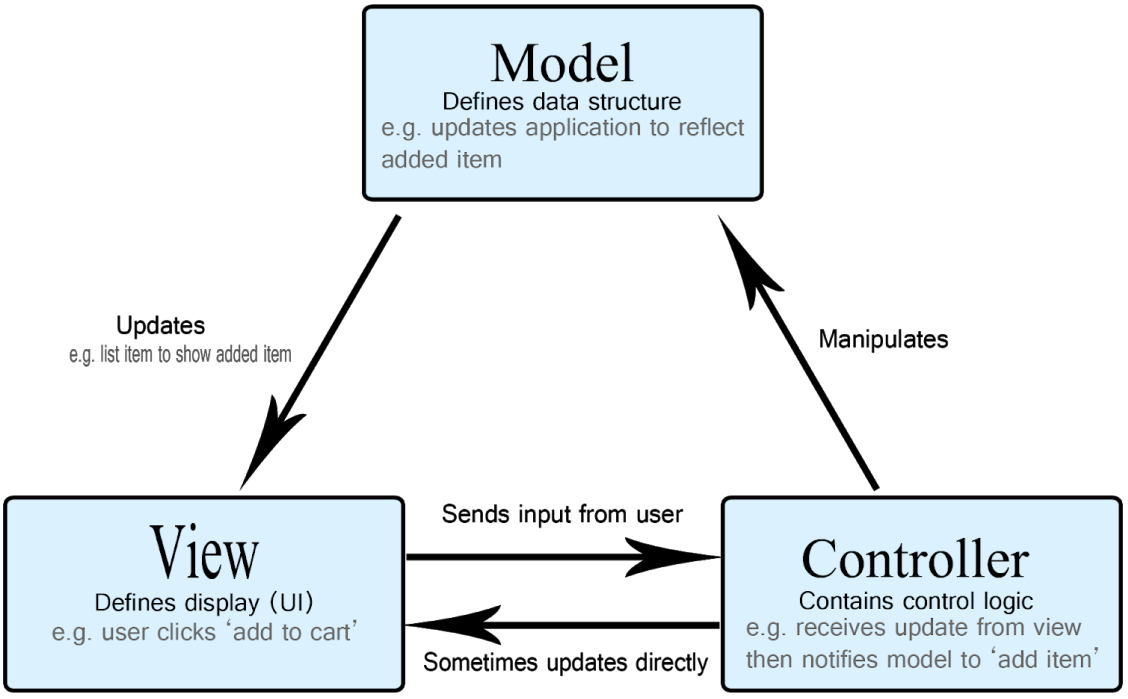
  UUID
  UserId1
  UserId2
  GroupId
  Owes

- **User-Group-Transaction**
  UUID
  EmailId
  Description
  Amount
  GroupId
- **Recent-activity**
  activityId
  OperationType

GroupId
GroupName

- **ORM(Object Relational Model)**
  1. ORM converts a relational database into objects
  2. Sequelize is a promise based node based ORM.

- **Backend Design:**



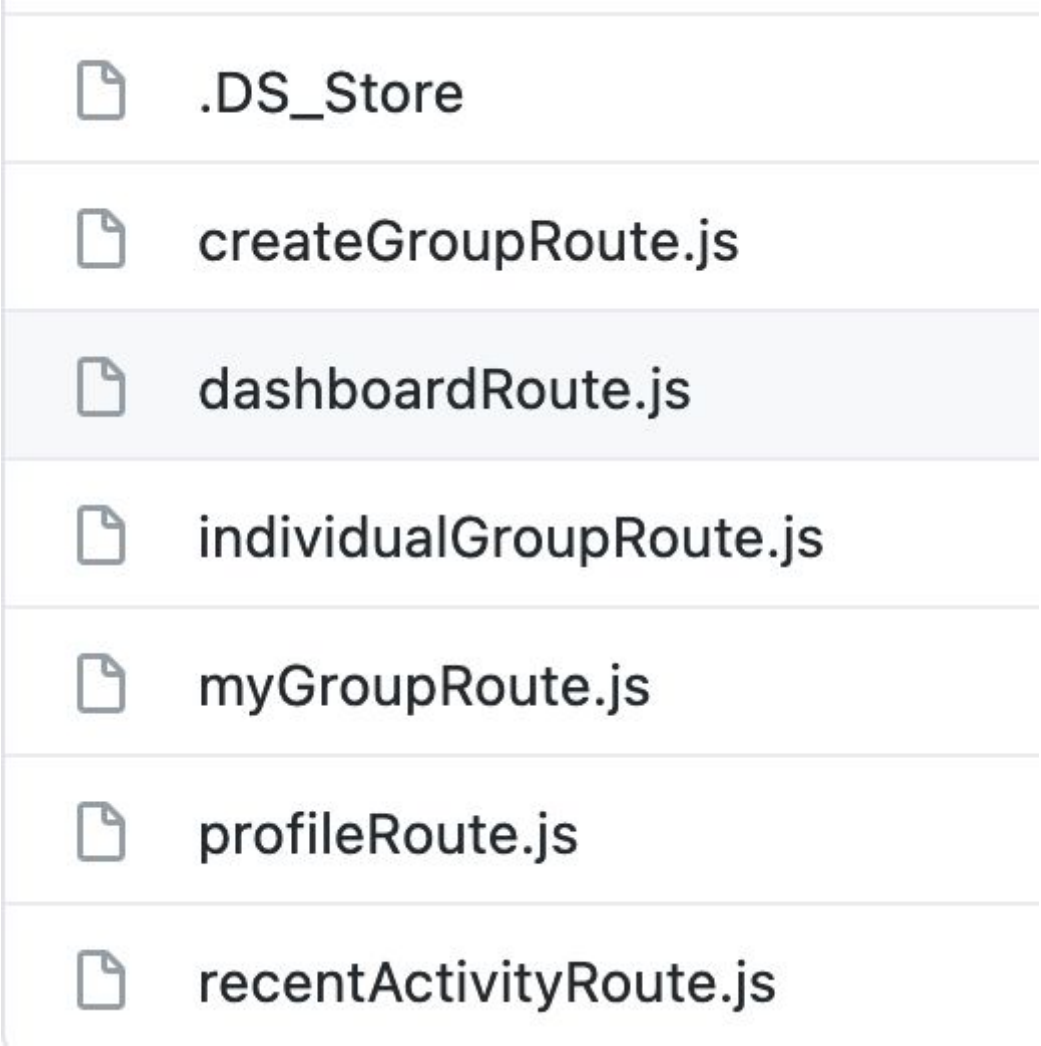
- A popular application architecture pattern MVC(Model Controller View) is used in this project.
- Model accesses the database and serves the data in object format.

- User_Grp_Transaction.js
- group.js
- group_user.js
- recentactivity.js
- user.js
- user_user.js

- Controller
  Used for database CRUD operations:

  - .DS_Store
  - User_Grp_Transaction.js
  - group.js
  - group_user.js
  - recentactivity.js
  - user.js
  - user_user.js

- Router
  Used to Route frontend API calls:



- **FRONTEND**
  1. **Landing Page**
  2. **Login Page**
  3. **Signup Page**
  4. **Dashboard Page**
  5. **MyGroups Page**
  6. **Group Page**
  7. **User Profile Page**
  8. **Group Profile Page**

- **Git Commit History:**

Commits on Mar 8, 2021

Redux Done
PRKKILLER committed 12 days ago — 319c0f3

Commits on Mar 5, 2021

signup_login_route
PRKKILLER committed 15 days ago — 6c8afb4

Commits on Mar 4, 2021

reducer skeleton code and linting
PRKKILLER committed 16 days ago — b4956b5

Commits on Mar 3, 2021

login route
PRKKILLER committed 17 days ago — de696dc

Commits on Feb 28, 2021

database-connected
PRKKILLER committed 20 days ago — 58cacff

Commits on Feb 27, 2021

Initial commit
PRKKILLER committed 21 days ago — 021746f

Commits on Mar 14, 2021

feat:landing
PRKKILLER committed 6 days ago — bfd2c69

feat: add profilepage
PRKKILLER committed 6 days ago — 2834c18

feat:recent activity,all group page
PRKKILLER committed 6 days ago — 6586e43

Commits on Mar 11, 2021

feat: add upper and side navbar
PRKKILLER committed 9 days ago — b583be8

feat: add commonpage views
PRKKILLER committed 9 days ago — 088ce15

Commits on Mar 10, 2021

dashboard css
PRKKILLER committed 10 days ago — 0ae841e

dashboard UI
PRKKILLER committed 10 days ago — 2edc8d1

Commits on Mar 17, 2021

feat: modify create grp PRKKILLER committed 3 days ago	cb95cd0
Update createGroupRoute.js PRKKILLER committed 3 days ago	8886bca
Update upperNavbar.js PRKKILLER committed 3 days ago	5e1ce03
feat: add create group PRKKILLER committed 3 days ago	1787434

Commits on Mar 16, 2021

feat: logout connect PRKKILLER committed 4 days ago	b5a6bd0
feat:local storage email on login PRKKILLER committed 4 days ago	904f7b4

Commits on Mar 15, 2021

feat:add git ignore for backend PRKKILLER committed 5 days ago	d36b061
feat:login and signup complete PRKKILLER committed 5 days ago	abaf580
feat:login and signup frontend PRKKILLER committed 6 days ago	681be02

Commits on Mar 19, 2021

feat: complete individual group route PRKKILLER committed yesterday	24ef567

Commits on Mar 18, 2021

cleaning PRKKILLER committed 2 days ago	b94e6bf
cleanup PRKKILLER committed 2 days ago	7c57e15
feat: add profile route PRKKILLER committed 2 days ago	2c78063
creategroup route complete PRKKILLER committed 2 days ago	45e8ab0
create group route complete PRKKILLER committed 2 days ago	897d5ea
feat:transcation and recent activity PRKKILLER committed 3 days ago	51a01c2

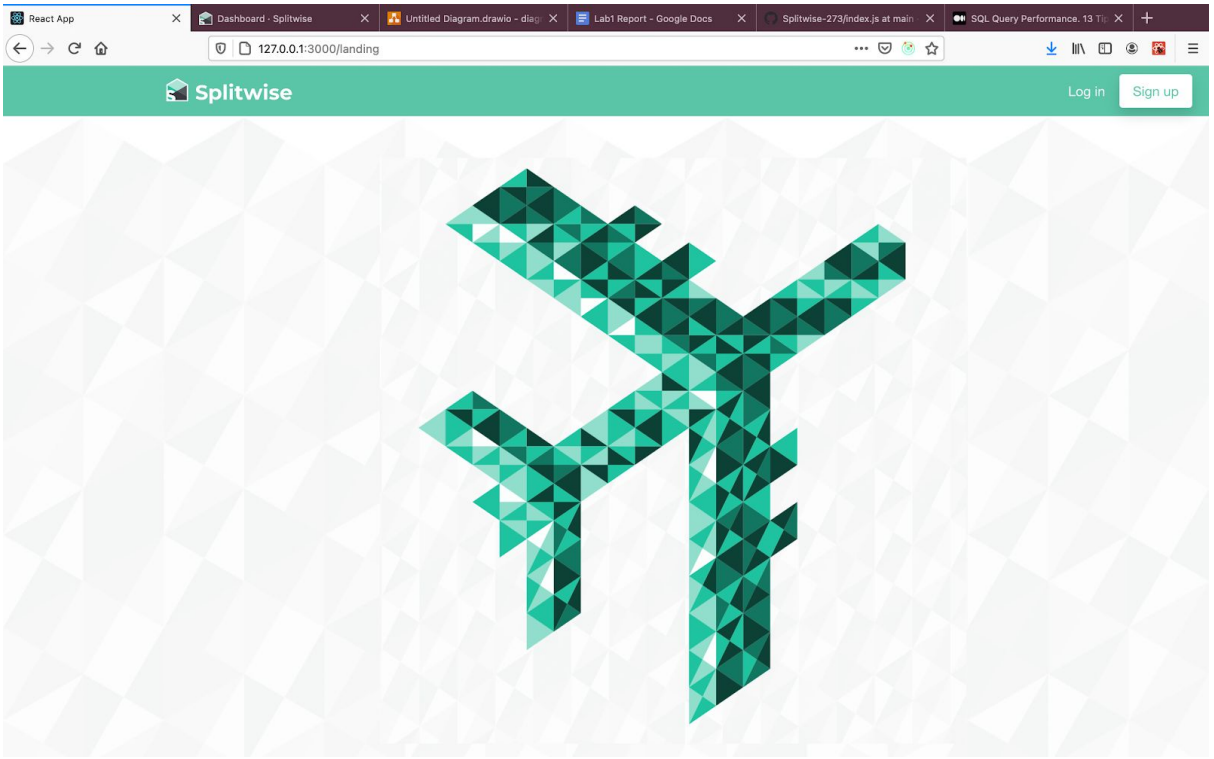
Commits on Mar 20, 2021

datachange PRKKILLER committed 16 hours ago	06113f0
bugg removed PRKKILLER committed 16 hours ago	1c70e98
feat:add dashboard route PRKKILLER committed 16 hours ago	7f1d5b9
cleaning PRKKILLER committed 17 hours ago	3863610
feat:recent activity route complete PRKKILLER committed 17 hours ago	232eecd
Feat: add simplify algorithm user user PRKKILLER committed 19 hours ago	4ca5d19

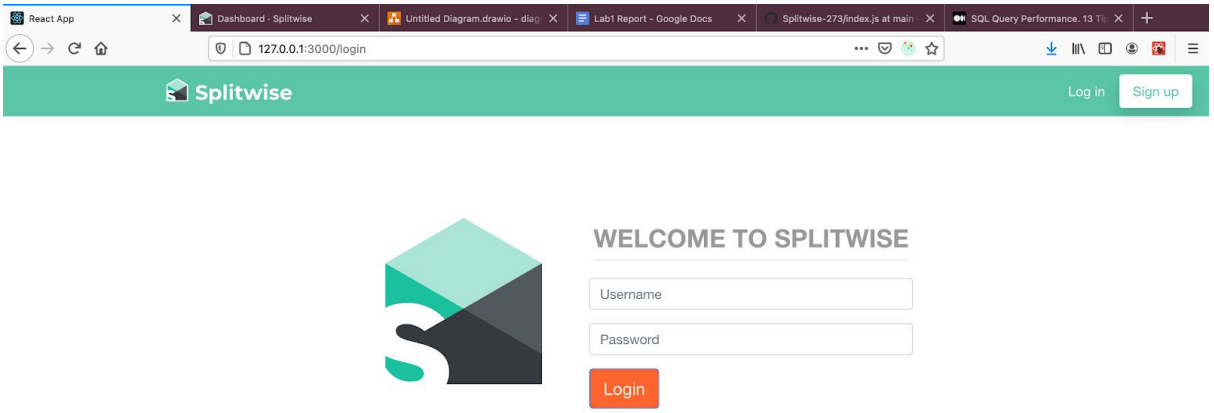
- **Frontend UI:**

PRATIK KASLE

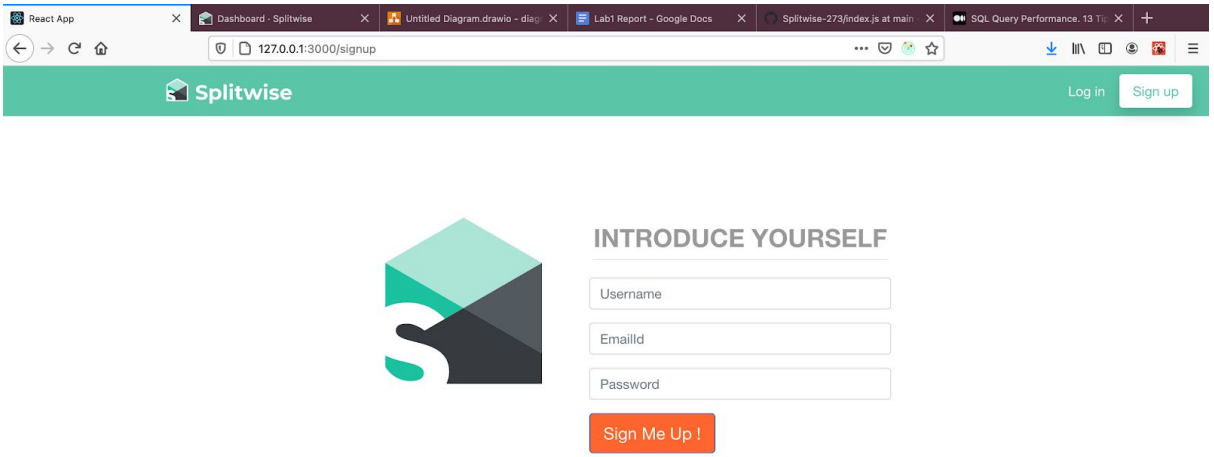
Landing page:



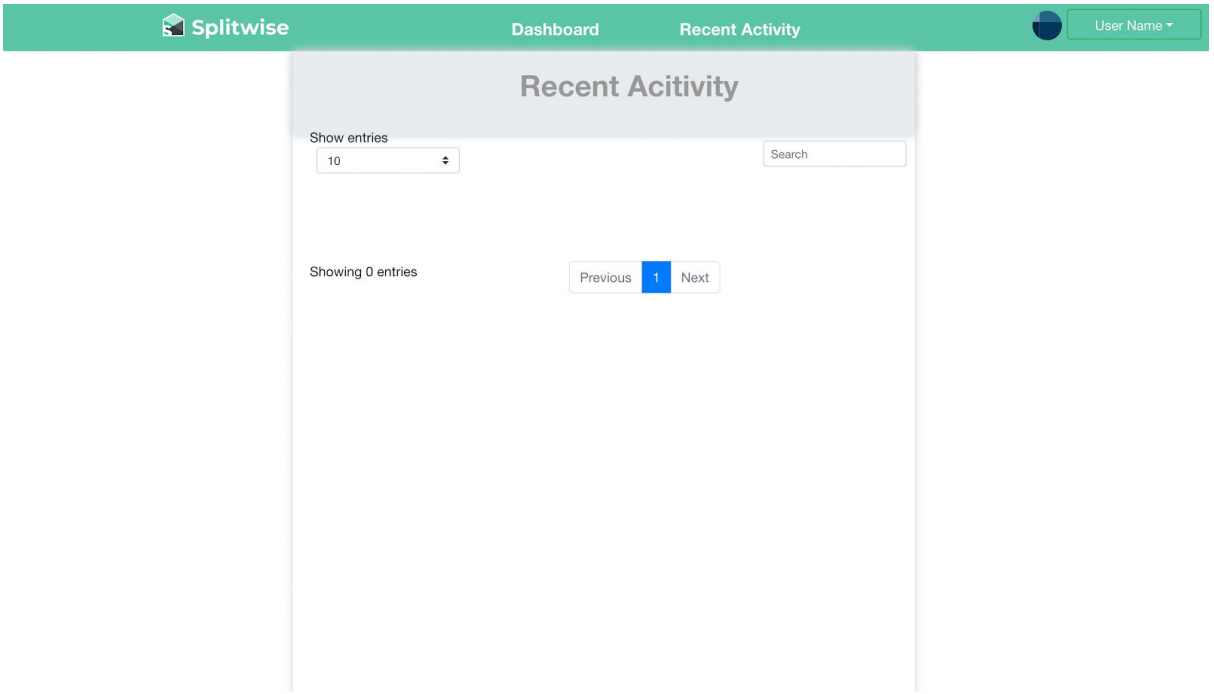
Login:



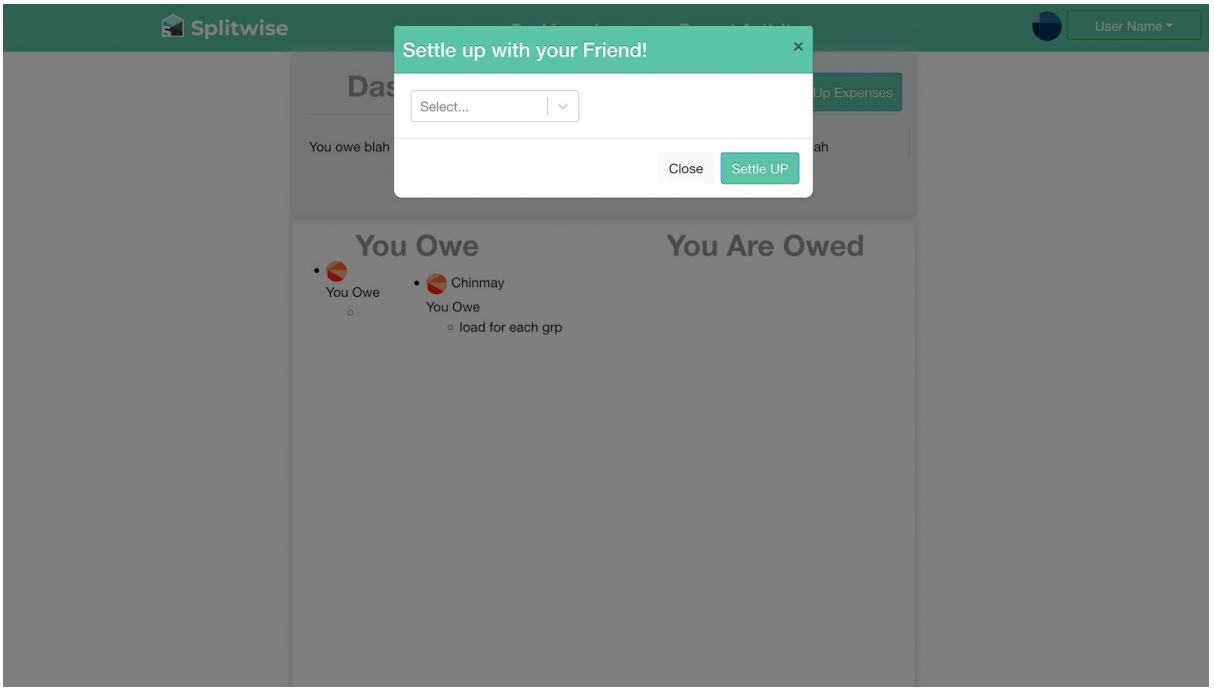
Register:



- ## Recent Activity:

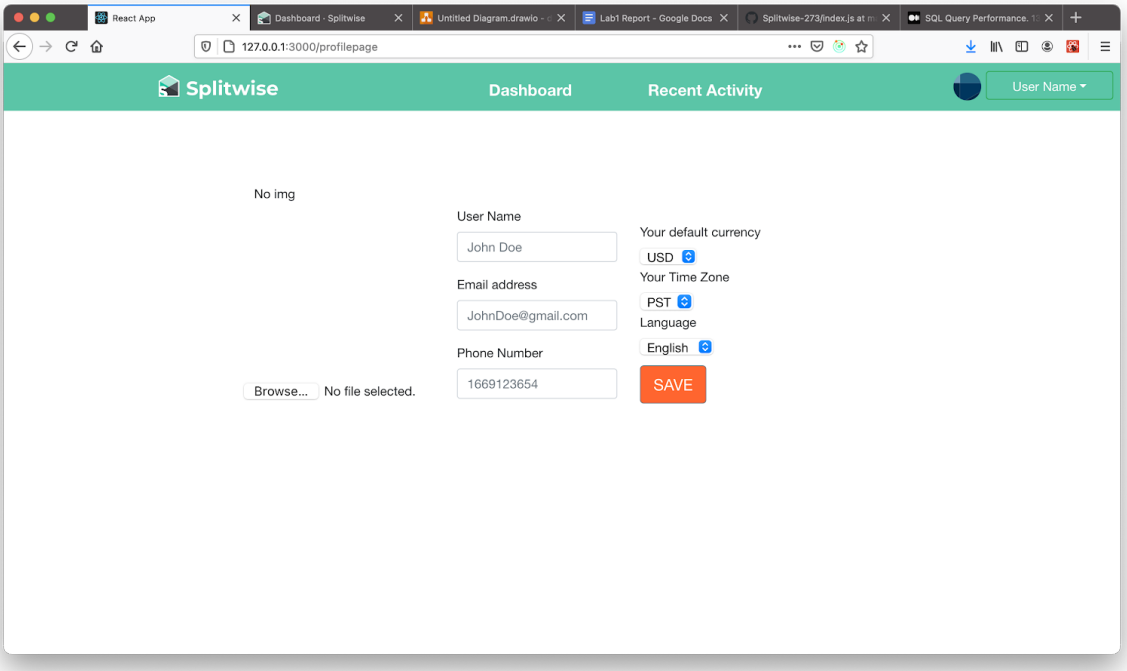


Settle up:

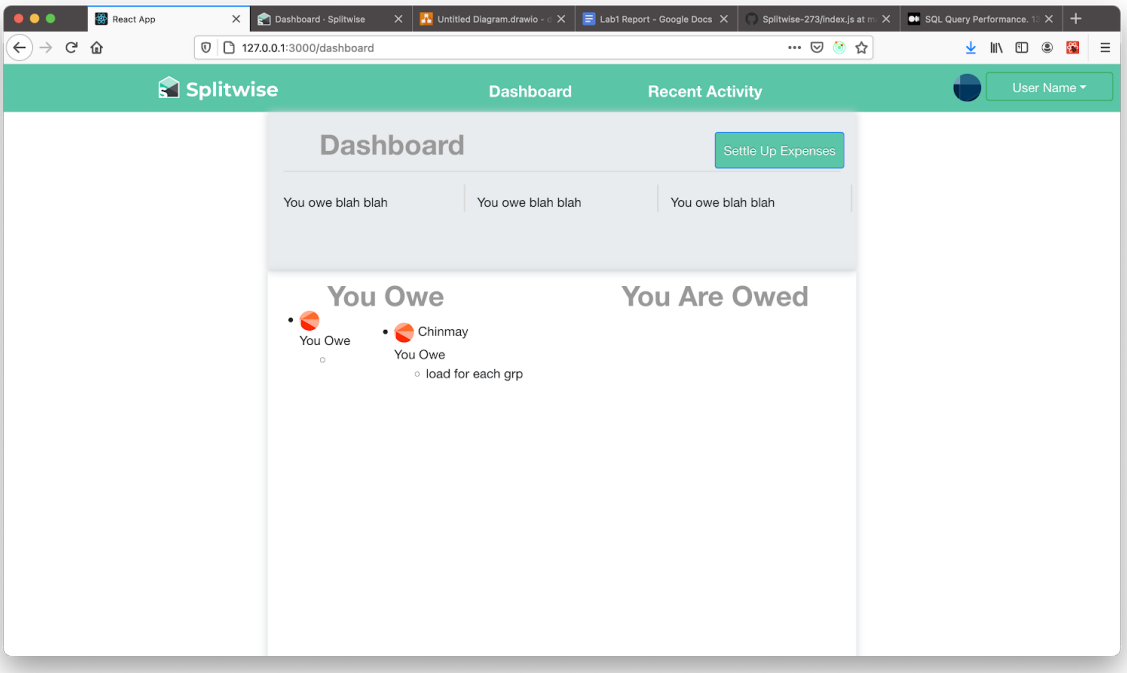


Profile page:

PRATIK KASLE

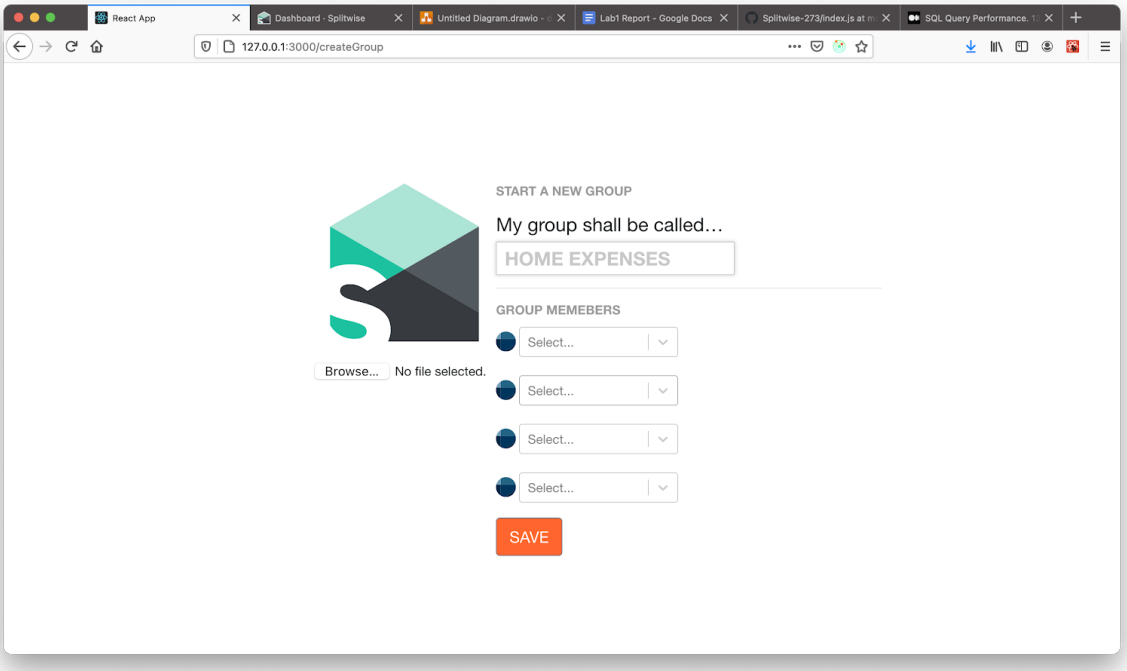


Dashboard:

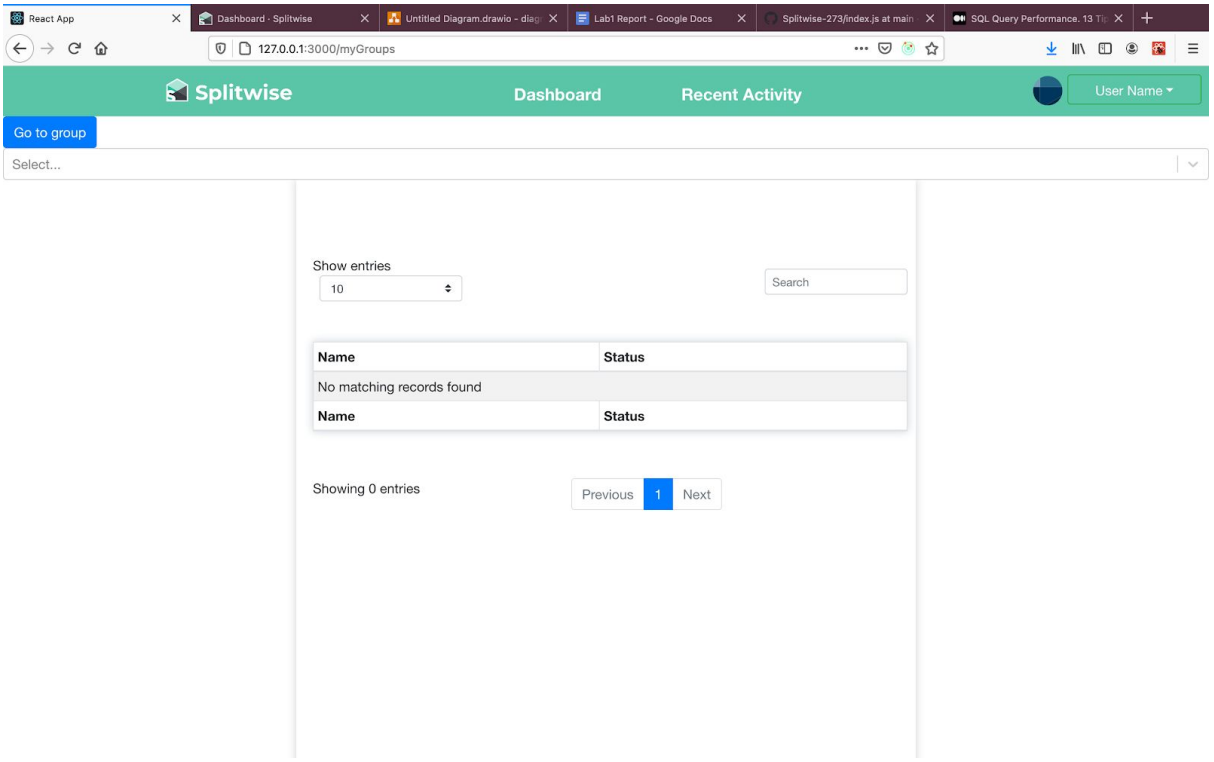


Create group:

PRATIK KASLE



My groups:

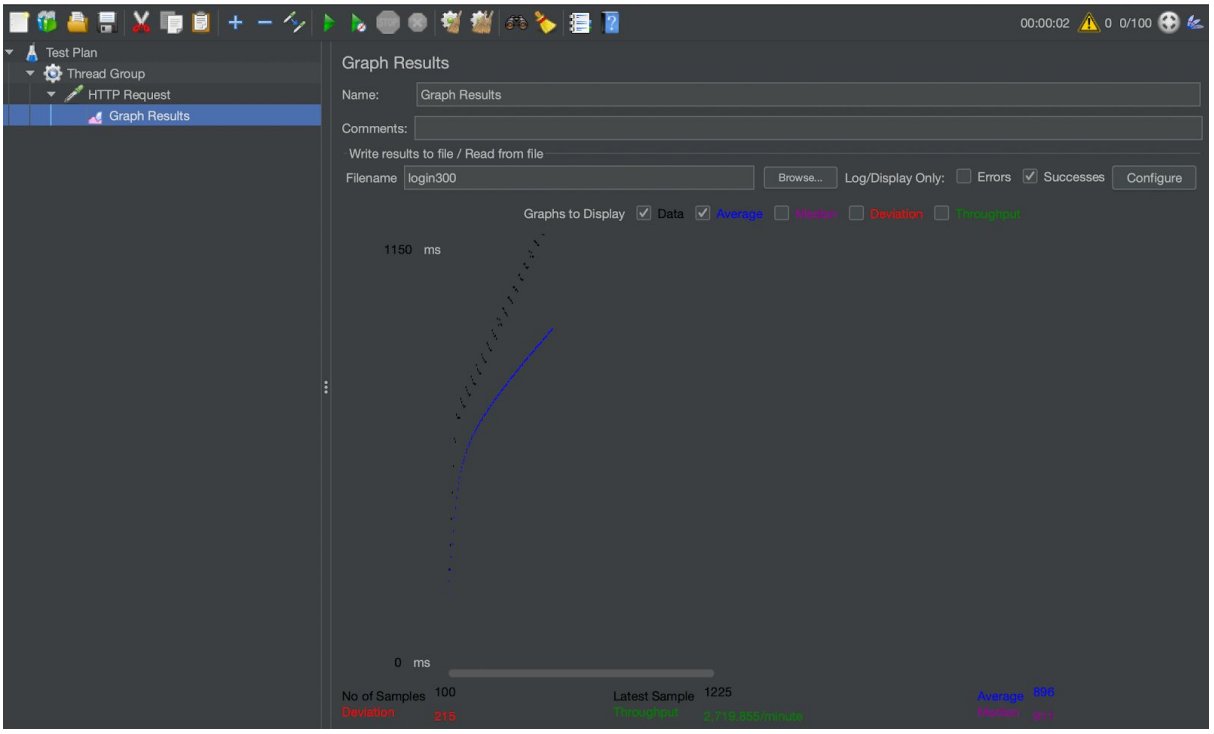


Mocha testing:

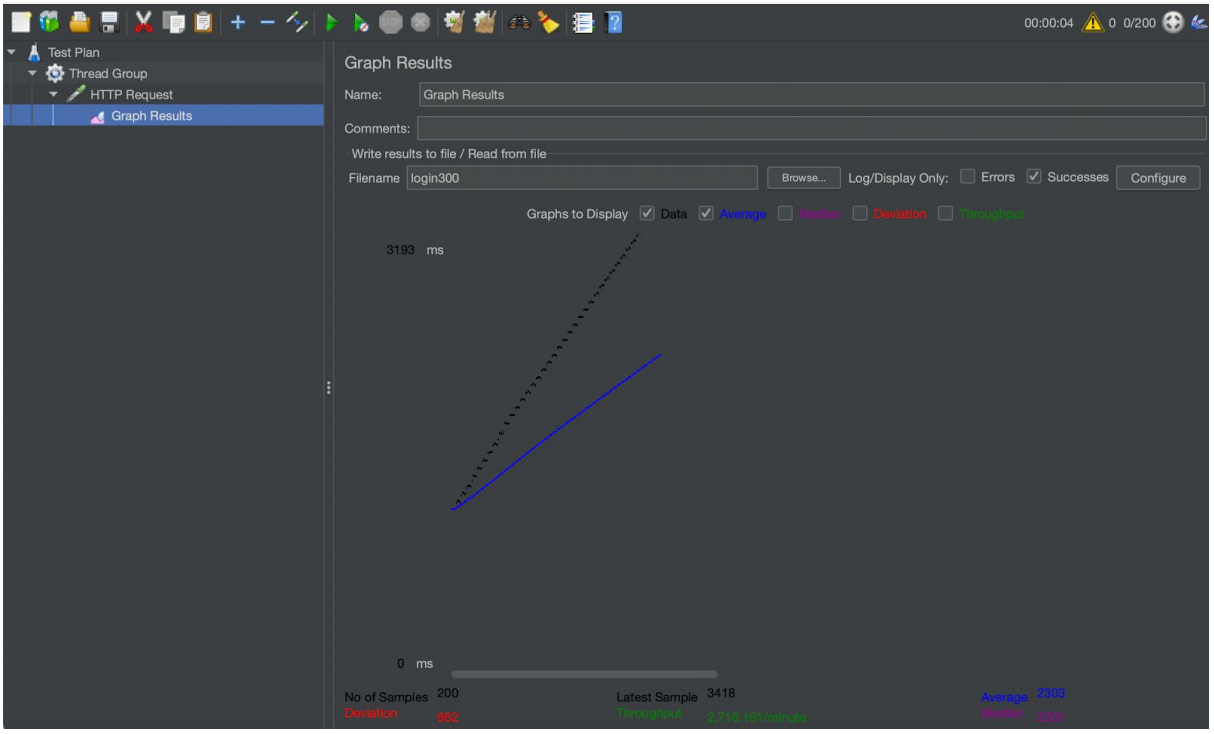
```
  Ping server
GET /user/pingServer 200 3.676 ms - 31
    ✓ It should get response from server

  Test Login
Executing (default): SELECT `user_id`, `name`, `phone_number`, `photo_URL`, `default_c
urrency`, `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/nrfPtYheK00/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`,`c
reatedAt`,`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_c
urrency`, `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**

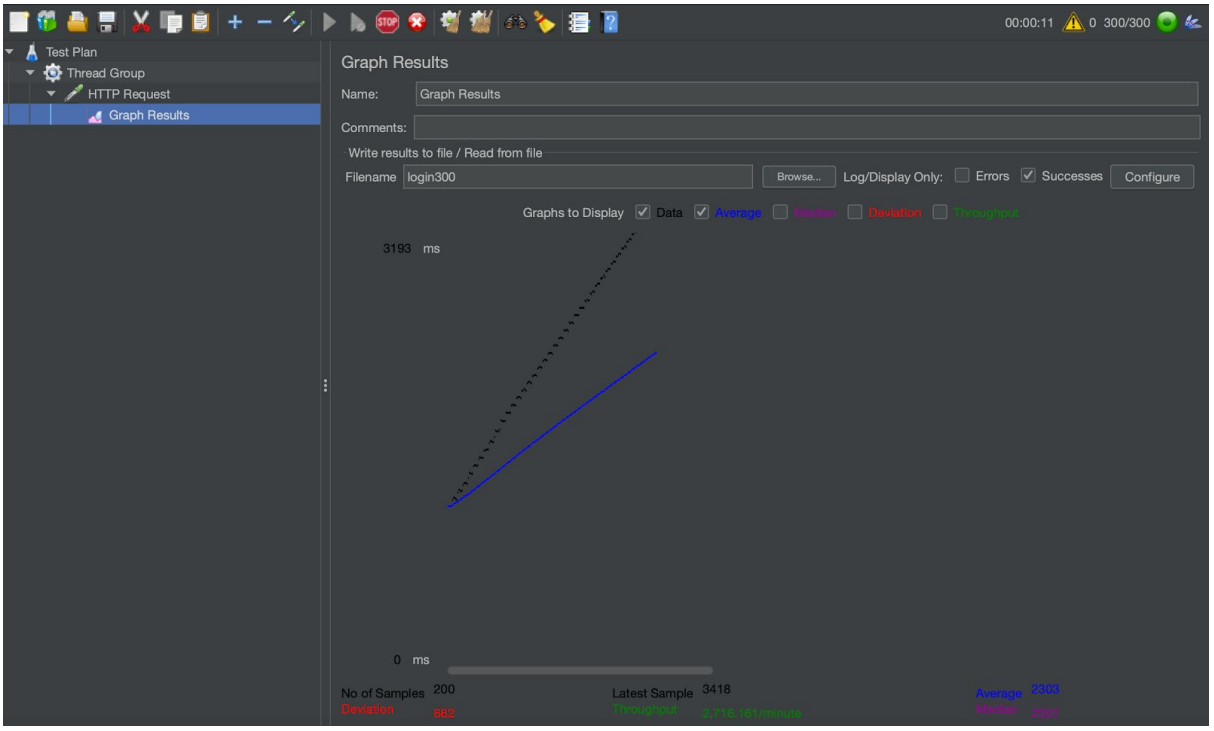
  100 Users Without Pooling



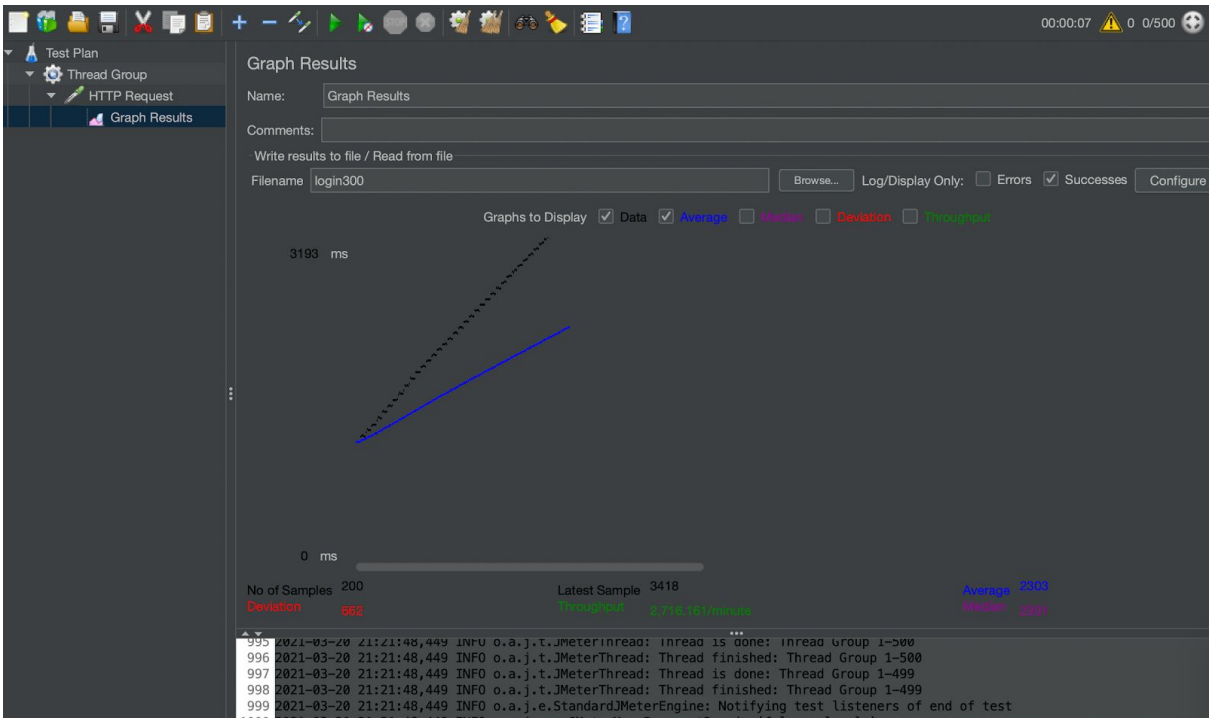
200 Users without pooling



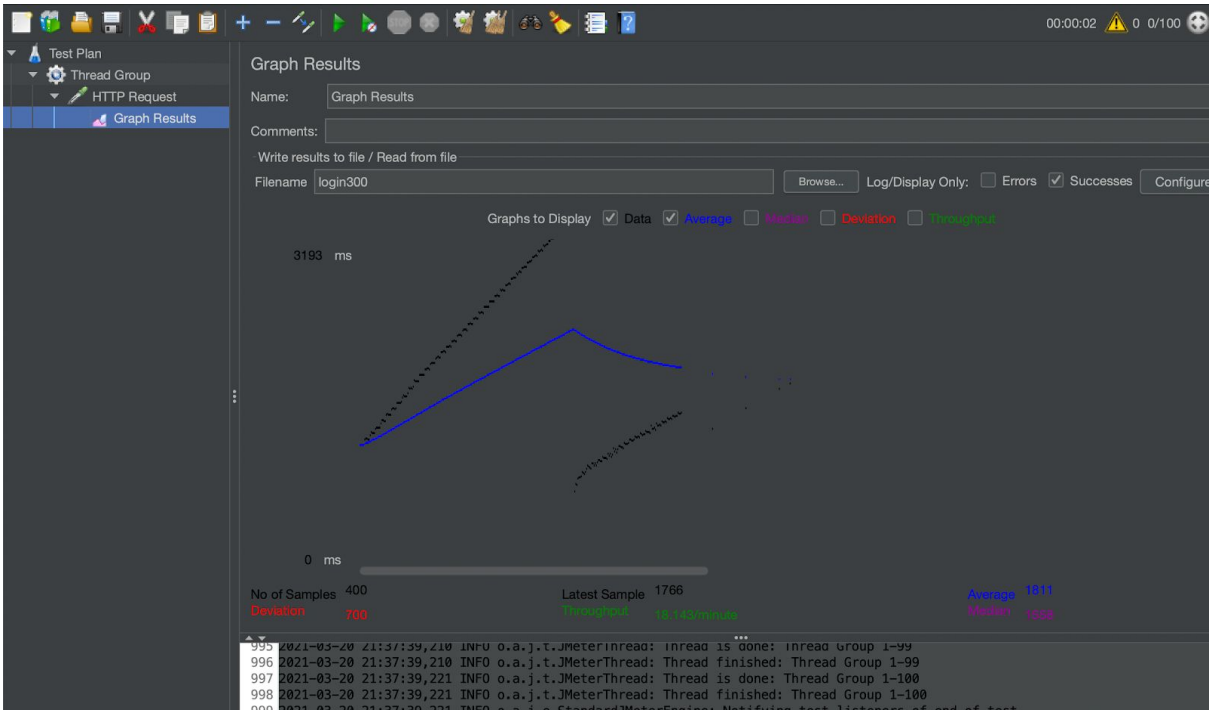
300 Users without pooling



400 user without pooling

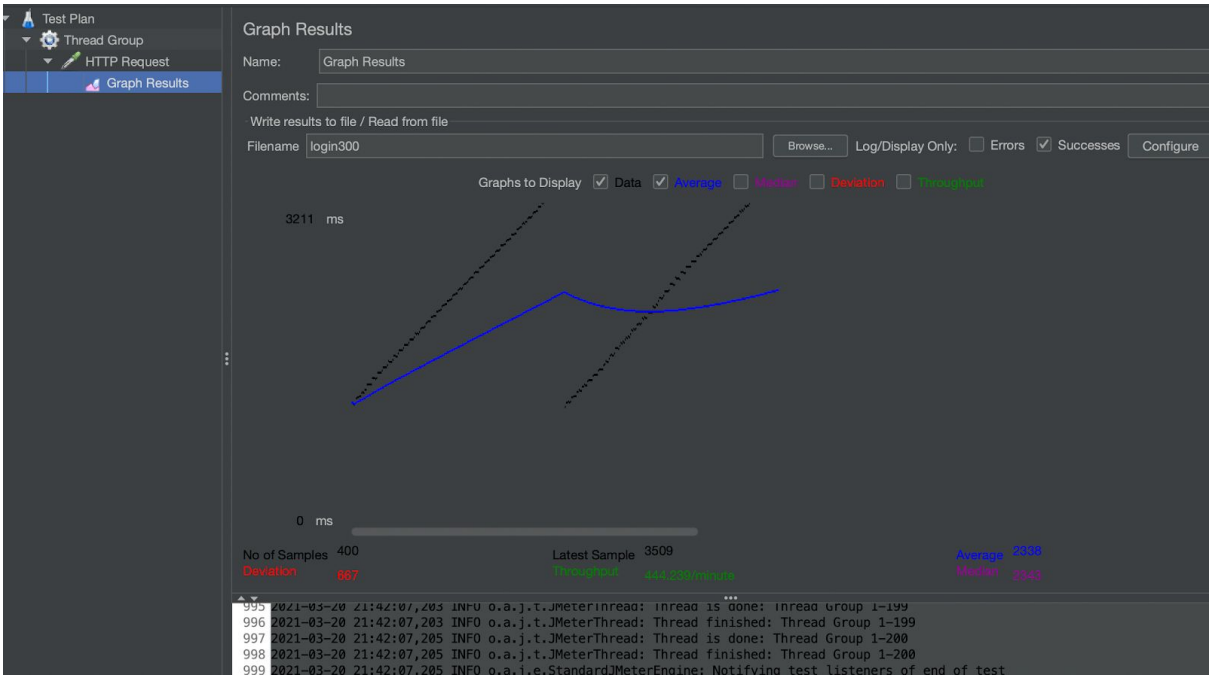


100 Users with pooling

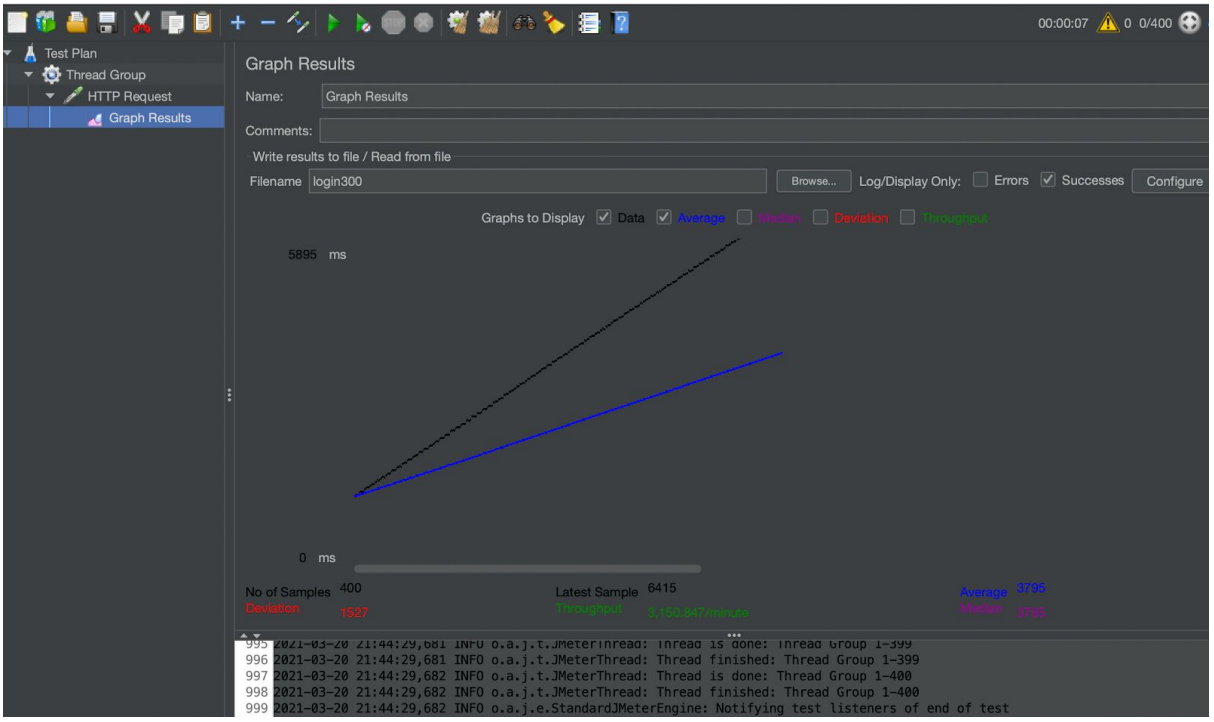


200 user with pooling

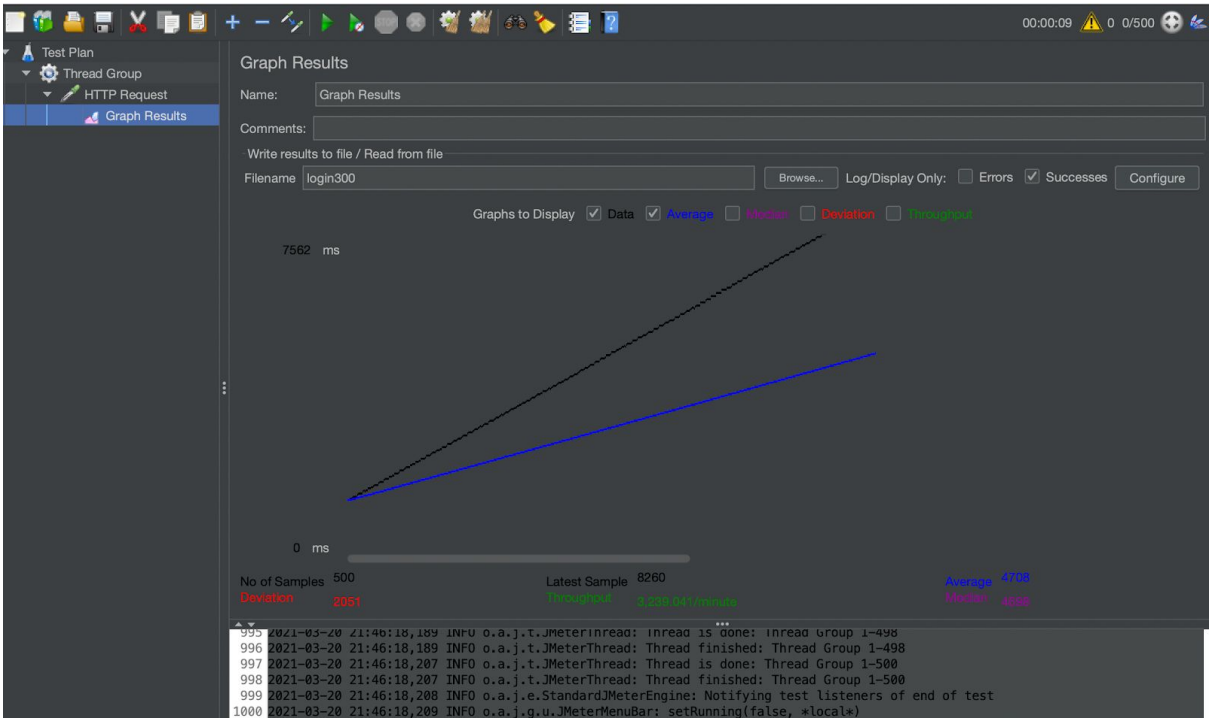
PRATIK KASLE



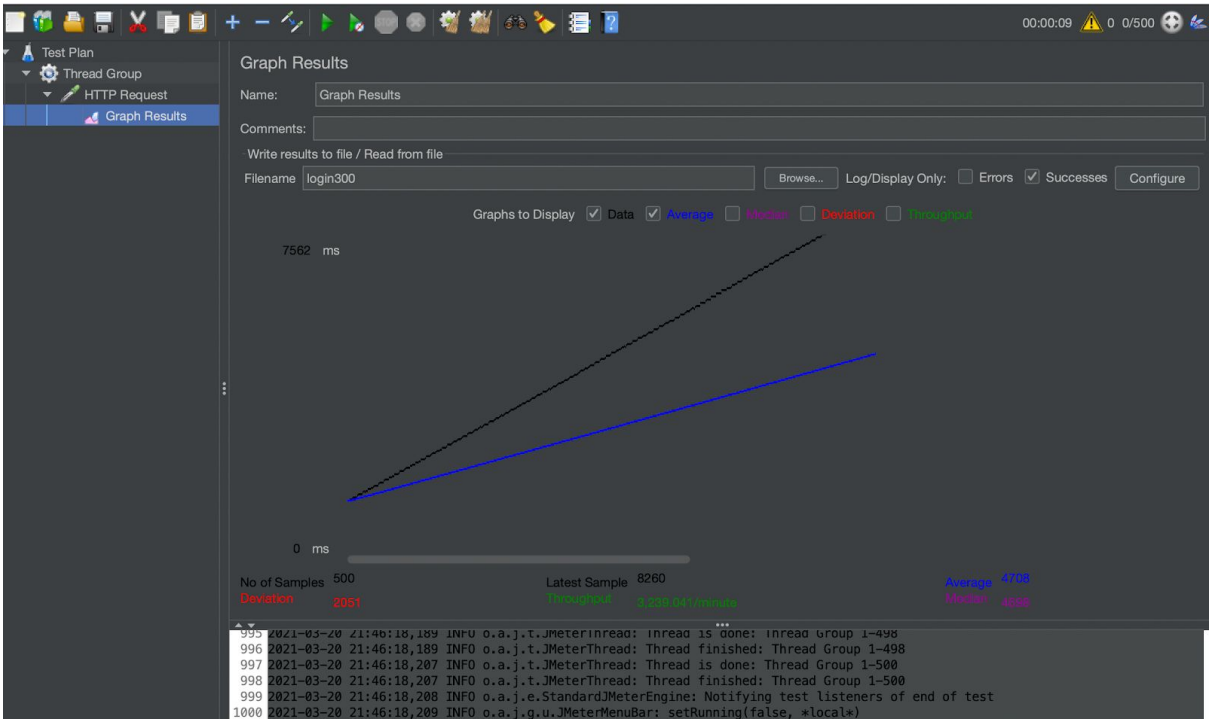
300 users with pooling



400 users with pooling



500 users with pooling



Answers to the questions:

1) To better understand the output of the load testing performed on Jmeter, here is a comparative table of the tests performed with different number of concurrent users:

threads	With pooling	Without pooling
100	18.142	2719
200	444	2716
300	3150	2716
400	3239	2716
500	3239	2700

Hence there is conclusive evidence that the load tests faired better in the case of connection pooling.
Connection pooling algorithm:
Data Structure to use: Stacks
=> The algorithm can be a simple usage of stacks. An active connection when demanded is popped from the top of the stack and when it is to be returned back to the pool because of being idle the connection thread is pushed back onto the stack.
=> This will keep the algorithm simple and connection pool highly available. Collections can be used so that the length of the stack can be dynamic.

2)Strategies for improving SQL performance:
a) Not using the * operator = specifying the columns to be fetched for the current query is more performant as it will fetch lesser data as opposed to fetching the whole table
b) Using indexes = using indexes especially for the fields on which we join increases the performance because to join SQL does a full table scan which the indexes reduce because it uses a hash table to index data
c) Avoiding nested joins = nested joins drastically exacerbate the performance of the query because they imply a cartesian product pattern. Instead use outer joins, inner joins, etc.