CMPE-273 Lab 3 Splitwise

Yusuf Juzar Soni

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:

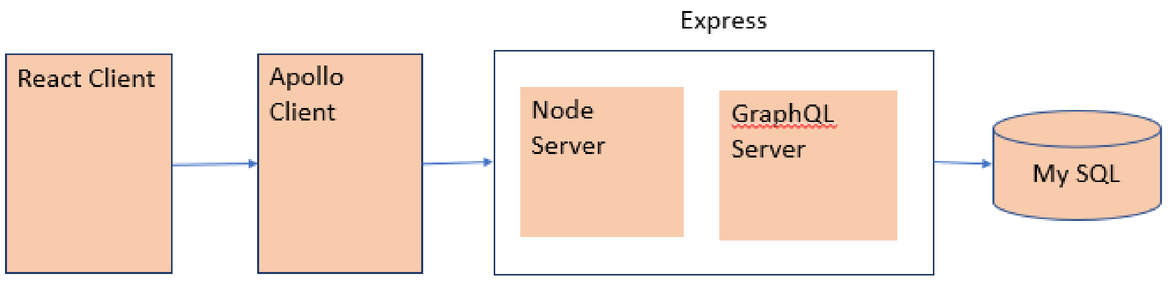
- React JS was used for frontend coding.
- Node was used as to implement the API layer etc.
- My SQL was used to implement the databases.
- GraphQL used instead of REST.

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.
- GraphQL was used instead of traditional REST API



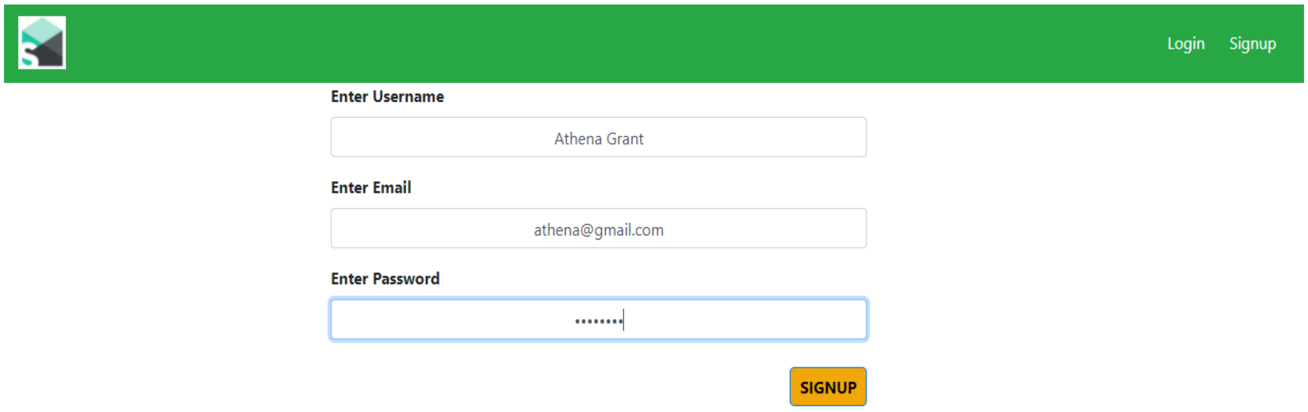
Database Schema:

- users (stores user elated information like username, <u>email</u> etc.) **Email is used as primary key and hence is used in all operations.**
- groups (stores <u>group name</u>, 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, <u>bill id</u>, 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.

Screenshots of various workflows are added below

Frontend Screenshots (Along with entries in database) :



Enter Username

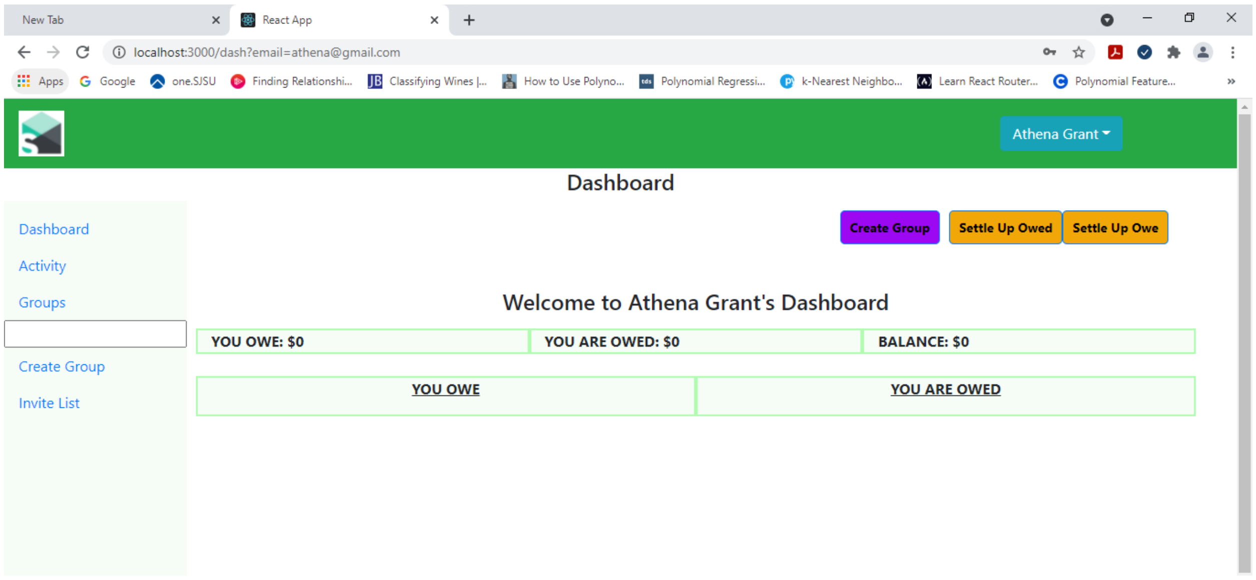
Athena Grant

Enter Email

athena@gmail.com

Enter Password

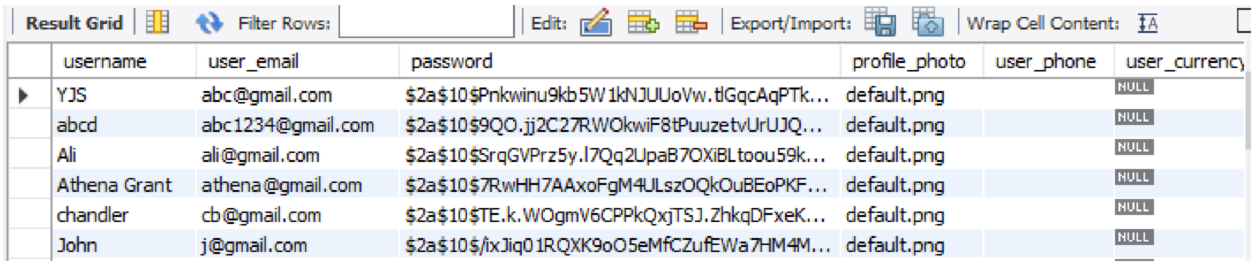
••••••••



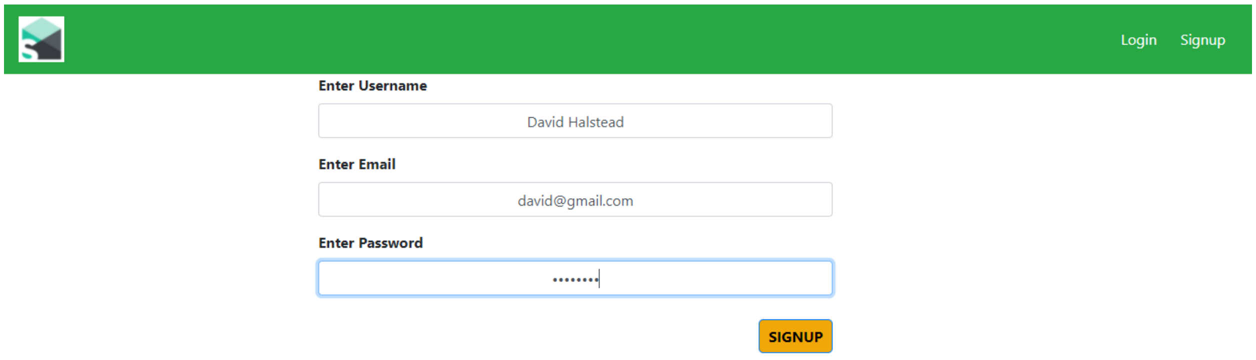
Welcome to Athena Grant's Dashboard

YOU OWE: $0	YOU ARE OWED: $0	BALANCE: $0

YOU OWE	YOU ARE OWED



username	user_email	password	profile_photo	user_phone	user_currency
YJS	abc@gmail.com	$2a$10$Pnkwinu9kb5W1kNJUUoVw.tlGqcAqPTk...	default.png		NULL
abcd	abc1234@gmail.com	$2a$10$9QO.jj2C27RWOkwiF8tPuuzetvUrUJQ...	default.png		NULL
Ali	ali@gmail.com	$2a$10$SrqGVPrz5y.l7Qq2UpaB7OXiBLtoou59k...	default.png		NULL
Athena Grant	athena@gmail.com	$2a$10$7RwHH7AAxoFgM4ULszOQkOuBEoPKF...	default.png		NULL
chandler	cb@gmail.com	$2a$10$TE.k.WOgmV6CPPkQxjTSJ.ZhkqDFxeK...	default.png		NULL
John	j@gmail.com	$2a$10$/ixJiq01RQXK9oO5eMfCZufEWa7HM4M...	default.png		NULL



Enter Username

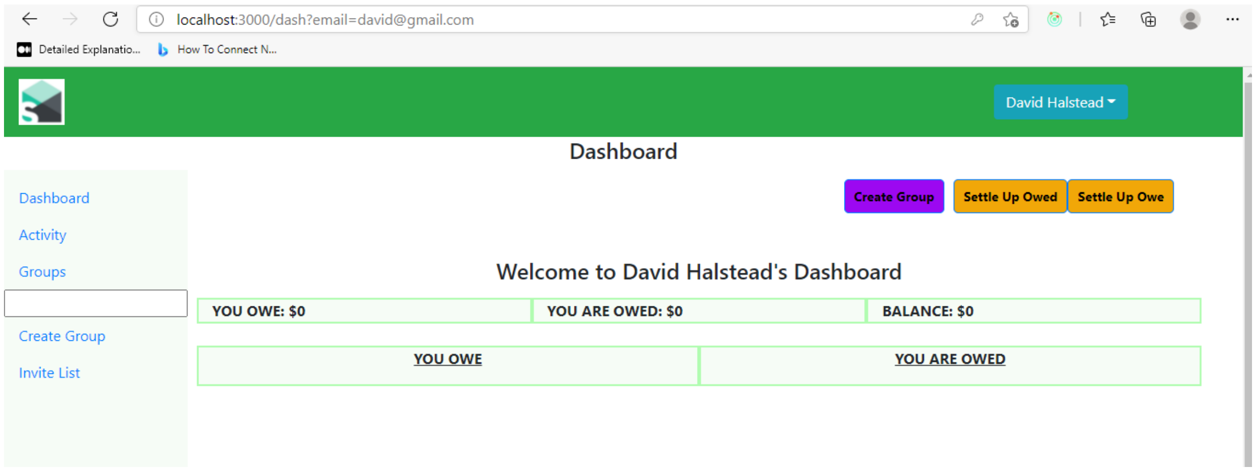
David Halstead

Enter Email

david@gmail.com

Enter Password

••••••••

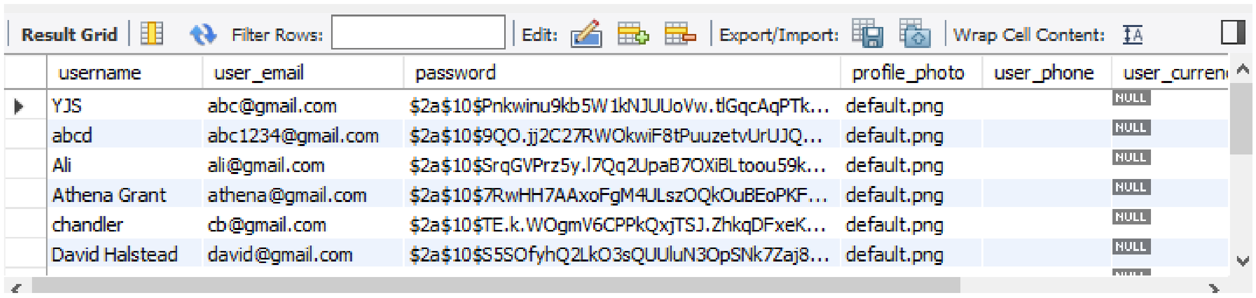


Dashboard

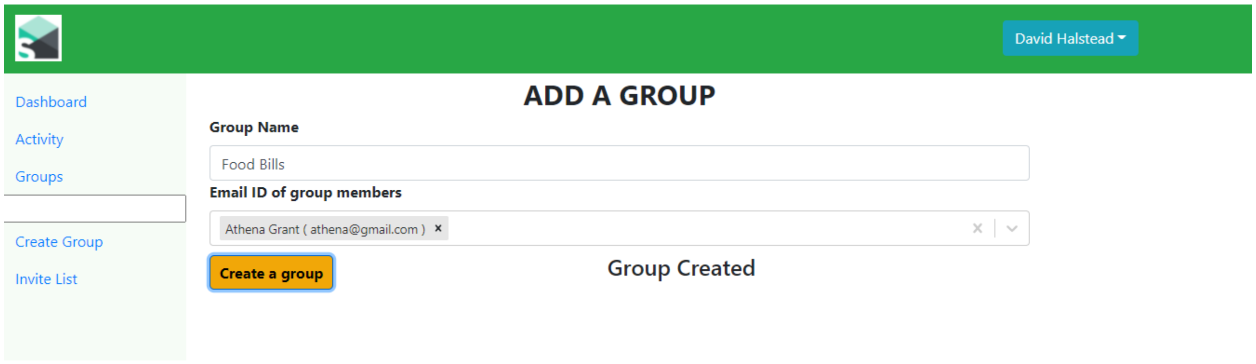
Welcome to David Halstead's Dashboard

YOU OWE: $0	YOU ARE OWED: $0	BALANCE: $0

YOU OWE	YOU ARE OWED



	username	user_email	password	profile_photo	user_phone	user_curren
▶	YJS	abc@gmail.com	$2a$10$Pnkwinu9kb5W1kNJUUoVw.tlGqcAqPTk...	default.png		NULL
	abcd	abc1234@gmail.com	$2a$10$9QO.jj2C27RWOkwiF8tPuuzetvUrUJQ...	default.png		NULL
	Ali	ali@gmail.com	$2a$10$SrqGVPrz5y.l7Qq2UpaB7OXiBLtoou59k...	default.png		NULL
	Athena Grant	athena@gmail.com	$2a$10$7RwHH7AAxoFgM4ULszOQkOuBEoPKF...	default.png		NULL
	chandler	cb@gmail.com	$2a$10$TE.k.WOgmV6CPPkQxjTSJ.ZhkqDFxeK...	default.png		NULL
	David Halstead	david@gmail.com	$2a$10$S5SOfyhQ2LkO3sQUUluN3OpSNk7Zaj8...	default.png		NULL



ADD A GROUP

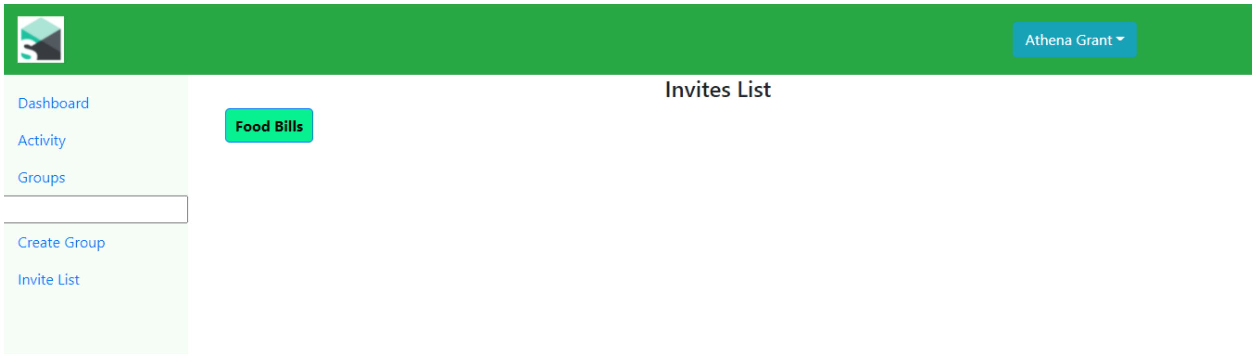
Group Name

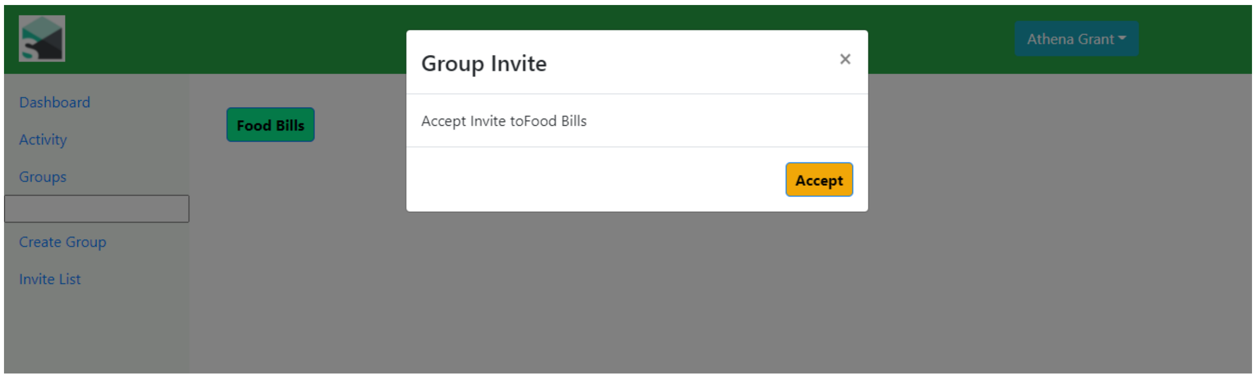
Food Bills

Email ID of group members

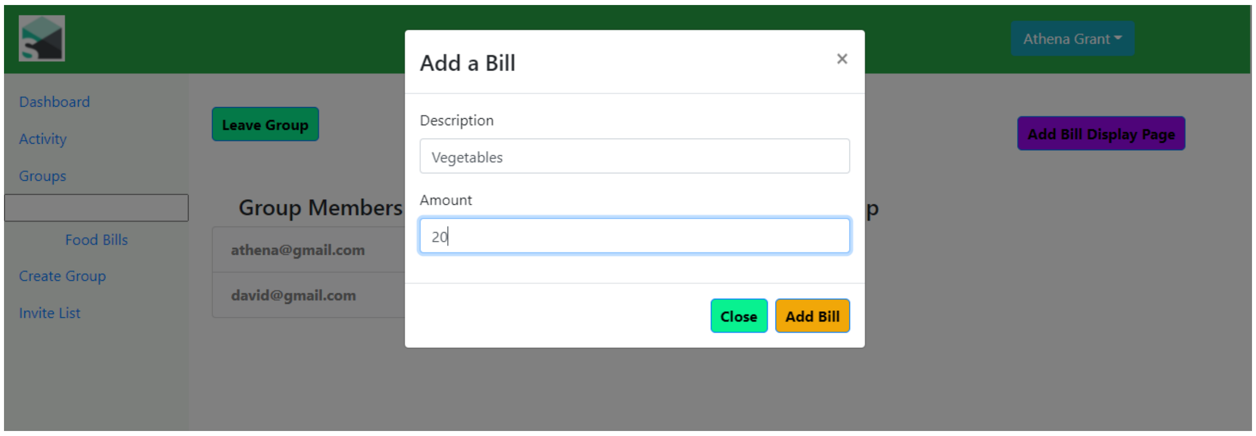
Athena Grant (athena@gmail.com) ×

Group Created



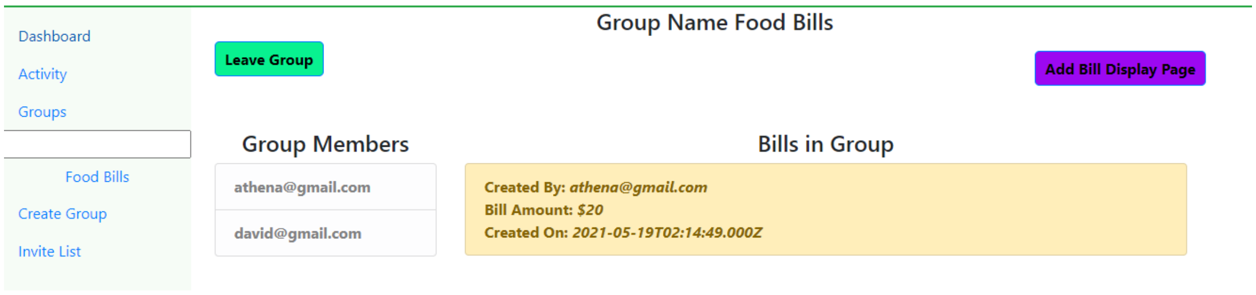

Group Invite

Accept Invite toFood Bills


Add a Bill

Description: Vegetables

Amount: 20


Group Name Food Bills

Group Members

athena@gmail.com

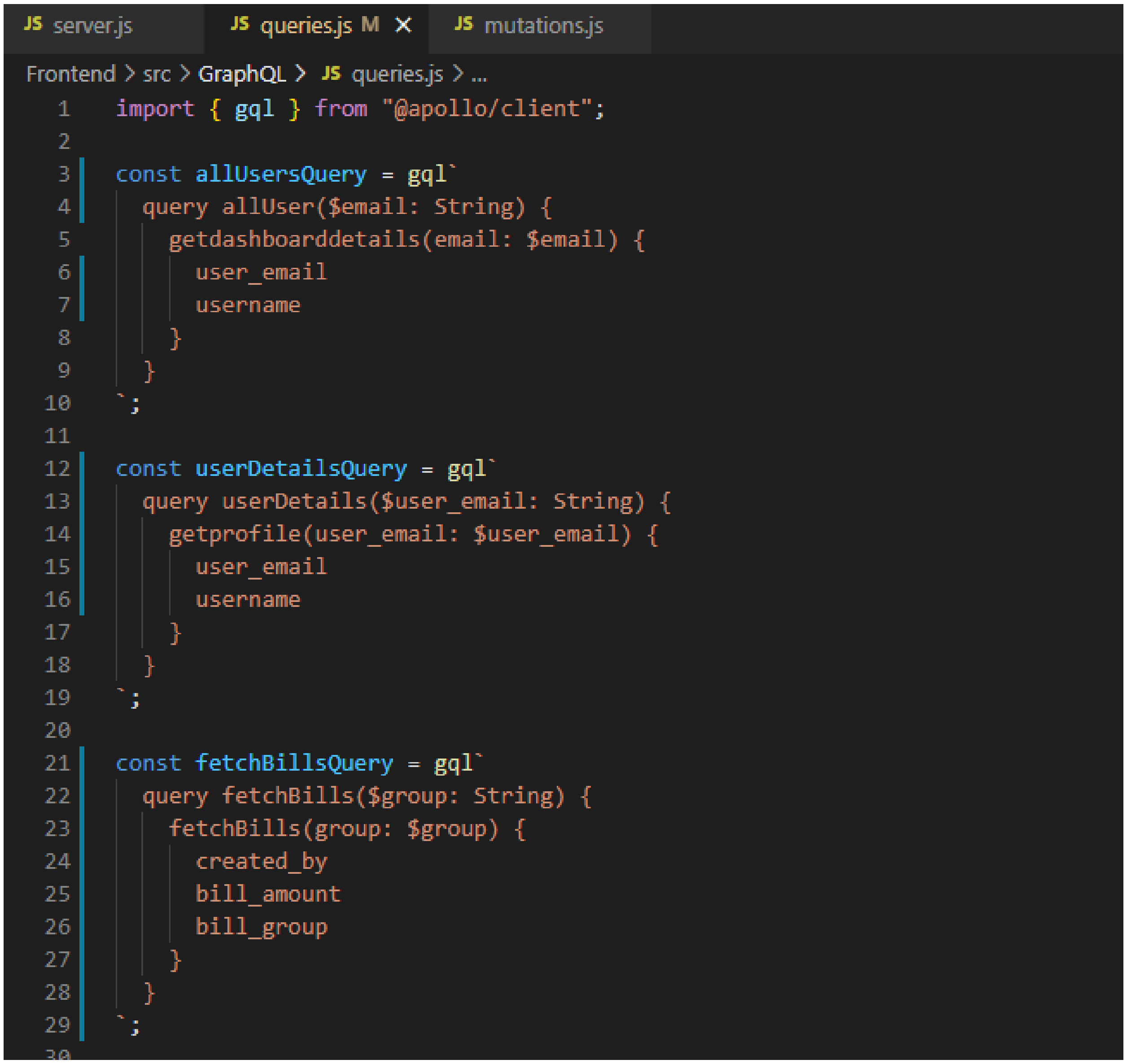
david@gmail.com

Bills in Group

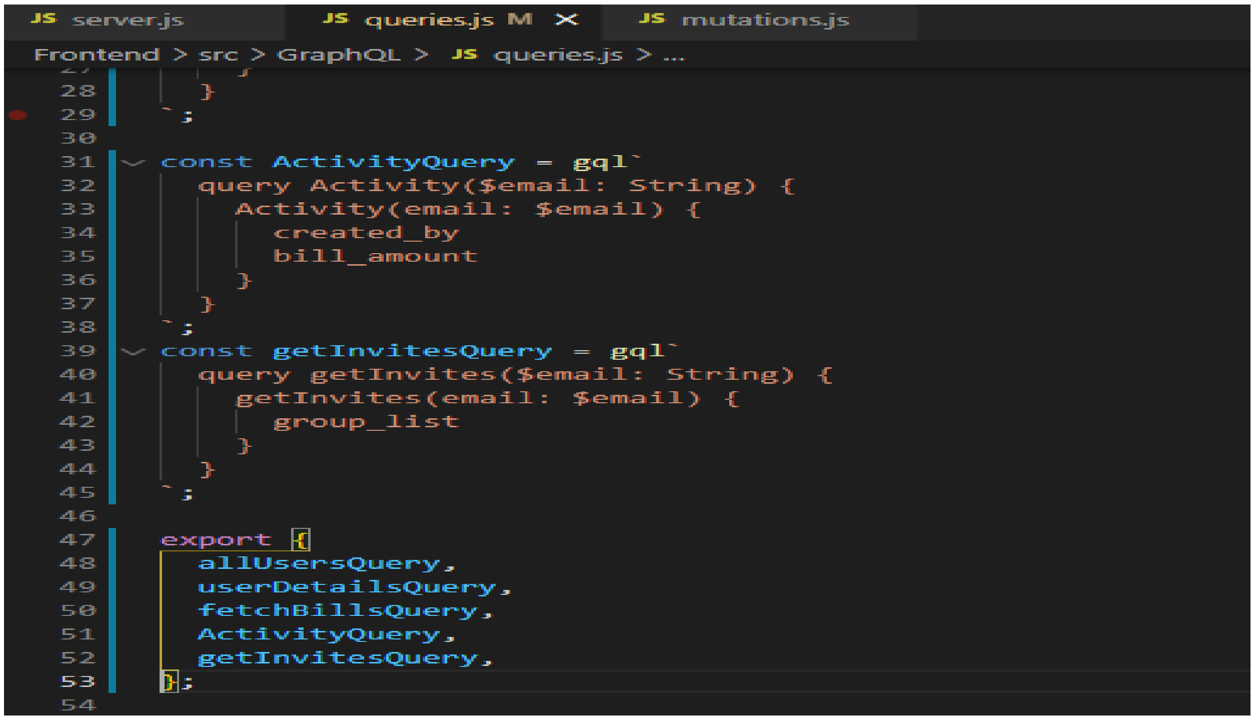
Created By: *athena@gmail.com*
Bill Amount: *$20*
Created On: *2021-05-19T02:14:49.000Z*

Screenshots of files with code:

Queries.js (Frontend)

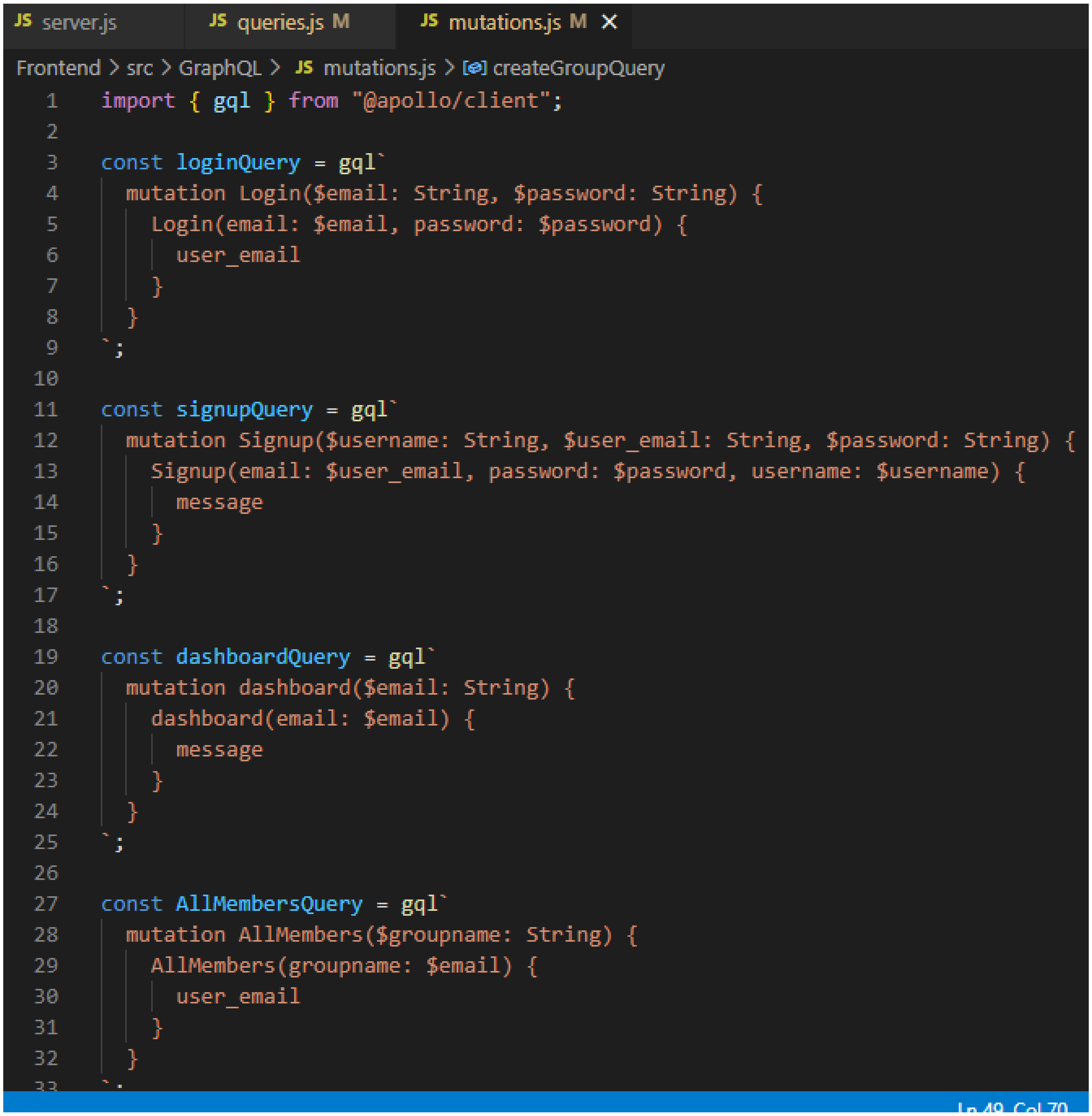


```
import { gql } from "@apollo/client";

const allUsersQuery = gql`
  query allUser($email: String) {
    getdashboarddetails(email: $email) {
      user_email
      username
    }
  }
`;

const userDetailsQuery = gql`
  query userDetails($user_email: String) {
    getprofile(user_email: $user_email) {
      user_email
      username
    }
  }
`;

const fetchBillsQuery = gql`
  query fetchBills($group: String) {
    fetchBills(group: $group) {
      created_by
      bill_amount
      bill_group
    }
  }
`;
```

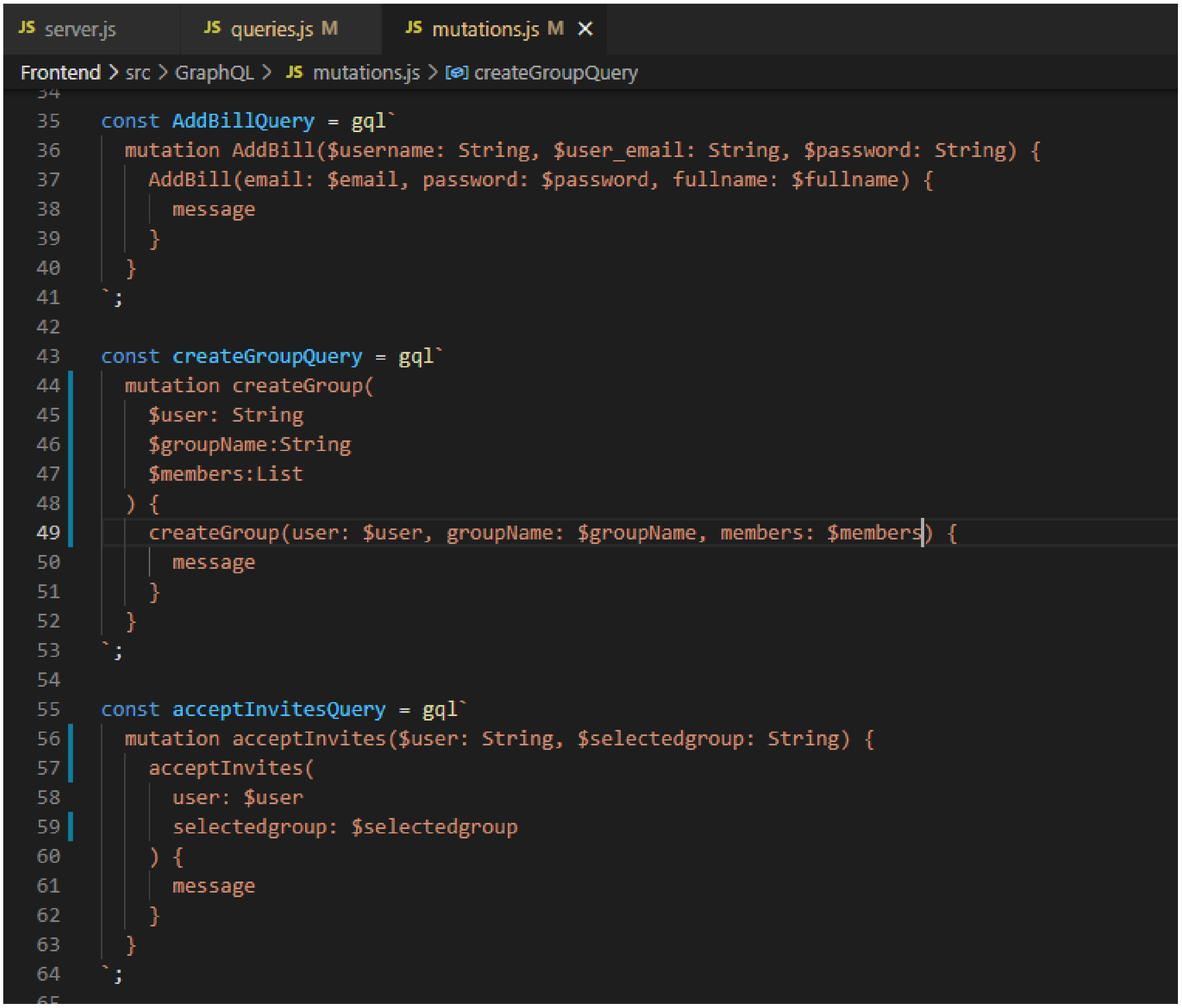


```
    }
  }
`;

const ActivityQuery = gql`
  query Activity($email: String) {
    Activity(email: $email) {
      created_by
      bill_amount
    }
  }
`;
const getInvitesQuery = gql`
  query getInvites($email: String) {
    getInvites(email: $email) {
      group_list
    }
  }
`;

export {
  allUsersQuery,
  userDetailsQuery,
  fetchBillsQuery,
  ActivityQuery,
  getInvitesQuery,
};
```

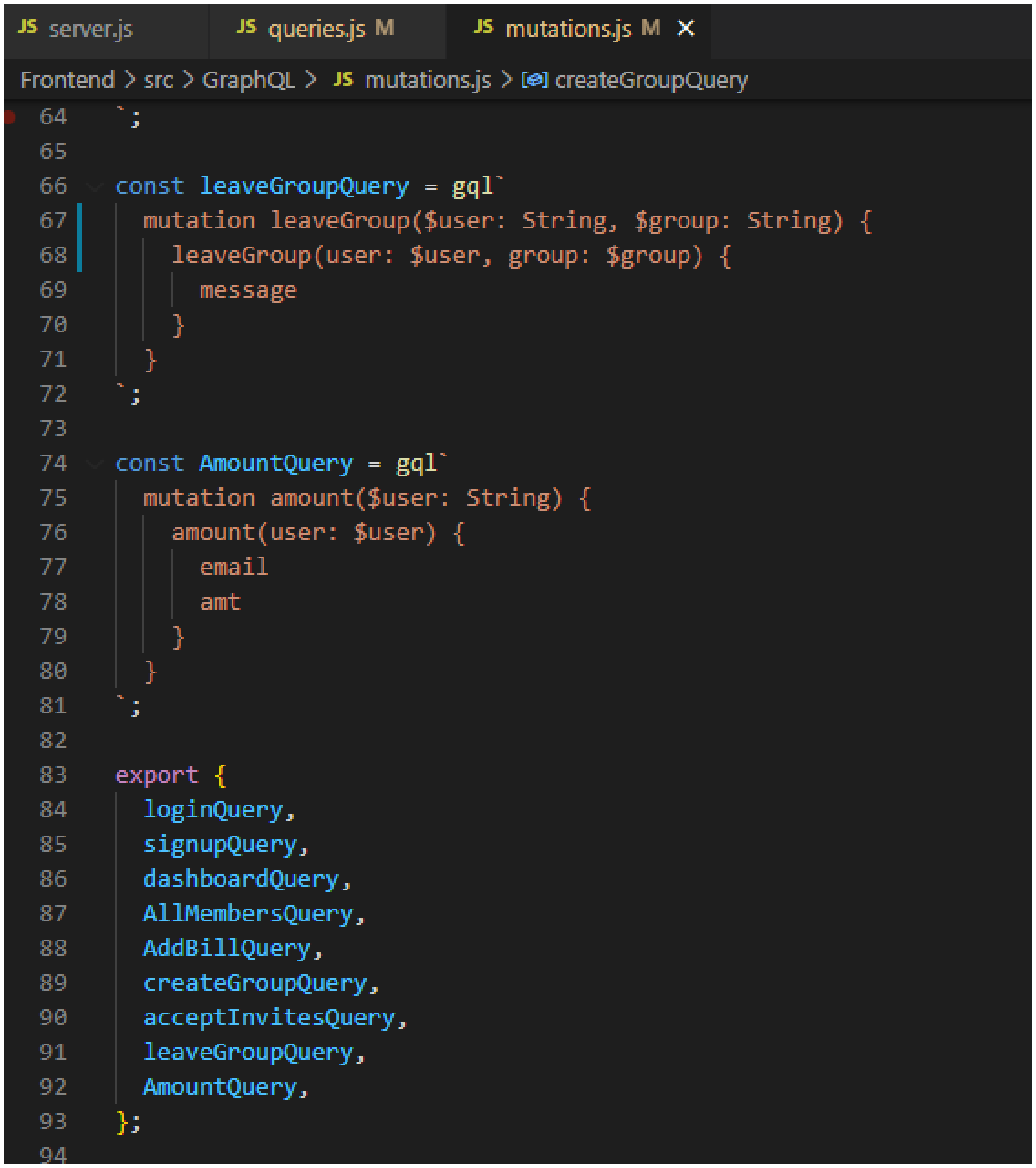
Mutations.js (Frontend File)



```
import { gql } from "@apollo/client";

const loginQuery = gql`
  mutation Login($email: String, $password: String) {
    Login(email: $email, password: $password) {
      user_email
    }
  }
`;

const signupQuery = gql`
  mutation Signup($username: String, $user_email: String, $password: String) {
    Signup(email: $user_email, password: $password, username: $username) {
      message
    }
  }
`;

const dashboardQuery = gql`
  mutation dashboard($email: String) {
    dashboard(email: $email) {
      message
    }
  }
`;

const AllMembersQuery = gql`
  mutation AllMembers($groupname: String) {
    AllMembers(groupname: $email) {
      user_email
    }
  }
```


JS server.js
JS queries.js M
JS mutations.js M ✕
Frontend > src > GraphQL > JS mutations.js > createGroupQuery

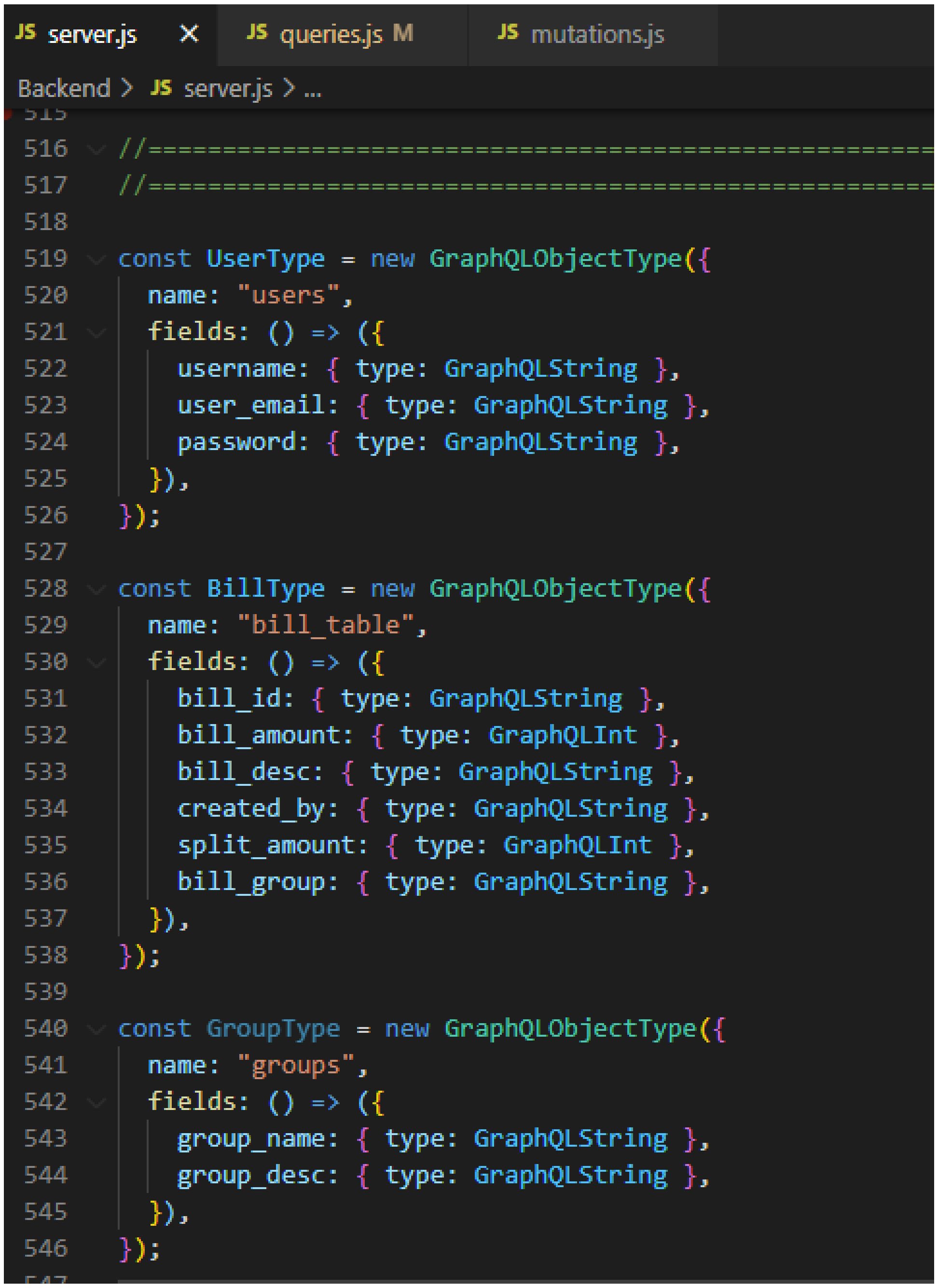
```
const AddBillQuery = gql`
  mutation AddBill($username: String, $user_email: String, $password: String) {
    AddBill(email: $email, password: $password, fullname: $fullname) {
      message
    }
  }
`;

const createGroupQuery = gql`
  mutation createGroup(
    $user: String
    $groupName:String
    $members:List
  ) {
    createGroup(user: $user, groupName: $groupName, members: $members) {
      message
    }
  }
`;

const acceptInvitesQuery = gql`
  mutation acceptInvites($user: String, $selectedgroup: String) {
    acceptInvites(
      user: $user
      selectedgroup: $selectedgroup
    ) {
      message
    }
  }
`;
```


server.js | queries.js M | mutations.js M

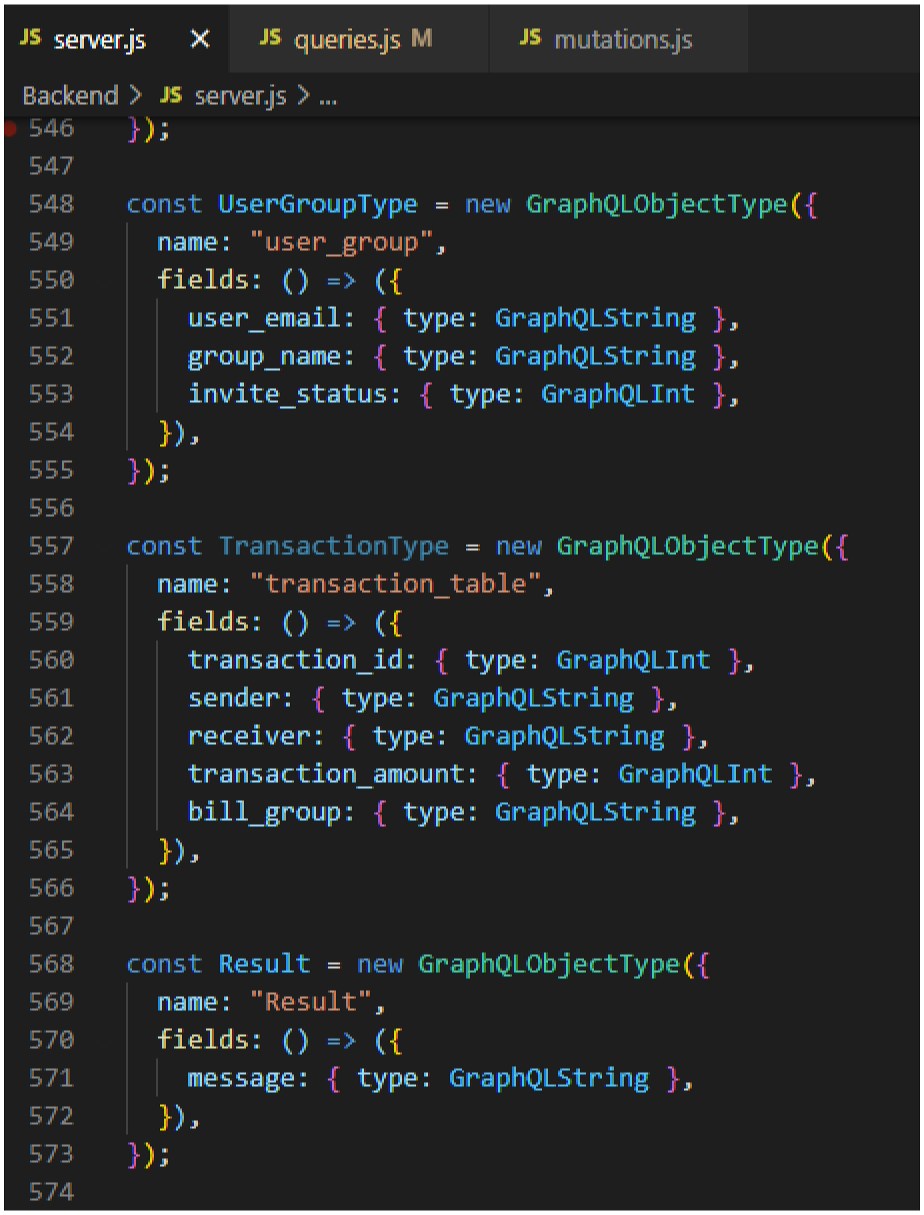
Frontend > src > GraphQL > mutations.js > createGroupQuery

```
`;

const leaveGroupQuery = gql`
  mutation leaveGroup($user: String, $group: String) {
    leaveGroup(user: $user, group: $group) {
      message
    }
  }
`;

const AmountQuery = gql`
  mutation amount($user: String) {
    amount(user: $user) {
      email
      amt
    }
  }
`;

export {
  loginQuery,
  signupQuery,
  dashboardQuery,
  AllMembersQuery,
  AddBillQuery,
  createGroupQuery,
  acceptInvitesQuery,
  leaveGroupQuery,
  AmountQuery,
};
```

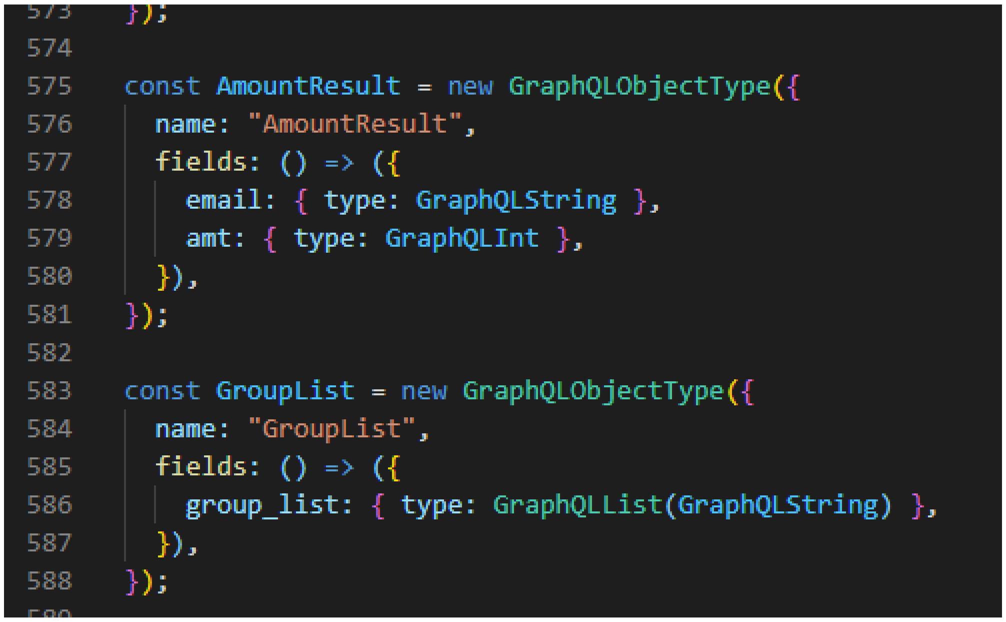
Screenshot of GraphQL Schema in backend:



```
//=====================================================
//=====================================================

const UserType = new GraphQLObjectType({
  name: "users",
  fields: () => ({
    username: { type: GraphQLString },
    user_email: { type: GraphQLString },
    password: { type: GraphQLString },
  }),
});

const BillType = new GraphQLObjectType({
  name: "bill_table",
  fields: () => ({
    bill_id: { type: GraphQLString },
    bill_amount: { type: GraphQLInt },
    bill_desc: { type: GraphQLString },
    created_by: { type: GraphQLString },
    split_amount: { type: GraphQLInt },
    bill_group: { type: GraphQLString },
  }),
});

const GroupType = new GraphQLObjectType({
  name: "groups",
  fields: () => ({
    group_name: { type: GraphQLString },
    group_desc: { type: GraphQLString },
  }),
});
```

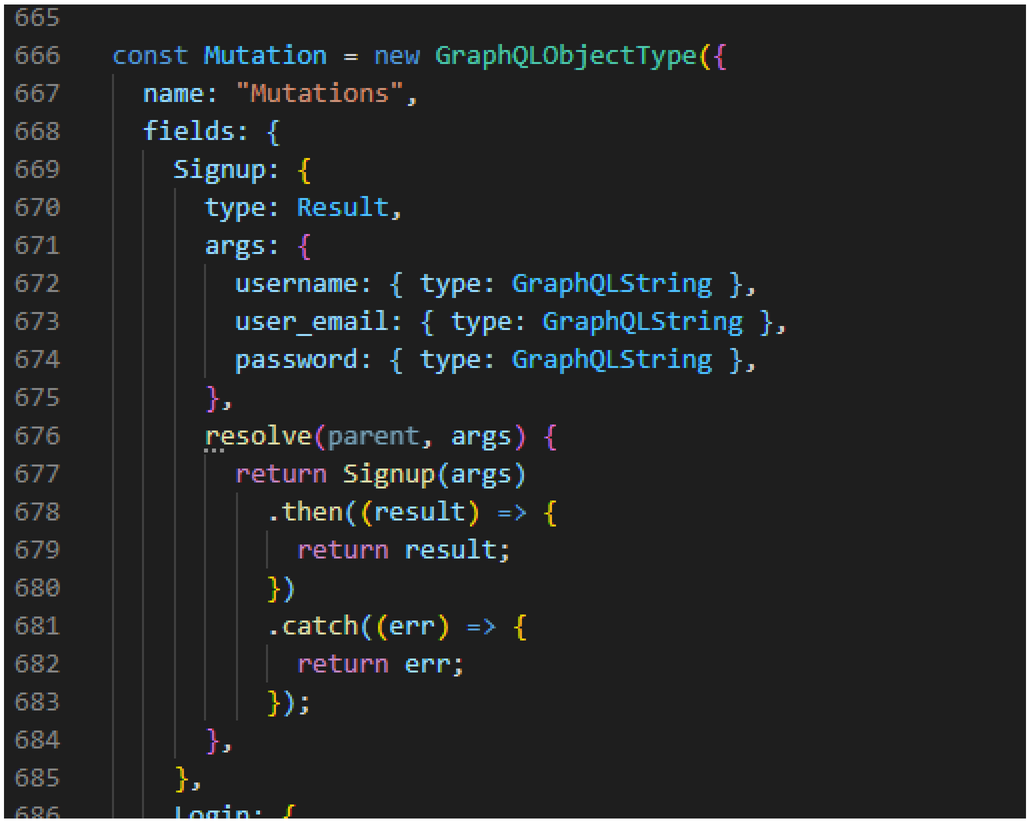


```
});

const UserGroupType = new GraphQLObjectType({
  name: "user_group",
  fields: () => ({
    user_email: { type: GraphQLString },
    group_name: { type: GraphQLString },
    invite_status: { type: GraphQLInt },
  }),
});

const TransactionType = new GraphQLObjectType({
  name: "transaction_table",
  fields: () => ({
    transaction_id: { type: GraphQLInt },
    sender: { type: GraphQLString },
    receiver: { type: GraphQLString },
    transaction_amount: { type: GraphQLInt },
    bill_group: { type: GraphQLString },
  }),
});

const Result = new GraphQLObjectType({
  name: "Result",
  fields: () => ({
    message: { type: GraphQLString },
  }),
});

```

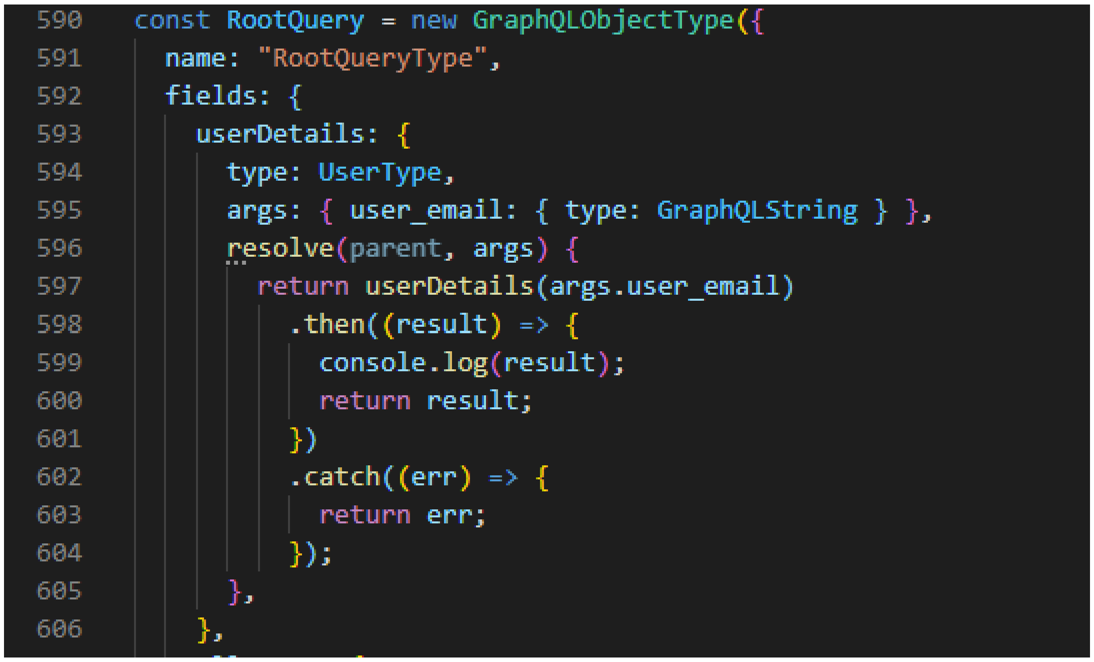


```
});

const AmountResult = new GraphQLObjectType({
  name: "AmountResult",
  fields: () => ({
    email: { type: GraphQLString },
    amt: { type: GraphQLInt },
  }),
});

const GroupList = new GraphQLObjectType({
  name: "GroupList",
  fields: () => ({
    group_list: { type: GraphQLList(GraphQLString) },
  }),
});
```

GraphQL Backend implementation:



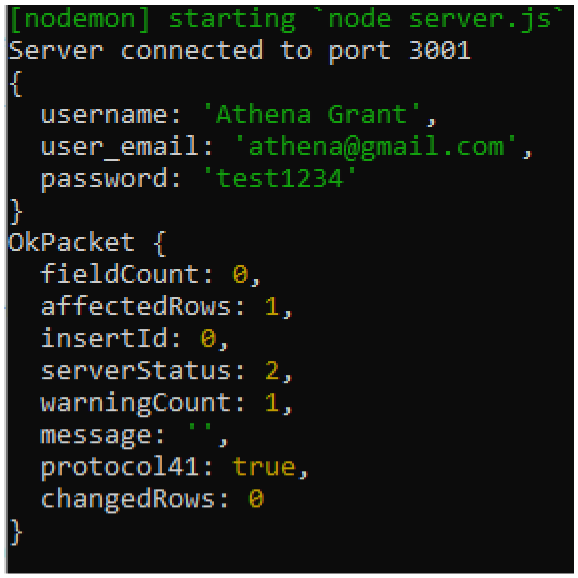
```
const Mutation = new GraphQLObjectType({
  name: "Mutations",
  fields: {
    Signup: {
      type: Result,
      args: {
        username: { type: GraphQLString },
        user_email: { type: GraphQLString },
        password: { type: GraphQLString },
      },
      resolve(parent, args) {
        return Signup(args)
          .then((result) => {
            return result;
          })
          .catch((err) => {
            return err;
          });
      },
    },
    Login: {
```



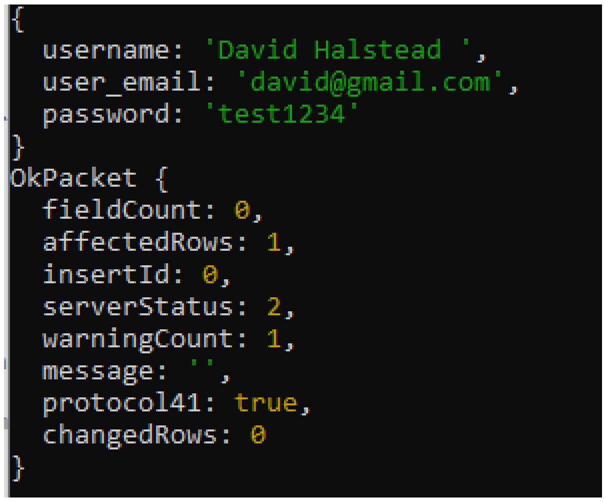
```
const RootQuery = new GraphQLObjectType({
  name: "RootQueryType",
  fields: {
    userDetails: {
      type: UserType,
      args: { user_email: { type: GraphQLString } },
      resolve(parent, args) {
        return userDetails(args.user_email)
          .then((result) => {
            console.log(result);
            return result;
          })
          .catch((err) => {
            return err;
          });
      },
    },
```

<u>Screenshots of console.logs</u>

After Signup of Athena Grant and David Halstead

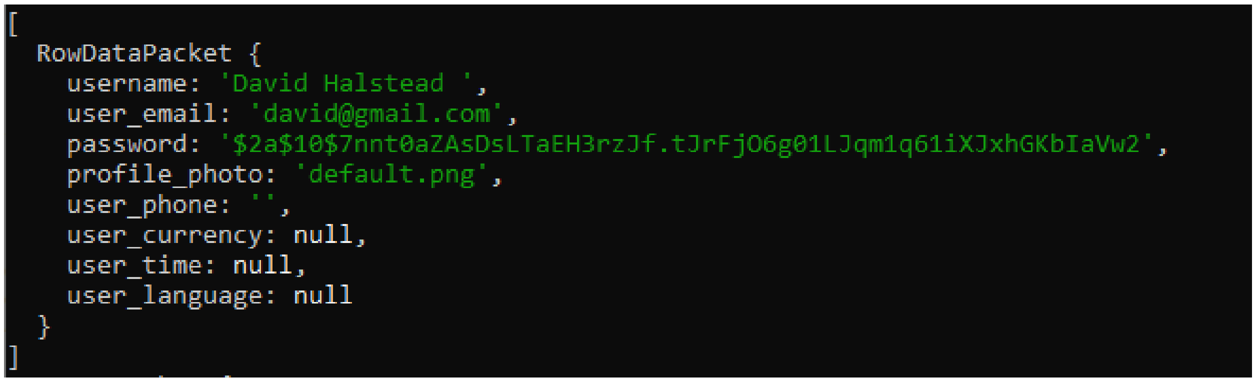


```
[nodemon] starting `node server.js`
Server connected to port 3001
{
  username: 'Athena Grant',
  user_email: 'athena@gmail.com',
  password: 'test1234'
}
OkPacket {
  fieldCount: 0,
  affectedRows: 1,
  insertId: 0,
  serverStatus: 2,
  warningCount: 1,
  message: '',
  protocol41: true,
  changedRows: 0
}
```



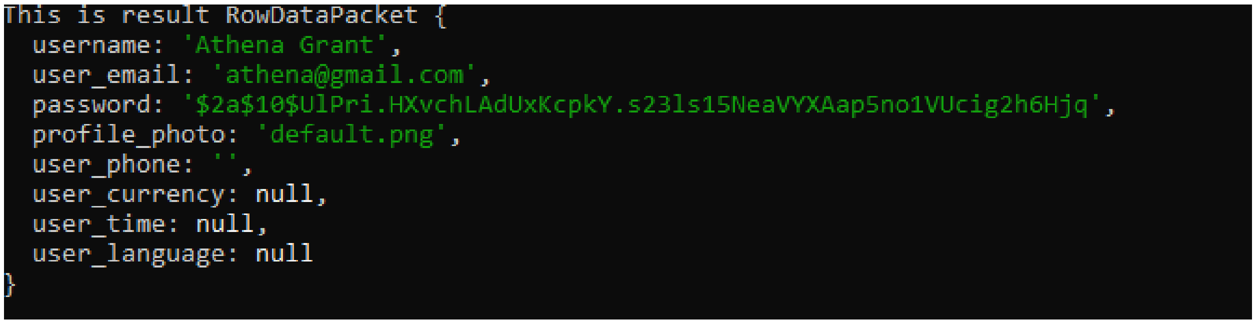
```
{
  username: 'David Halstead ',
  user_email: 'david@gmail.com',
  password: 'test1234'
}
OkPacket {
  fieldCount: 0,
  affectedRows: 1,
  insertId: 0,
  serverStatus: 2,
  warningCount: 1,
  message: '',
  protocol41: true,
  changedRows: 0
}
```

Getting details of David Halstead



```
[
  RowDataPacket {
    username: 'David Halstead ',
    user_email: 'david@gmail.com',
    password: '$2a$10$7nnt0aZAsDsLTaEH3rzJf.tJrFjO6g01LJqm1q61iXJxhGKbIaVw2',
    profile_photo: 'default.png',
    user_phone: '',
    user_currency: null,
    user_time: null,
    user_language: null
  }
]
```

Athena Grant after login console.log



```
This is result RowDataPacket {
  username: 'Athena Grant',
  user_email: 'athena@gmail.com',
  password: '$2a$10$UlPri.HXvchLAdUxKcpkY.s23ls15NeaVYXAap5no1VUcig2h6Hjq',
  profile_photo: 'default.png',
  user_phone: '',
  user_currency: null,
  user_time: null,
  user_language: null
}
```

Performance with GraphQL

Fetching data becomes much easier in GraphQL as multiple entities can be fetched in a single call and then used as and when required. Since the number of API calls are greatly reduced this improves speed over REST.

QUESTIONS:

1.How will you enable multi part data in GraphQL?

Using multi-part data in GraphQL is a pain point because GraphQL basically has only 4 Scalar Type support and those are String, Int, Float and Boolean. A plausible strategy to upload another data type like maybe an image would be: Making a different route which handles the file uploading and returns the path to the GraphQL Mutation to do necessary database transaction. Another strategy would be using some opensource library to enable multipart data.

2. Discuss the architecture for using multi part data in GraphQL without using any opens source library from git.

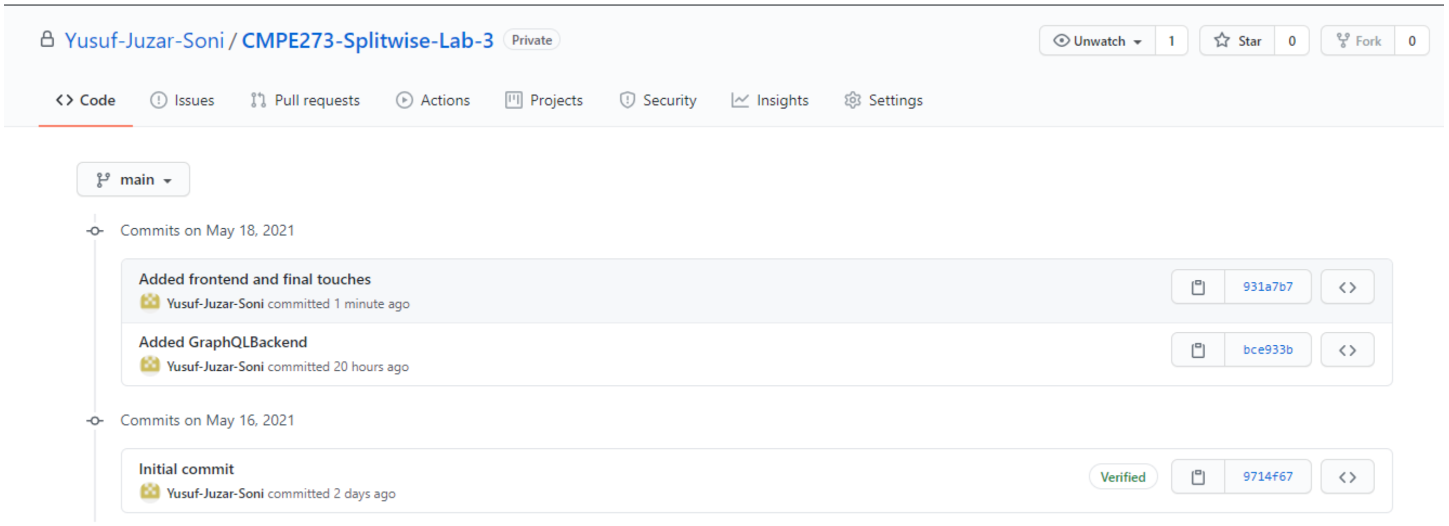
Taking the example of uploading images in the previous question.
I can think of two primary strategies to upload files through GraphQL
First Strategy would be encoding the image in Base64 and pass it as a string in mutation.
Second strategy would be: Making a different route which handles the file uploading and returns the path to the GraphQL Mutation to do necessary database transaction. Drawback for first strategy would be larger files would need to be handled as encoding increases file size. Second strategy has a drawback that GraphQL will have to wait for the file upload, impacting asynchronicity.

3. State any open source library for enabling multi part data transfer using GraphQL with sample code. Argue why do you think this library is a good fit?

Apollo-upload-server one of the libraries for file uploads available for GraphQL. It is a good fit as there are very few libraries currently offering this functionality. It also comes with a host of other features that can be used to work on multi part data.

```
import express from 'express'
import graphqlHTTP from 'express-graphql'

import { apolloUploadExpress } from 'apollo-upload-server'

import schema from './schema'

express()

  .use(

    '/graphql',

    apolloUploadExpress({ maxFileSize: 10000000, maxFiles: 10 }),

    graphqlHTTP({ schema })

  ).listen(3000)
```

GIT COMMIT HISTORY



Yusuf-Juzar-Soni / CMPE273-Splitwise-Lab-3 Private

main

Commits on May 18, 2021

Added frontend and final touches
Yusuf-Juzar-Soni committed 1 minute ago — 931a7b7

Added GraphQLBackend
Yusuf-Juzar-Soni committed 20 hours ago — bce933b

Commits on May 16, 2021

Initial commit
Yusuf-Juzar-Soni committed 2 days ago — Verified — 9714f67

======================XXXXXXXXXXXXXXXXXXXXXXX-============================