

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.

The screenshot shows the MongoDB Compass interface. On the left, there's a sidebar with a tree view of a database named 'splitwisedb' containing a collection named 'groups'. Under 'groups', there are four items: 'recentactivities', 'transactions', 'userconnections', and 'users'. The main area has tabs for 'Find', 'Indexes', 'Schema Anti-Patterns', 'Aggregation', and 'Search Indexes'. A search bar at the top contains the filter query: {"filter": "example"}. Below the search bar, it says 'QUERY RESULTS 1-2 OF 2'. A single document is listed with its JSON structure:

```

_id: ObjectId("60866f520269693be3de9b73")
name: "team event"
creatorId: "eddietaylor@gmail.com"
> users: Array
> expenses: Array
__v: 0

```

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

GroupId: 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**

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

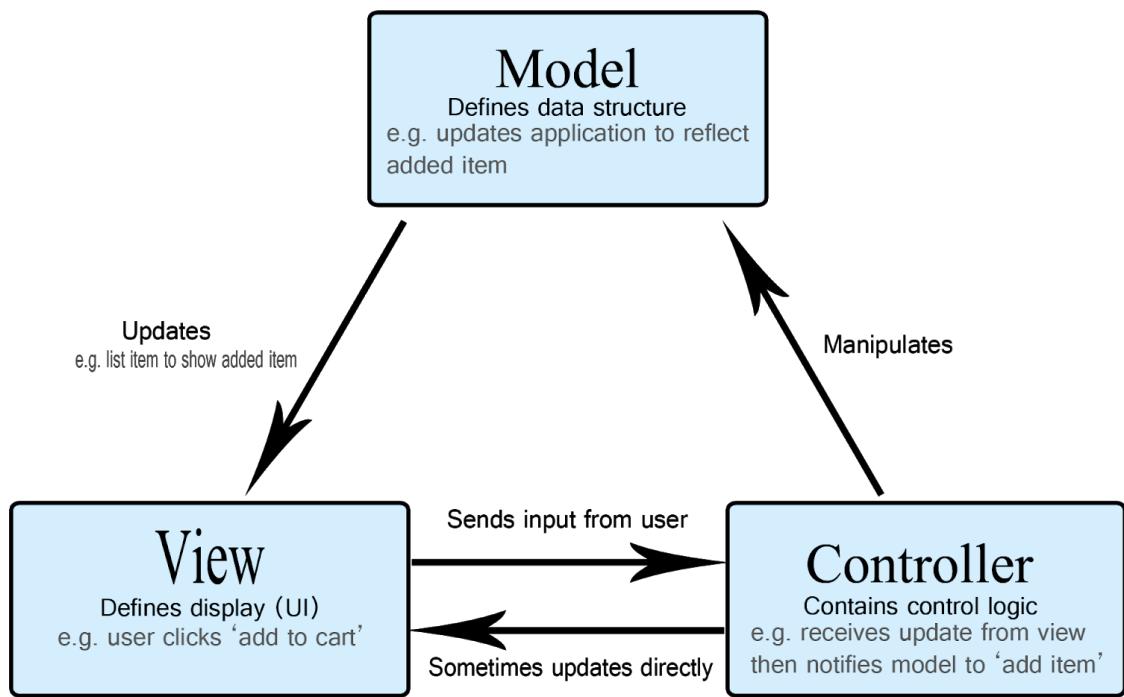
- **Recent-activity**

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

- **ORM(Object Relational Model Mongoose)**

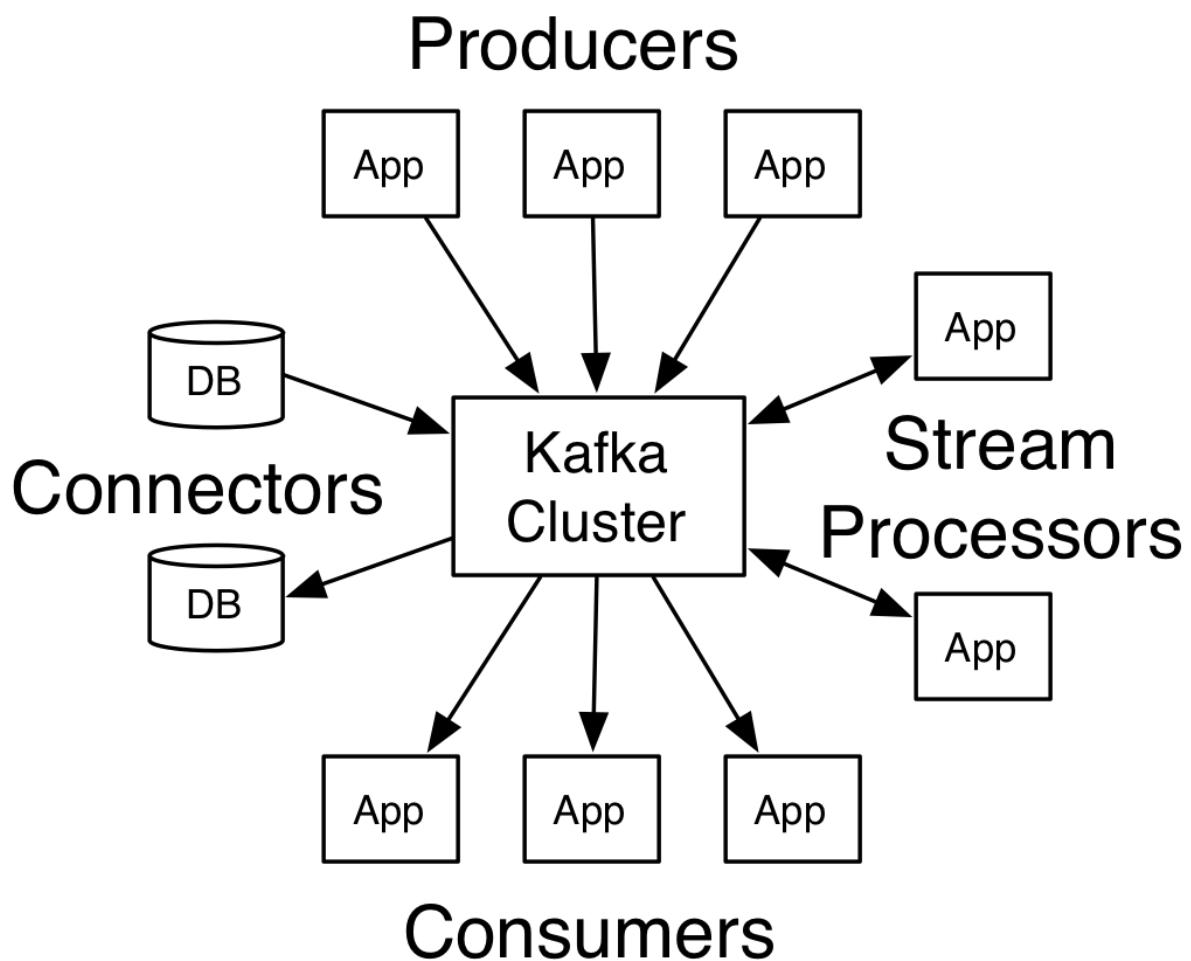
1. ORM converts a mongo into objects
2. Mongoose is a promise based node based ORM.

- **Backend Design:**



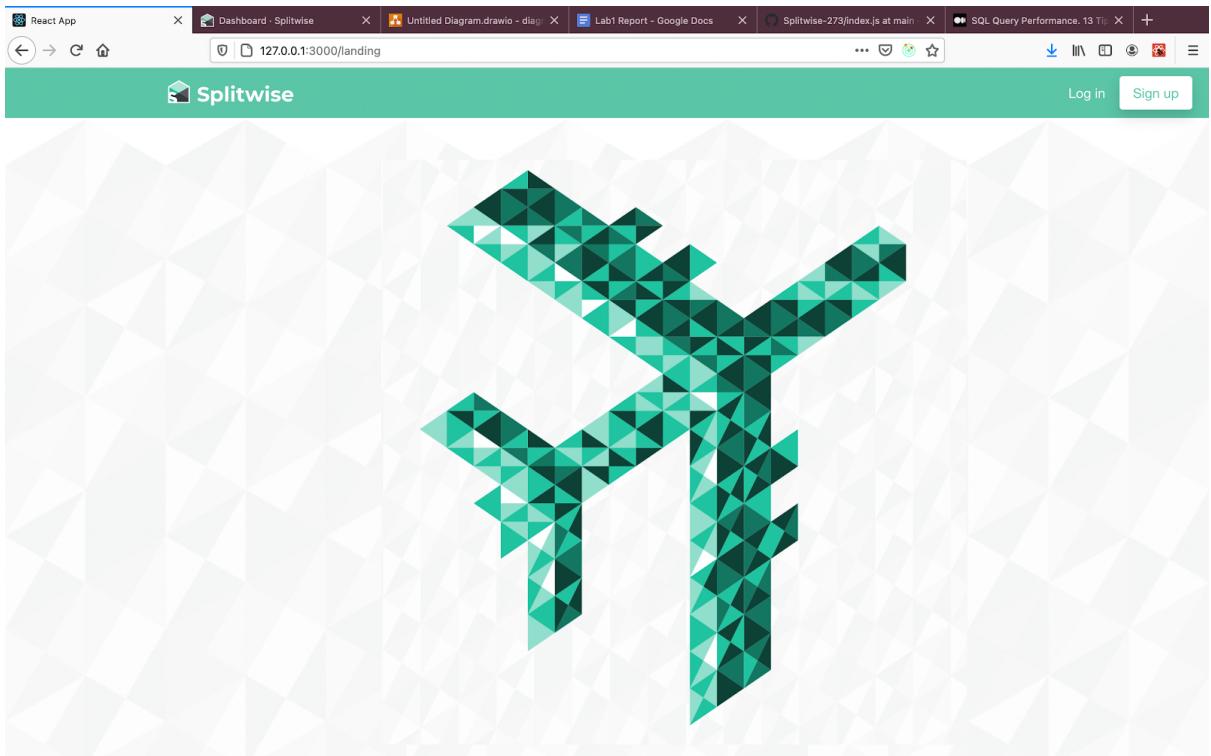
- A popular application architecture pattern MVC(Model Controller View) is used in this project.
- Model accesses the database and serves the data in object format.

KAFKA

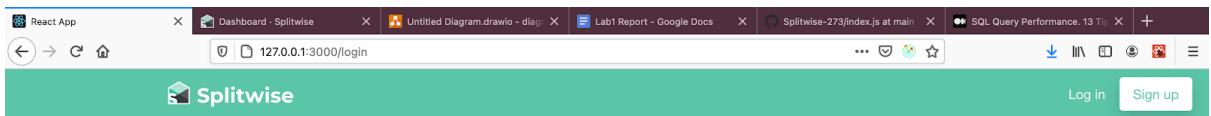


- Frontend UI:

Landing page:



Login:



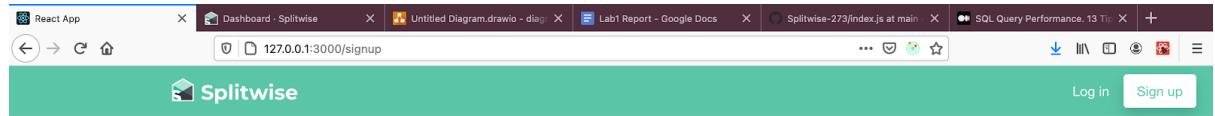
WELCOME TO SPLITWISE

Username

Password

Login

Register:



- Recent Activity:

Recent Acitivity

Show entries
10

Search

Showing 0 entries

Previous 1 Next

Settle up:

Settle up with your Friend!

Select...

Close Settle UP

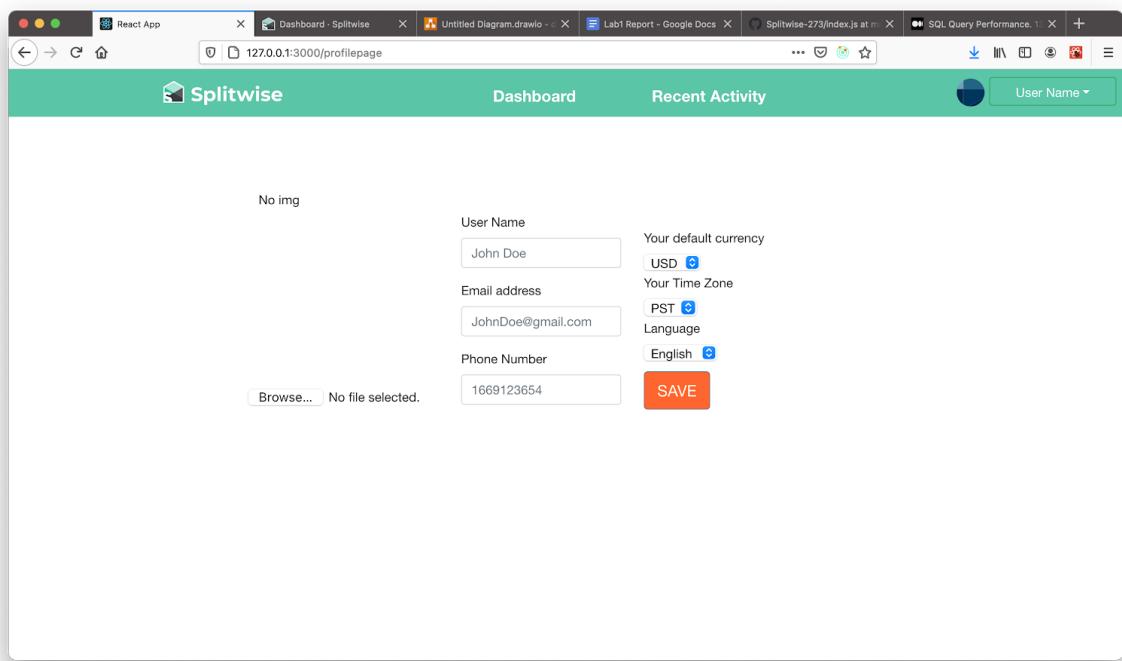
You Owe

- You Owe
- Chinmay

You Are Owed

- load for each grp

Profile page:



Dashboard:

The screenshot shows the Splitwise dashboard. At the top, there's a navigation bar with tabs for 'Dashboard' and 'Recent Activity'. A green header bar displays the title 'Splitwise' and a user profile icon with the text 'User Name'. Below the header, a large button labeled 'Settle Up Expenses' is visible. The main content area is titled 'Dashboard' and contains a section for a group named 'You owe blah blah'. This group has three members: 'You Owe', 'Chinmay', and 'You Owed'. A note below the group says 'load for each grp'.

Create group:

The screenshot shows the 'Create group' form on the Splitwise website. The title 'START A NEW GROUP' is at the top. Below it, a placeholder text 'My group shall be called...' is followed by a text input field containing 'HOME EXPENSES'. To the left of the input field is a stylized logo consisting of overlapping geometric shapes in teal, grey, and black. Below the input field is a section titled 'GROUP MEMBERS' with four dropdown menus labeled 'Select...'. Each dropdown has a 'Browse...' button and a note 'No file selected.' To the right of the dropdowns is a large orange 'SAVE' button.

My groups:

The screenshot shows a browser window with multiple tabs open at the top. The active tab is '127.0.0.1:3000/myGroups' under the title 'Splitwise - 273/index.js at main'. The page has a green header with the 'Splitwise' logo, 'Dashboard', 'Recent Activity', and a user profile icon. A blue button labeled 'Go to group' is visible. Below the header is a search bar with the placeholder 'Select...'. A table with columns 'Name' and 'Status' is shown, with a message 'No matching records found'. At the bottom, it says 'Showing 0 entries' with a page number '1' highlighted in blue.

Group Page:

The screenshot shows a simplified version of the 'Group Page'. It features a green header with the 'Splitwise' logo, 'Dashboard', 'Mygroup', 'Recent Activity', and a user profile icon for 'eddie'. A red button labeled 'Add an Expense' is at the top right. Below is a table with columns 'Expense description', 'Paid By', 'Amount', 'Date', 'Notes', and 'Add Note'. One row is present: 'pgne' paid by 'eddie' for '80' on '2021-04-26T08:24:42.283Z'. The 'Notes' column contains a note from 'oliver' and a reply from 'eddie'. The 'Add Note' column has a text input field and a 'Post' button.

Commit history:

- Commits on Apr 25, 2021
 - backend end bugs
PRKKILLER committed 5 hours ago
- Commits on Apr 23, 2021
 - feat: add profile redux
PRKKILLER committed 2 days ago
- Commits on Apr 21, 2021
 - redux login signup
PRKKILLER committed 4 days ago
 - frontend setup with redux
PRKKILLER committed 4 days ago
 - feat:notes route
PRKKILLER committed 5 days ago
 - bug: user connection groupname rewire
PRKKILLER committed 5 days ago
 - feat: recentactivity route
PRKKILLER committed 5 days ago
 - bug: solves repeated userconnection problem
PRKKILLER committed 5 days ago
- Commits on Apr 20, 2021
 - feat: add dashboard route
PRKKILLER committed 5 days ago
 - feat: individual group complete
PRKKILLER committed 5 days ago
 - feat: add Mygroup route
PRKKILLER committed 6 days ago
 - feat: add AddExpense route
PRKKILLER committed 6 days ago
- Commits on Apr 15, 2021
 - feat: add creategrouproute
PRKKILLER committed 10 days ago
 - feat: add user profile
PRKKILLER committed 10 days ago
- Commits on Apr 14, 2021
 - feat: jwt passport
PRKKILLER committed 11 days ago

The screenshot shows a GitHub commit history for a repository. It is organized into three main sections by date:

- Commits on Apr 12, 2021**: Contains two commits:
 - feat: signup** by PRKKILLER committed 13 days ago. Commit hash: 98eb5cf.
 - feat: setup kafka** by PRKKILLER committed 13 days ago. Commit hash: 6a7c569.
- Commits on Apr 8, 2021**: Contains eight commits:
 - clean** by PRKKILLER committed 17 days ago. Commit hash: 1995ace.
 - fix: gitignore change** by PRKKILLER committed 17 days ago. Commit hash: e5fd98a.
 - gitignore** by PRKKILLER committed 17 days ago. Commit hash: 9197f8e.
 - backend kafka setup** by PRKKILLER committed 17 days ago. Commit hash: 0548f0b.
 - connection kafka** by PRKKILLER committed 17 days ago. Commit hash: 4b64787.
 - ignore modules** by PRKKILLER committed 17 days ago. Commit hash: 79226ec.
 - dependencies** by PRKKILLER committed 17 days ago. Commit hash: 1c5800e.
- Commits on Apr 7, 2021**: Contains one commit:
 - kafka backend setup** by PRKKILLER committed 19 days ago. Commit hash: 21b8fab.

Below the commit history, there are navigation buttons: **Newer** and **Older**.

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/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`, `createdAt`, `updatedAt`) VALUES (?, ?, ?, ?, ?, ?);
POST /user/register 201 234.526 ms - 105
  ✓ It should create user data (240ms)

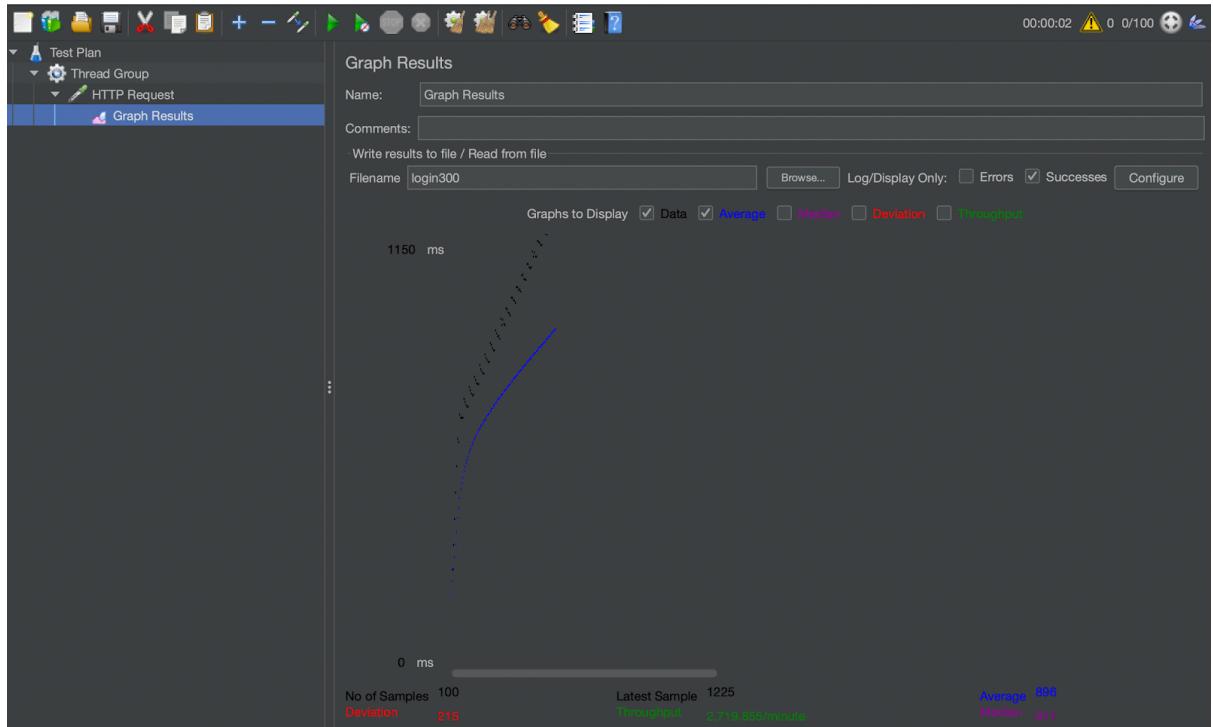
Update user profile
Executing (default): UPDATE `user_tables` SET `name`=? , `updatedAt`=? WHERE `user_id` = ?
POST /user/updateUserDetails 200 174.665 ms - 26
  ✓ It should update user data (184ms)

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)
```

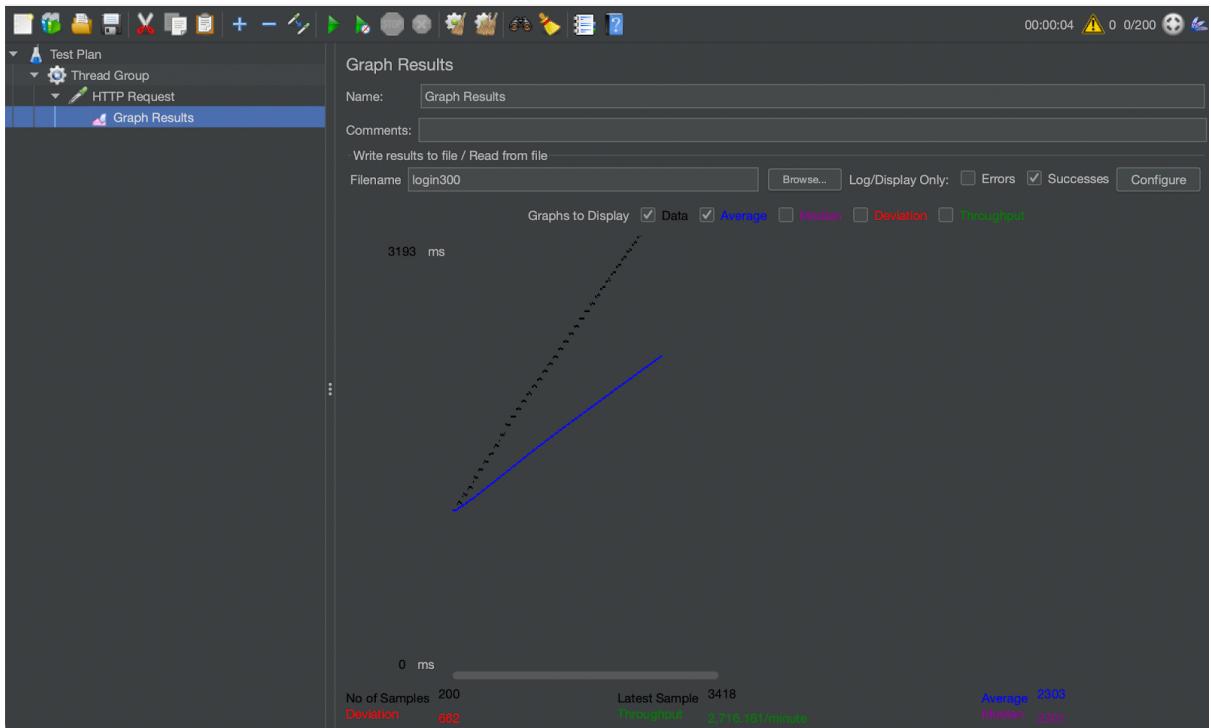


5 passing (2s)

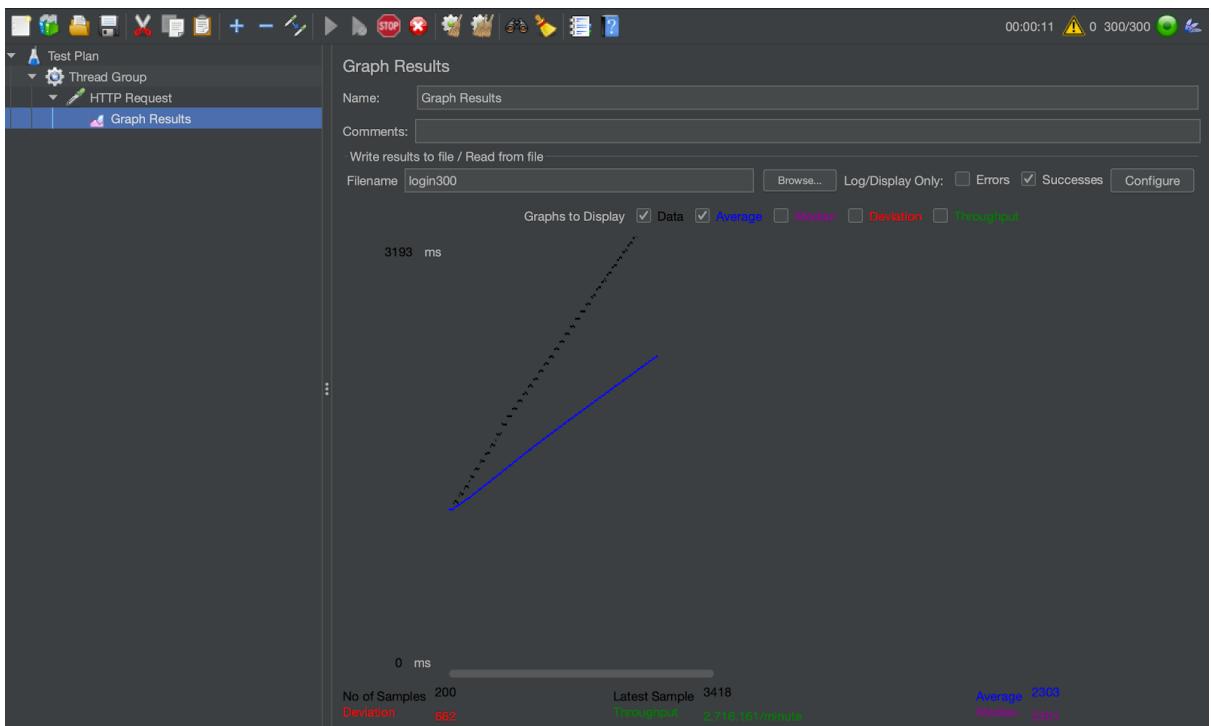
- **Jmeter Testing**
100 Users Without Pooling



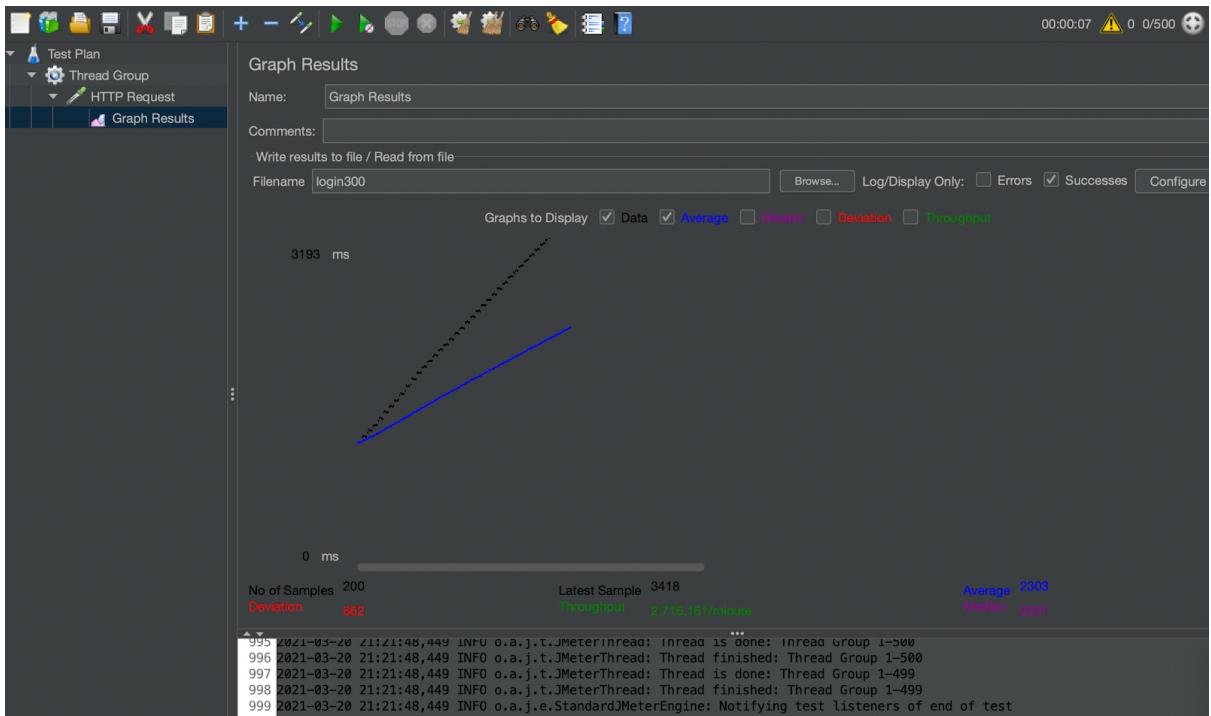
200 Users without pooling



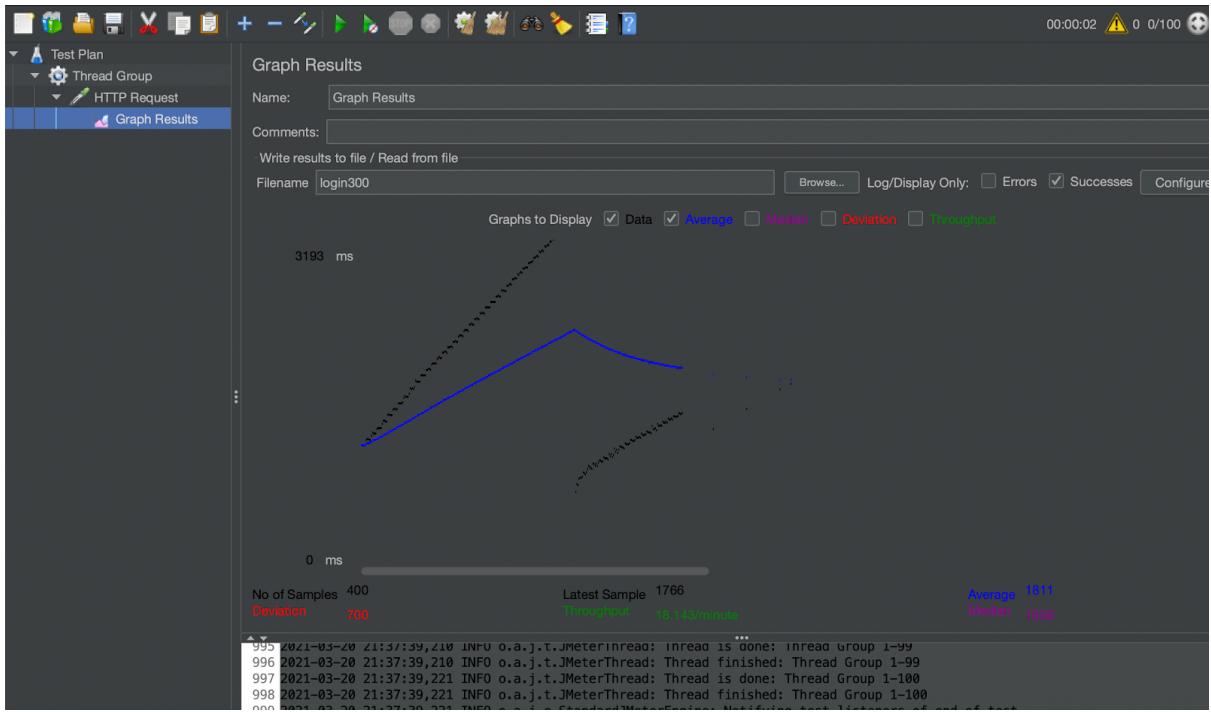
300 Users without pooling



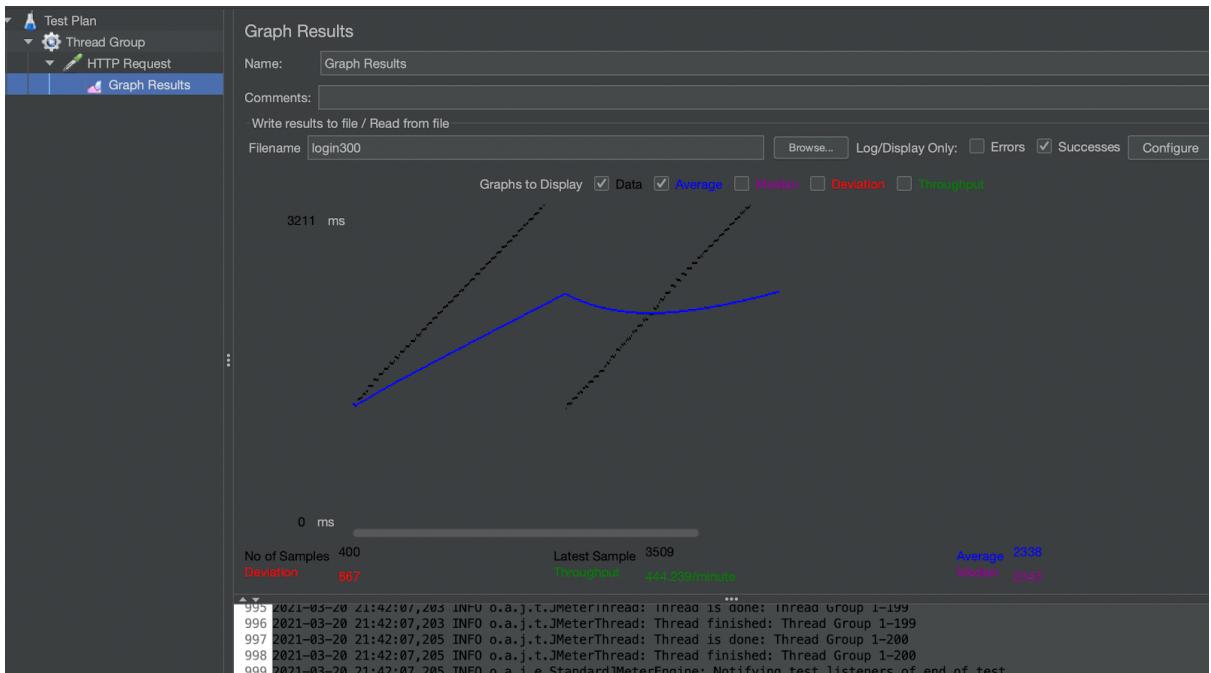
400 user without pooling



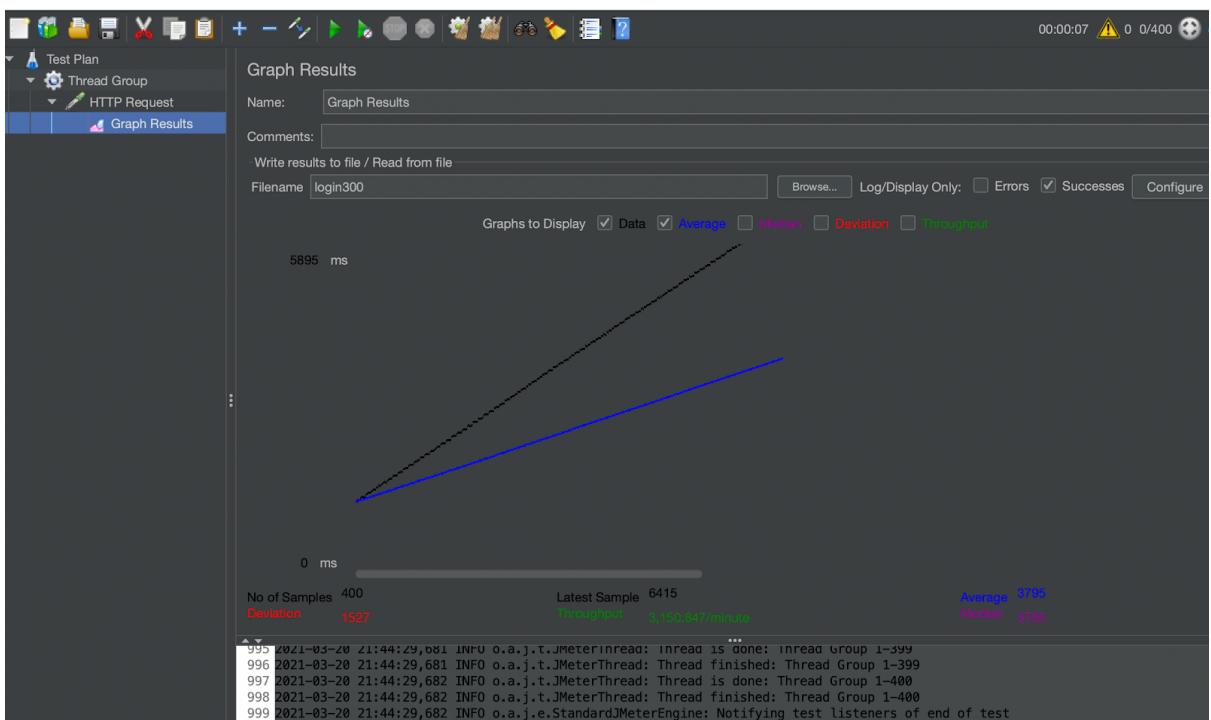
100 Users with pooling



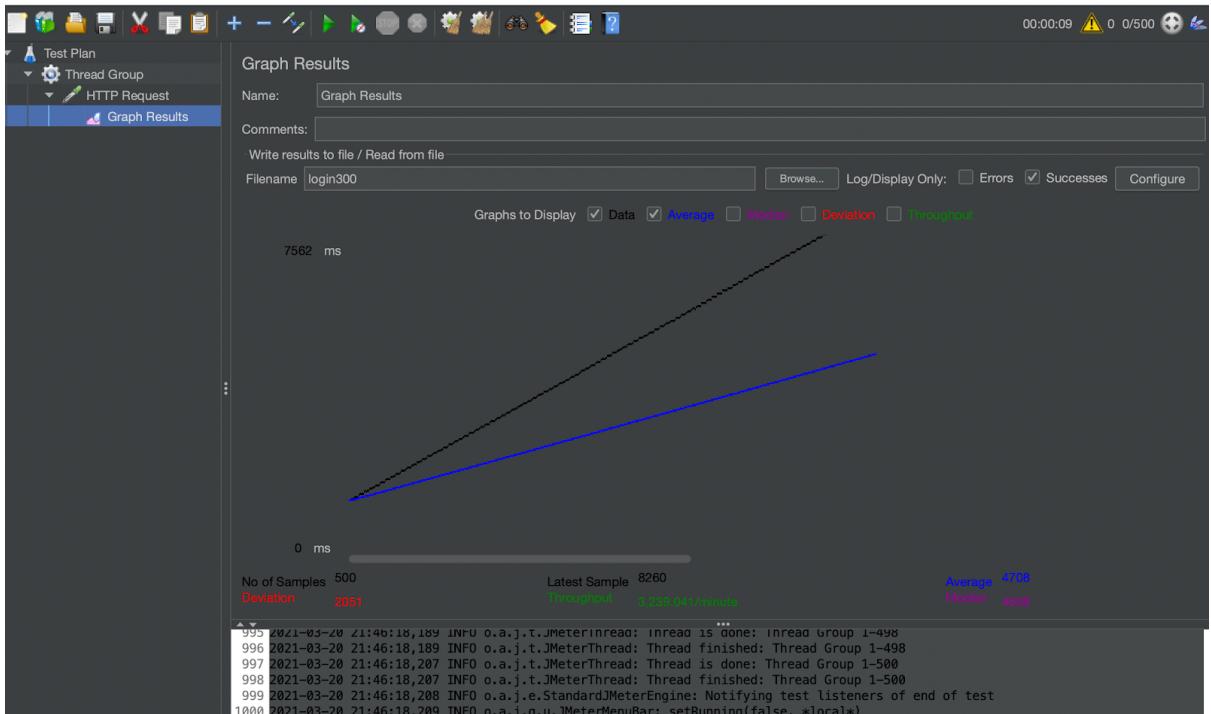
200 user with pooling



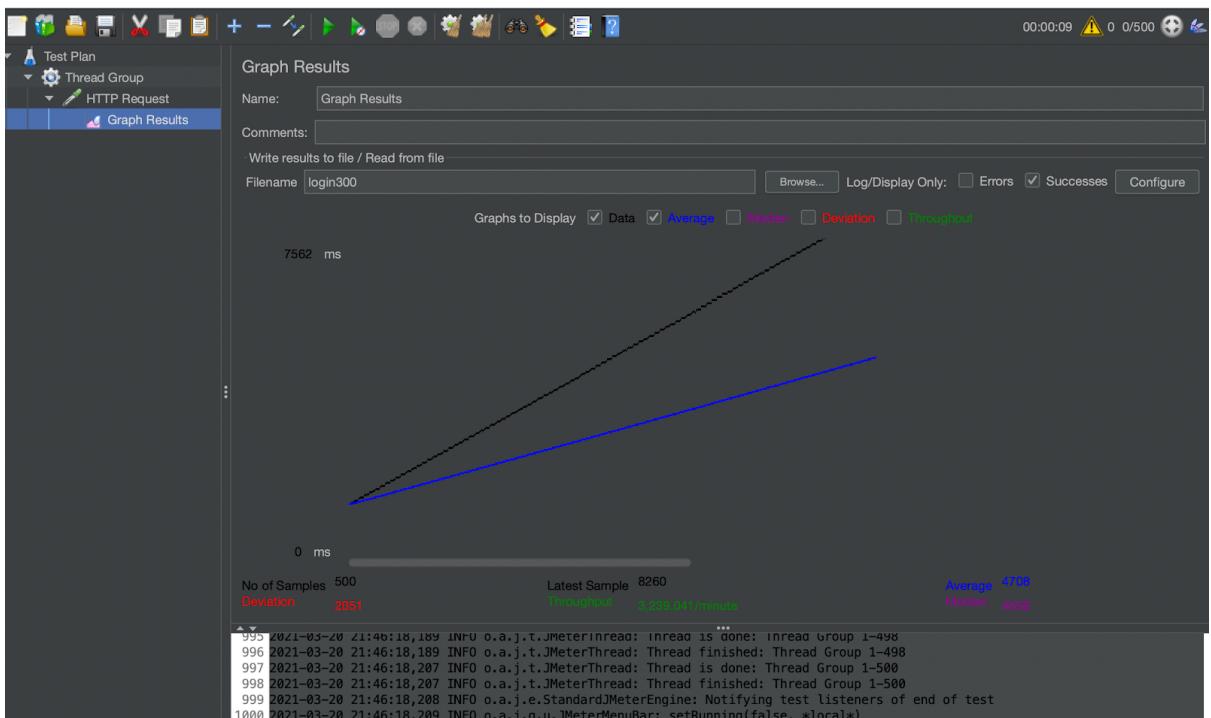
300 users with pooling



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 be 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.

