**MeetUP**

Spring-2020

Team: CEDAH

Nghi Tran

Ethan Minter

Harry Park

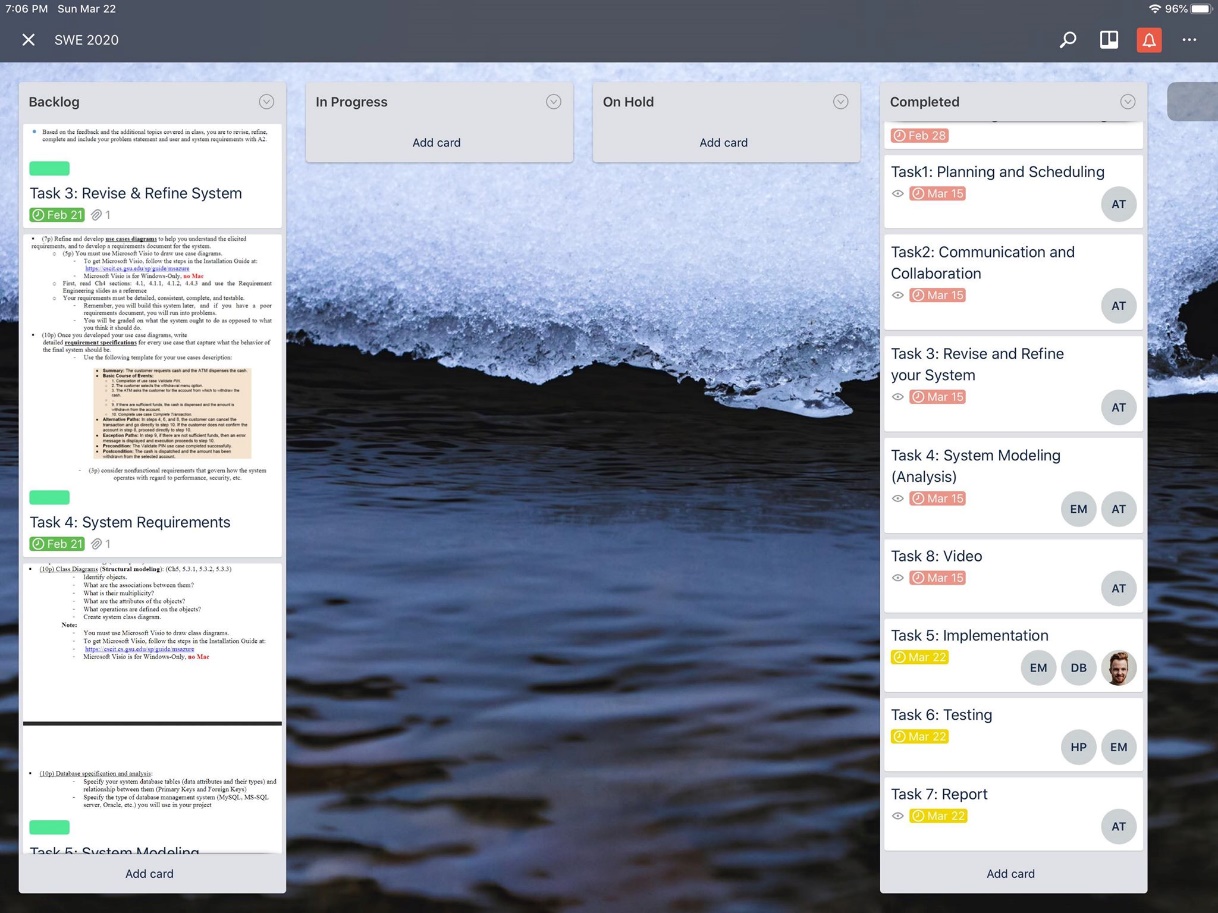
Daniel Busby

Connor Lacy

Planning and Scheduling

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Assignee  Name | Email | Task | Duration  (Hours) | Prereq. | Due date |
| Nghi Tran  (coordinator) | ntran52@student.gsu.edu | 1,2,3,  4,7,8 | 5 hours | None | 3/22/2020 |
| Ethan Minter | eminter2@student.gsu.edu | 4,5,6 | 5 hours | none | 3/22/2020 |
| Daniel Busby | dbusby1@student.gsu.edu | 5 | 3 hours | none | 3/20/2020 |
| Connor Lacy | clacy2@student.gsu.edu | 5 | 5 hours | Task 5 part 1 | 3/22/2020 |
| Harry Park | [hpark44@student.gsu.edu](mailto:hpark44@student.gsu.edu) | 6 | 4 hours | Task 5 | 3/22/2020 |

Everyone did their job as assigned.



Teamwork Basics

o Ground Rules: Norms 1 to norms 5

Work Norms: All of the assignments are distributed upon the interest and the availability of time. The group coordinator will set the deadlines for everyone. When someone doesn’t follow through on their commitment or people have different opinions about the quality of work, all group members will collaborate to fix the problem. Different work habits are not a big problem unless people can get the assignments done in time. We agreed upon not procrastinating this project as a last-minute assignment.

Facilitator Norms: Our group will have a facilitator. The group coordinator will be the facilitator. As the group coordinator is going to be rotated for each assignment, every member will have a chance to learn new things as a facilitator. The responsibility of the facilitator is (1) to encourage the members to give their hundred percent of the work to the assignment, (2) help them to figure out an alternative if any problems occur, and (3) make sure they finished the tasks on time.

Communication Norms: The communications are being done via Discord app so everyone could see the messages and reply instantly.

Communication Norms: The communications are being done via Discord app, so everyone could

check the messages and reply instantly.

Meeting Norms: All the group members have different availabilities with work schedules and other

classes. The group coordinator is responsible for deciding the meeting time, but every person should agree on the decision. Everyone has his preference about the meeting time and place. The best place for the meeting would be the library rooms. If any members are late or absent to a meeting, we are going to post the things discussed during the meeting on the Discord app so they would not be left behind during the assignment. If they miss several meetings, we would have to talk with the instructor to resolve the issue.

Consideration Norms: Most of the time, the meeting will take place at library rooms. So, people

might bring food based on library policy. If one person dominates the discussion, others need to make sure that he listens to others’ ideas unless he is making decisions as a coordinator. If someone is not comfortable with the norms, they can be changed by discussing with the team members and coming up with an agreement.

o Hints for Handling Difficult Behavior

If a person is overly talkative, others could try to discourage them from dominating the discussion subtly. It should be done gently so the person might not lose their enthusiasm for the project. If the person insists on their behavior, the group coordinator should have a private conversation with them to encourage the person to hear others’ opinions.

If a person is having difficulties communicating with others, others could ask them about their opinion, offer any help they might need, and encourage him to help others. Asking questions to break the ice and get familiar will make the person feel comfortable and be able to adjust to the group environment. Another way is to start talking with the person in Discord, as he may prefer online discussion over the face-to-face conversation.

One of the group members has experienced this during the Web Programming class project. When they first formed a group with another classmate, they seemed uncomfortable with discussing the project in person and remained quiet and non-assertive. For introverts, it is natural for them to take their time before saying something instead of thinking out loud. They might seem reserved because of this behavior, but the words they share can have much more thoughts behind them. So, their strength could be stood out within an online discussion. While having a conversation online, they were able to see that this person has a strong knowledge and initiative to lead the project.

If a person would like to give feedback to others, it can positively affect the performance and productivity of a group. However, everyone needs to be careful when criticizing others as it can hurt not only the person being criticized by making him discouraged but also the whole group by dragging out the discussion.

If a person keeps complaining, others should first try to listen to them to see if their complaints are legitimate. If so, the whole group should start a discussion to resolve the issue. Otherwise, other members should remind them part of the work is learning how to solve problems and motivate them to see the bigger picture, setting aside the small conflicts and disagreements.

o Hints for Handling Group Problems

When the groups have just been formed and members are getting to know each other, the productivity might not be in its full potential. In this phase, setting up a common goal and creating a list of tasks to accomplish the goal will help to unify the team.

If the conversation is not focused on the work at hand, the progress of a group can be dragged out. When the discussion goes off at a tangent, at least one of the members should be ready to remind the group to go back to the main topic.

It would not be a great idea to make a decision quickly and put pressure on others to follow. Understanding is a critical factor in making a decision. Everyone should propose their ideas in a discussion, and the group should decide as a whole. We should try to cooperate with each other, setting a common goal that everyone can agree on. If the group is not able to reach a consensus for a particular issue, it will have to decide by majority rule unless there is an alternative that can make everyone satisfied.

A conflict between the members might slow down the progress. As it is difficult to move forward without resolving the conflict, they will have to discuss the problem first. They should try to listen to what the other person has to say and think of the problem in other’s side.

Every person in the group matters. It is not acceptable to ignore or ridicule another member during the project. In the workplace, it is necessary to know how to deal with people that we are not comfortable with.

There is always a group member who is not willing to cooperate, does not complete their tasks in time, or is absent to the scheduled meetings. The group coordinator should be ready to talk to the uncooperative person, explaining how their behavior negatively affects the whole group. If the person is not willing to change, it would be necessary to talk with the instructor to address the problem.

One of the group members has experienced this during the Mobile App Development class project. Their classmate was responsible for implementing a couple of fragments for the Android application. However, they kept creating unreasonable excuses and had no contribution to the project. As their behavior has not changed after several reminders, other members had to speak to the instructor about the problem. They ended up having a significant penalty for the project grade. It would be ideal if we could convince the person to be more cooperative. However, it is sometimes necessary to blow the whistle on selfish, stubborn behavior.

Problem Statement

Our product is a mobile application that helps users schedule meetings. It is designed for teams and other small groups in both business and casual settings. It streamlines the scheduling process by handling the recognition of overlapping available times, and eases the decision making process. The original alternative to our product was emailing everyone involved and trying to figure out availability through back and forth communication. This process is inefficient because each person generally only knows their schedule. Our app will present available meeting times without needing discussion and debating.

The objective of our project is to allow users to make a group in the app, and then send notifications to group members stating their desire to schedule a meeting within the next week. Then, users indicate their availability at various times on each day within the period. Users can designate a time as unavailable, available, and partial availability (this can include an undesired time or a virtual presence in an in-person meeting). After the app determines the overlapping times, ranks them by percentage of participants that are fully available, then by partial availability. Users then vote for a time. We will keep the scope to scheduling a meeting within a one week period because setting availability for more than 7 days would be tedious, and schedules are often much more variable outside of that time.

Our main competitor is Doodle. Doodle’s approach is to allow hosts of events to send times to group members and have them take a poll on which they prefer. Our approach will be to have the app find availabilities for the user, which will provide options that a host could possibly miss. In addition, we will have our app adapt to emergency schedule changes automatically. When a user changes availability on an already voted on event, a message will be sent to the group leader asking if the group should vote again. Then, the app repopulates the list with the overlapping times, and the group decides another time. Another feature that will differentiate our product will be an agenda feature that implements image recognition technology to translate handwritten notes into digital notes for the meeting.

Requirements

The requirements were elicited through two main methods. Firstly, we faced a personal need for a mobile scheduling application and thought about what features would be most beneficial for us.

Secondly, we looked at an already existing scheduling application, Doodle, and figured out what it’s functionalities were and what features our product would also need.

User Functional Requirements

1. The group leader must be able to invite other users to a meeting.

2. Users must be able to set availabilities.

3. Users must be able to designate partial availabilities, and give a reason why.

4. The users must be able to vote on overlapping times.

5. Users then vote on the times in order to determine the best time for everyone.

6. The app should repopulate the list of times if a schedule changes.

7. The app should be able to convert handwritten notes to text notes for the meeting.

System Requirements

Functional Requirements:

1. Creation of user accounts (from user via web page)

2. Creation of a group of users. (from use via web page)

3. Creation of a new meeting including a list of attendees within a group. (from user in group, via web page)

4. Accept schedules of users (from user via web page)

5. Calculates meeting times from input schedules (at time of meeting creation)

6. Create a vote for an appointment time and date. (at time of meeting creation)

7. Meeting invitations accept attachments (at time of meeting creation)

8. Convert handwriting into text (on request, for meeting attachments)

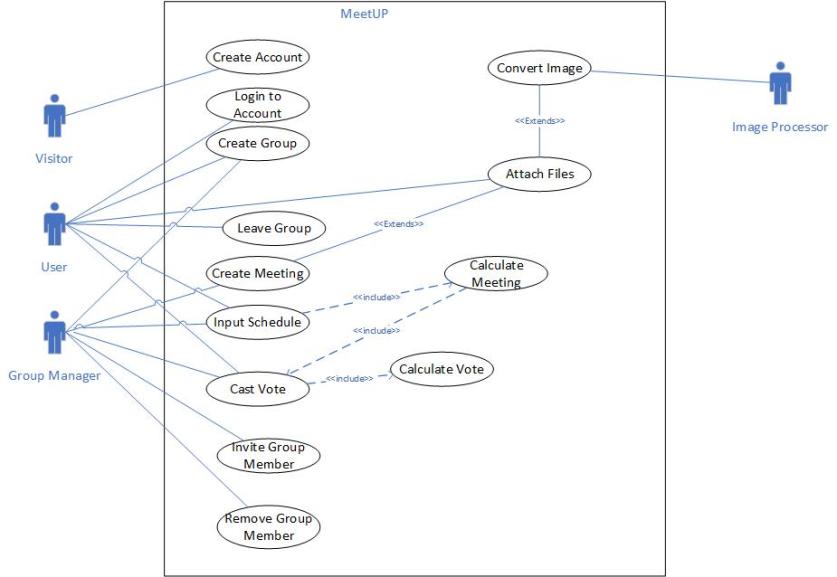
Non-Functional Requirements:

1. User credentials must be stored securely using hash

2. Each group must have a Manager and hierarchy of inheritance decided by manager.

3. Schedules may need to be confidential depending on type of organization

Use Case Diagram:



Use Case Specifications:

**Register Account**

**Summary:** The user creates an account on MeetUP.

**Basic Course of Events**

1. The user selects the create account option in the menu

2. The user is asked to input a username and password.

3. The system searches the database to confirm that the username and password are unique.

4. If the username is unique, the username is stored in the database.

5. If the password is unique, the password is hashed and then stored in the database.

6. A user id is created for the user.

7. The system outputs a message stating that registration was successful.

8. The user is taken to the login screen.

**Alternate Paths** : Users can at any time cancel the registration process.

**Exception Paths** : If the username or password is not unique, the user is asked to input a new

username and password.

**Preconditions** : None

**Postconditions:** An account has been created by the user.

**Login to Account**

**Summary:** The user inputs their username and password and logs in.

**Basic Course of Events**

1. The user selects the log in option in the menu.

2. The user is asked for their username and password.

3. The system checks to see if the username exists.

4. The System checks to see if inputted password matches the password associated with the username.

5. If the username exists and the password is correct, the user is logged in.

6. The user is then taken to the main menu.

**Alternate Paths:** If the user has just registered, then they will enter the login screen as soon as they do so.

**Exception Paths:** If the username or password are invalid, then the user is asked to input their login again.

**Preconditions:** None

**Postconditions:** The user has logged into their account.

**Create Group**

**Summary:** The user creates a group and sends invitations to other group members.

**Basic Course of Events**

1. The user selects the create group option.

2. The user is asked to input a name for their group.

3. The user invites other users by inputting their usernames.

4. The invited users receive the invitation, and either accept or reject the invitation.

5. Once a user accepts the invitation, they are added to the group.

6. The group creator becomes the Group Manager for the group.

**Exception Paths:** If the user inputs an invalid username in step 3, they are asked to input another

username.

**Precondition:** The user is logged in.

**Postcondition:** The group is created and members are invited into the group.

**Leave Group**

**Summary:** A user leaves a joined group.

**Basic Course of Events**

1. The user selects a joined group.

2. The user selects the leave group option.

3. The user id of the user is no longer associated with the group.

**Alternate Paths:** The user can cancel leaving the group.

**Precondition:** The user must be part of a group.

**Postcondition:** The user is no longer a member of the left group.

**Create Meeting**

**Summary:** The Group Manager creates a meeting to take place within the next week.

**Basic Course of Events**

1. The group manager selects the create meeting option.

2. The group manager can set the meeting to repeat next week.

3. The group manager names the meeting and sets the length.

4. Each user receives a notification of the meetings creation.

5. Complete use case input schedule for each user.

6. Complete use case calculate meeting.

7. Complete use case cast vote.

8. Complete use case calculate vote.

9. The meeting time is sent to each user.

10. The group manager can attach files to the meeting, using the Attach Files use case.

**Alternate Paths:** If the meeting is set to repeat, at the beginning of the next week the process begins

at step 3.

**Precondition:** The user must be the Group Manager of a group.

**Postcondition** : A meeting is scheduled, and all group members are notified on the date.

**Input Schedule**

**Summary:** The user inputs their availability for the given week.

**Basic Course of Events**

1. The user can indicate that their availability is the same as last week, or they can choose to make a new schedule.

2. If the user has a new schedule, they are asked what days that week they are available.

3. The user then indicates available times by selecting sections of time during that day.

4. The user can select from a list of types of partial availability and indicate sections of time where that applies.

5. Step 3 and 4 are repeated for each available day.

6. The user selects the finish schedule option, and the availabilities are saved.

**Alternate Paths:** If the user does not want to change their schedule from a previous week, the

process ends at step 1. If they wish to modify an existing schedule, the user is instead asked to

indicate what day to change availability for in step 2.

**Precondition:** User must be part of a meeting.

**Postcondition:** The user’s availability is set and ready to meeting time calculation.

**Calculate Meeting**

**Summary:** The system takes the schedules of the users within a group and finds the overlapping

times.

**Basic Course of Events**

1. The system analyzes the schedules of the users and finds overlapping complete availability from all users.

2. The system then analyzes the schedules to find overlaps that include partial availability from some users.

3. The system, if overlaps were located, lists the overlaps by percentage of complete availability, then percentage of partial availability.

4. The system then presents users with the list of overlaps to be voted on.

**Alternate Paths:** If no overlaps can be found, then the group manager and all group members are

alerted.

**Precondition:** A meeting must have been created and all members of the meeting must have a

created a schedule.

**Postcondition:** The group now has a list of meeting times to be voted on.

**Calculate Vote**

**Summary:** The system selects the overlap that has the most votes, and reports the winner to the

group.

**Basic Course of Events**

1. Once all users in a group have cast their votes on an overlap, then the system counts the number of votes each option received.

2. The system finds the overlap that received the most votes.

3. The system sends a message to all group members indicating the winning time slot.

**Precondition:** All users must have voted for a time slot.

**Postcondition:** The meeting time has been selected.

**Invite Group Member**

**Summary:** The group manager invites a new user into the group.

**Basic Course of Events:**

1. The group manager selects the add member option.

2. The group manager inputs the username of the invitee.

3. The invited user can choose to either accept or reject the invitation.

4. Upon accepting the invitation, the user is now a member of the group.

**Alternate Path:** The invited user rejects the invitation.

**Exception Path:** If the username does not exist, then the group manager receives an error that states

that the user does not exist.

**Precondition:** The invitee must not already be a member of the group.

**Postcondition:** The invitee is now a member of the group.

**Cast Vote**

**Summary:** Users and Group manager can vote on the suggested meeting times

**Basic Course of Events:**

1. A group meeting is proposed by a group manager

2. Scheduling engine builds suggestions for meeting times

3. Notification is distributed to all members of group

4. Group members cast their vote on the suggested meeting times

5. Votes are tallied and used for meeting creation

**Alternate Paths:**

- User or Group Manager does not cast vote

**Exception paths:**

- User contact information out of date or unavailable

**Precondition:**

- Users and group manager are in a group together

- Users have input their schedule for the week

**Remove Group Member**

**Summary:** A group member is removed from the group by the Group Manager

**Basic Course of Events:**

1. A group manager enters a group they manage

2. The group manager removes a group member from the group.

3. The group member removed will no longer receive updates or notifications about future meetings

**Alternate Paths:**

- A group member may leave the group on their own accord

**Exception Paths:**

- A group member who is not the group manager attempts to remove another individual

**Precondition:**

- The member of the group who wishes to remove another member on their behalf is a group

manager

- Group manager can only remove individuals from groups they manage

**Attach Files**

**Summary:** A group member can attach files to meeting

**Basic Course of Events:**

1. A group member attaches a video, image, or document to the meeting.

2. If it is an image, the user can indicate that there is handwriting to be converted to digital text.

3. If the user wants the image converted, complete use case Convert Image.

**Alternate Paths:**

- No group members upload attachments to the meeting

**Exception Paths:**

- Attachment too large to upload

- Broken link or incompatible attachment type

**Precondition:**

- A meeting exists with members

- Meeting able to accept attachments

**Convert Image**

**Summary:** Image processor converts image to digital text; attached to meeting description

**Basic Course of Events:**

1. An image frame

2. The image is processed with NLP techniques

3. Handwritten text is converted to digital language

4. Digital conversion is parsed into meeting body

**Alternate Paths:**

- The camera capture can be cancelled

- Image read from device local storage

**Exception paths:**

- Camera unable to be detected

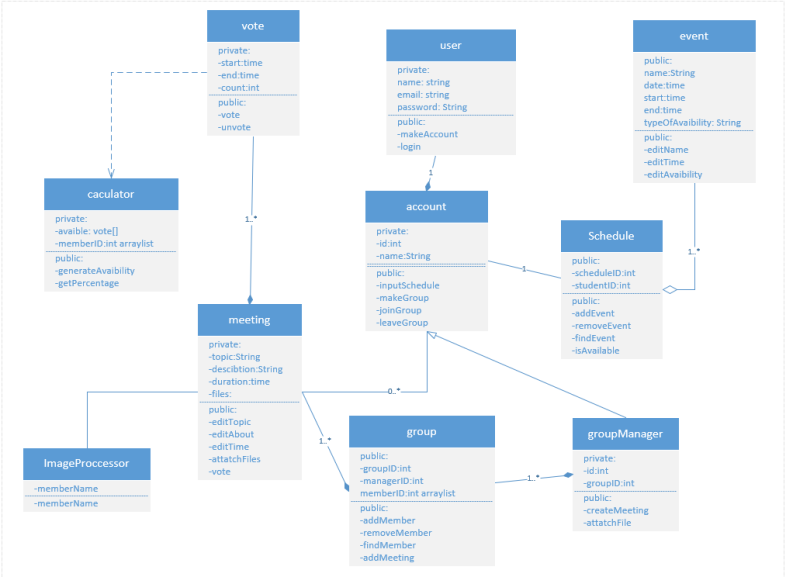
- Image processing unable to read handwritten language

**Precondition:**

- An image is able to be read from a camera or device storage

System Modeling

Class Diagram:



Database Specifications:

Since the project handles structured data, relational database would be ideal for storing and retrieving information with queries. Relational database also allows tiered access in applications. For example, users could view their accounts while managers could both view and make necessary changes. Among a few different SQL-based database management systems, MySQL will be used for the project.

1. Logins

|  |  |  |
| --- | --- | --- |
| ID | Username | Password |
| int | char (50) | char (50) |
| Primary Key |  |  |

1. Users

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ID | Name | E-mail | Phone | Group\_ID |
| int | char (50) | char (50) | char (50) | int |
| Primary Key |  |  |  | Foreign Key |

1. Groups

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ID | Meeting\_Day | Meeting\_Start\_Time | Meeting\_End\_Time | Manager\_User\_ID |
| int | char (50) | time | time | char (50) |
| Primary Key |  |  |  | Foreign Key |

1. User Availabilities

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ID | Day | Start\_Time | End\_Time | User\_ID |
| int | char (50) | time | time | int |
| Primary Key |  |  |  | Foreign Key |

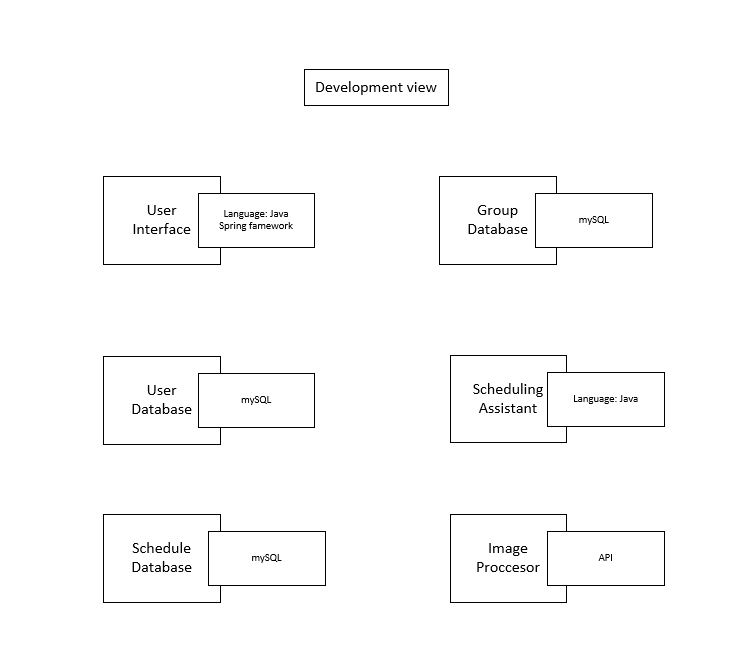
1. Meeting Possibilities

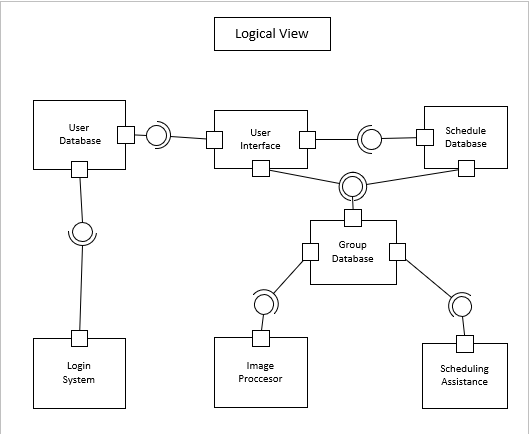
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ID | Day | Start\_Time | End\_Time | Vote\_Count | Group\_ID |
| int | char (50) | time | time | int | int |
| Primary Key |  |  |  |  | Foreign Key |

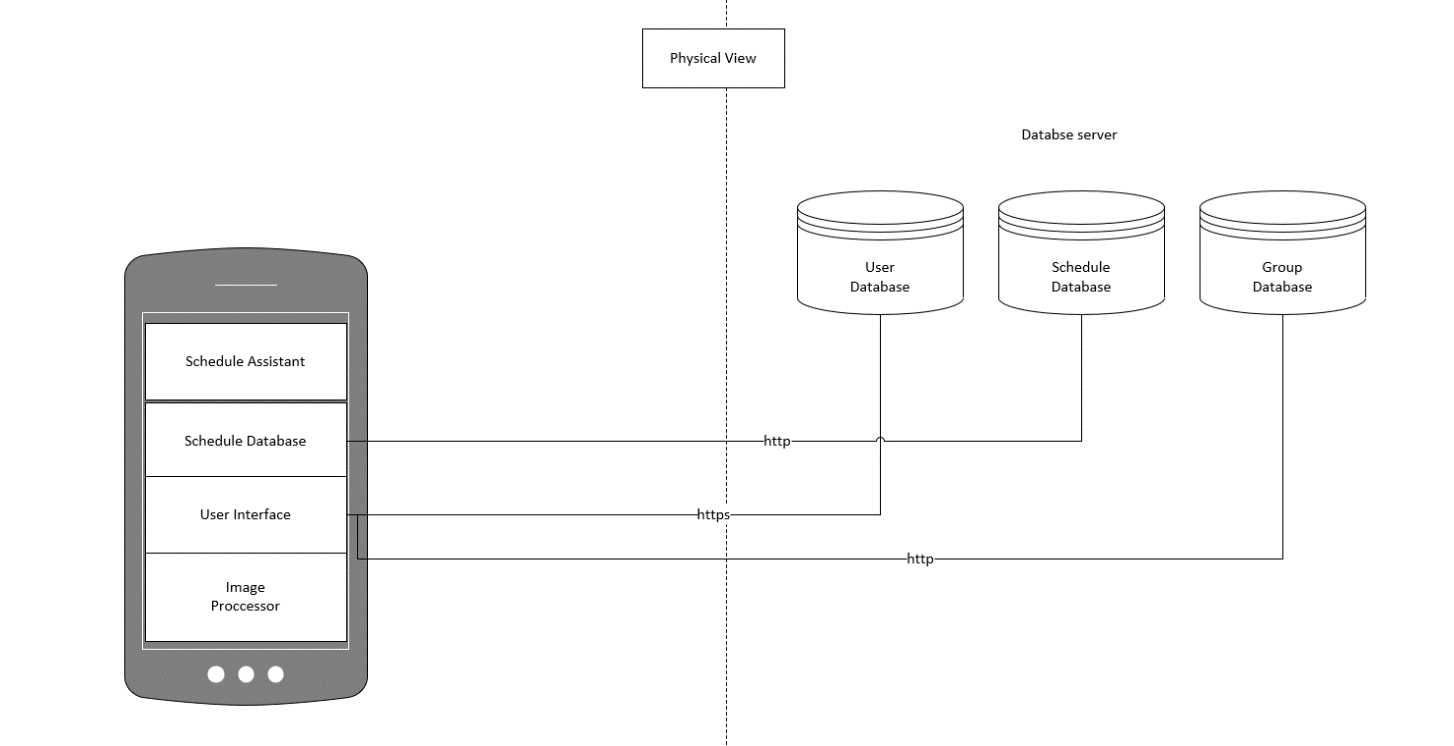
1. Files

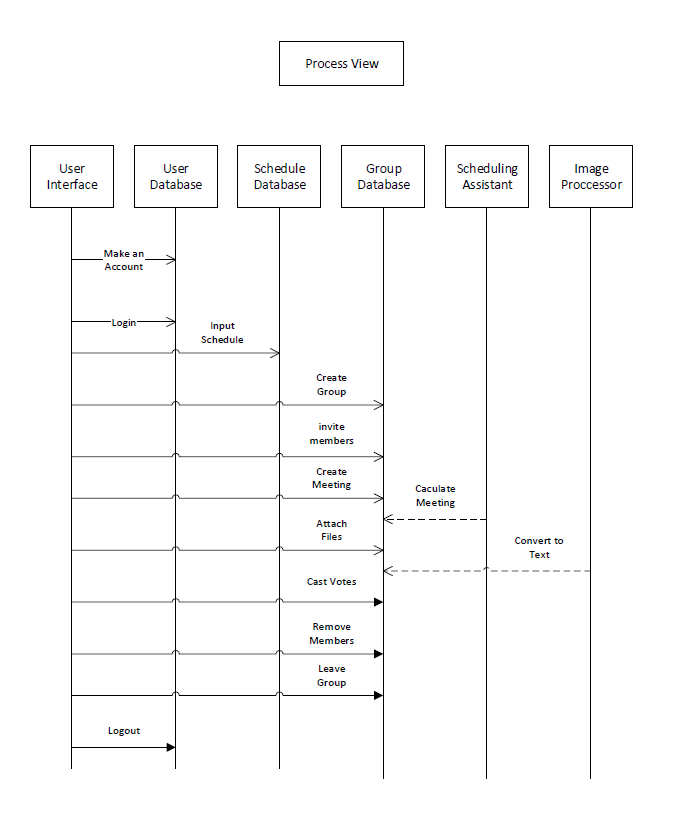
|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| ID | Description | Data | Filename | Filesize | Filetype | Group\_ID |
| int | char (50) | longblob | char (50) | char (50) | char (50) | char (50) |
| Primary Key |  |  |  |  |  | Foreign Key |

4+1 Diagram

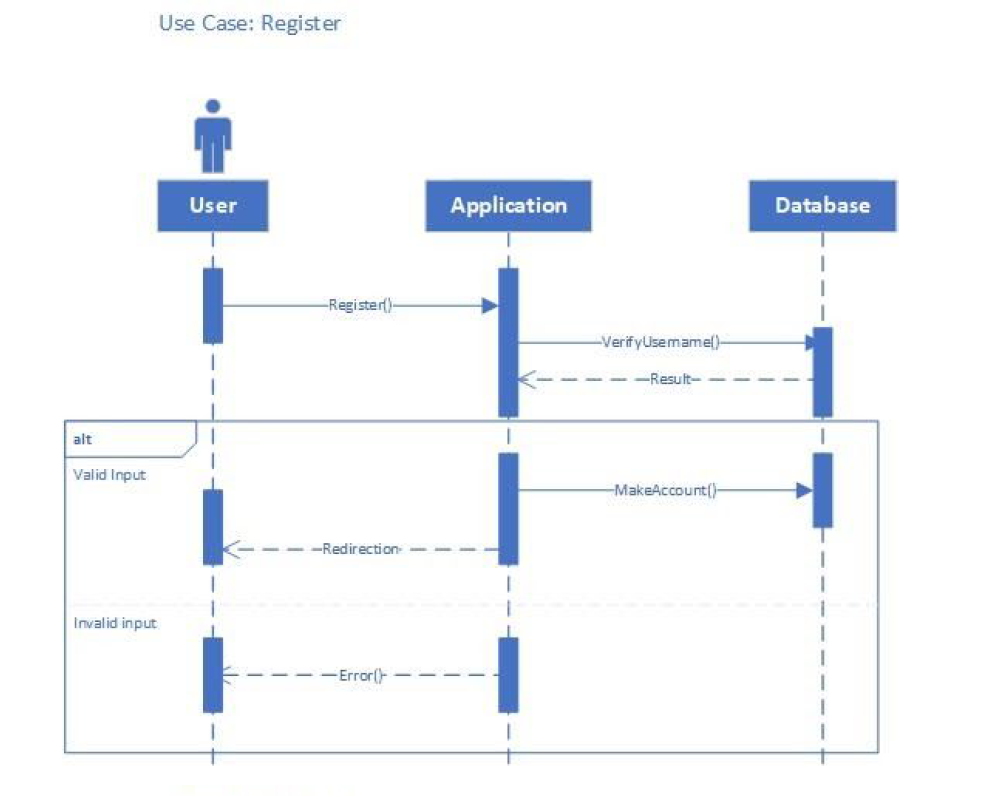


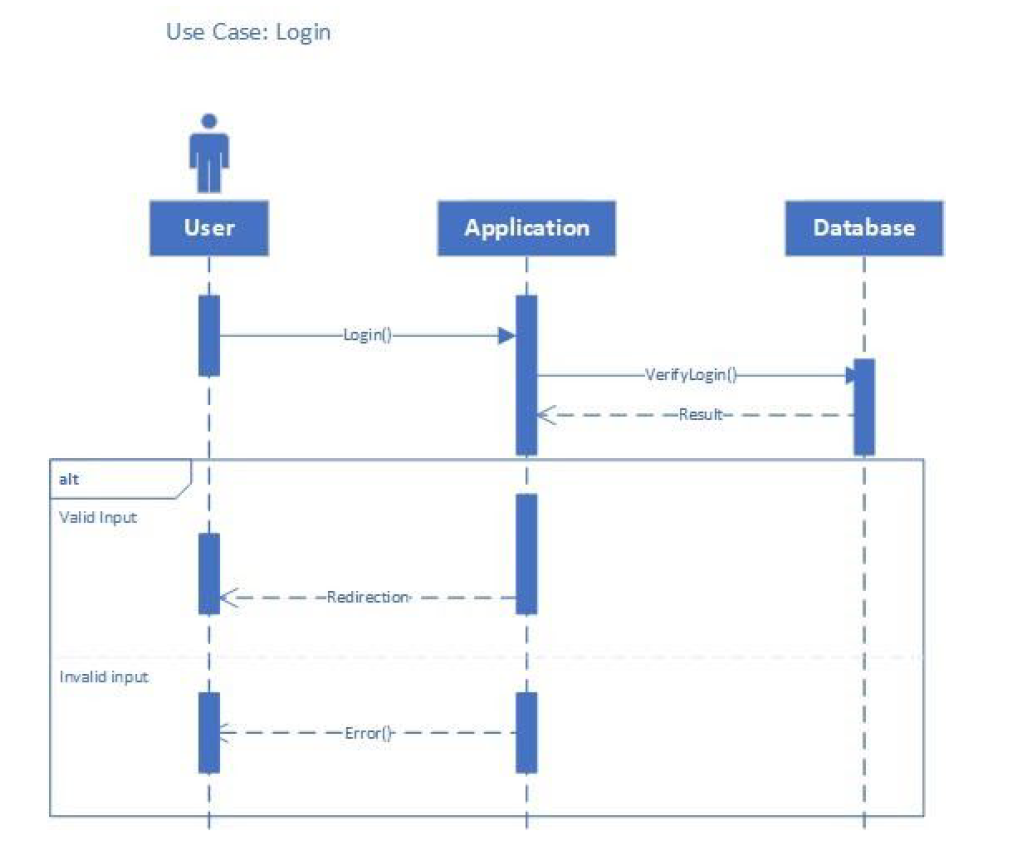




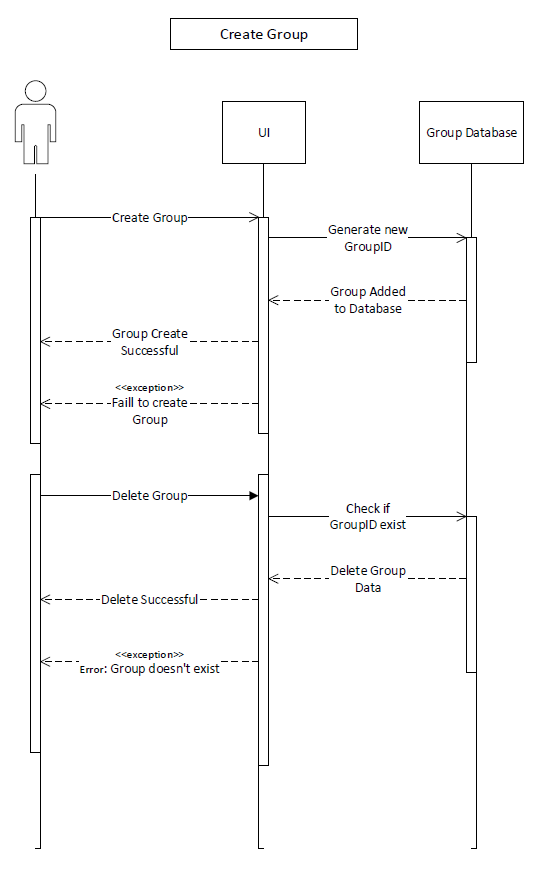


Sequence Diagrams

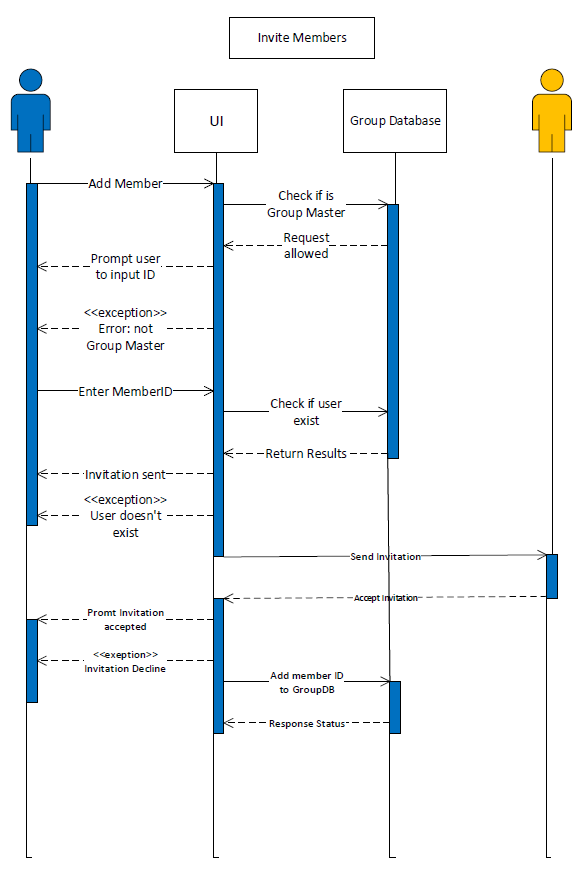




Use Case: Create Group

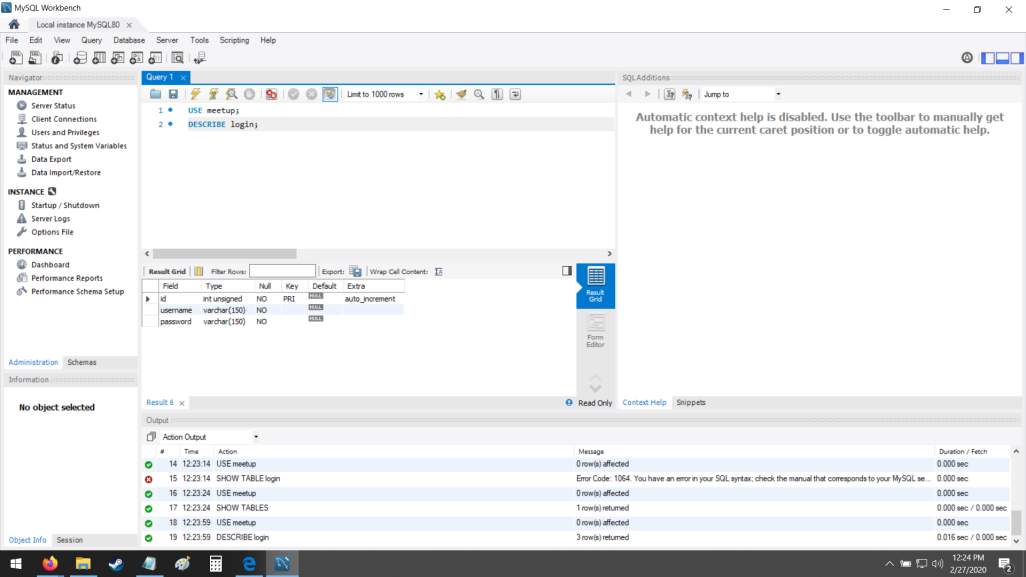


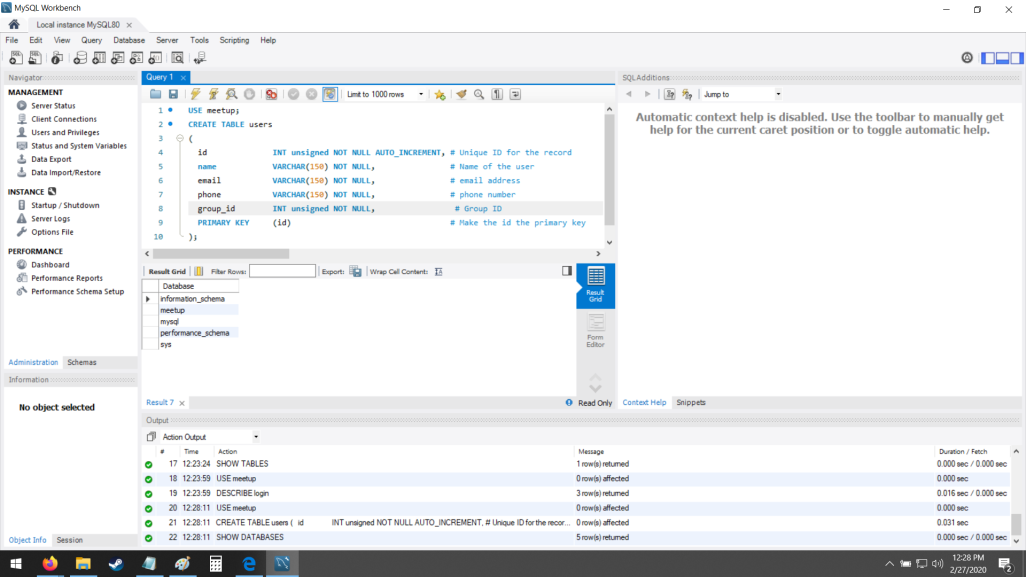
Use Case: Add Member

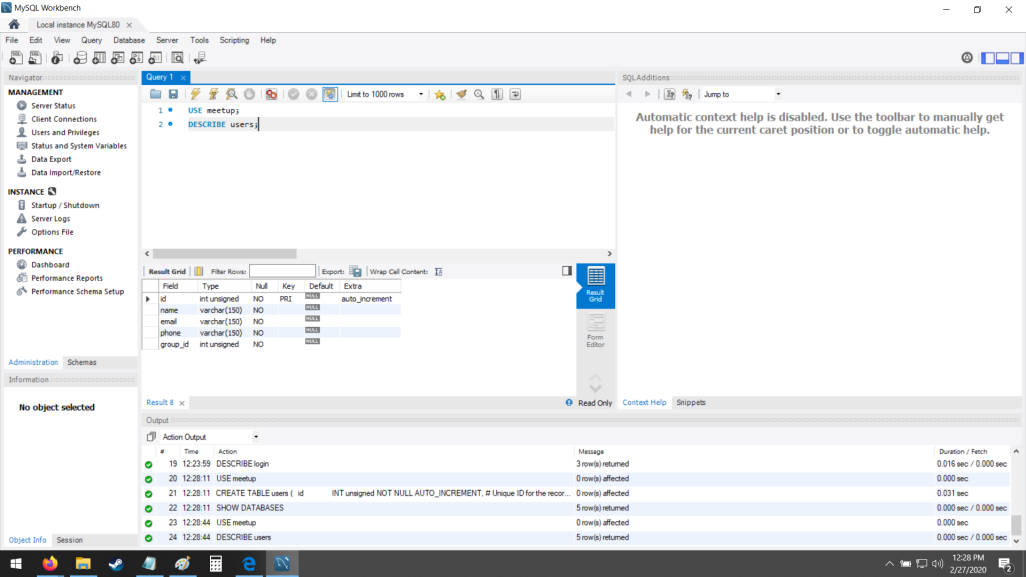


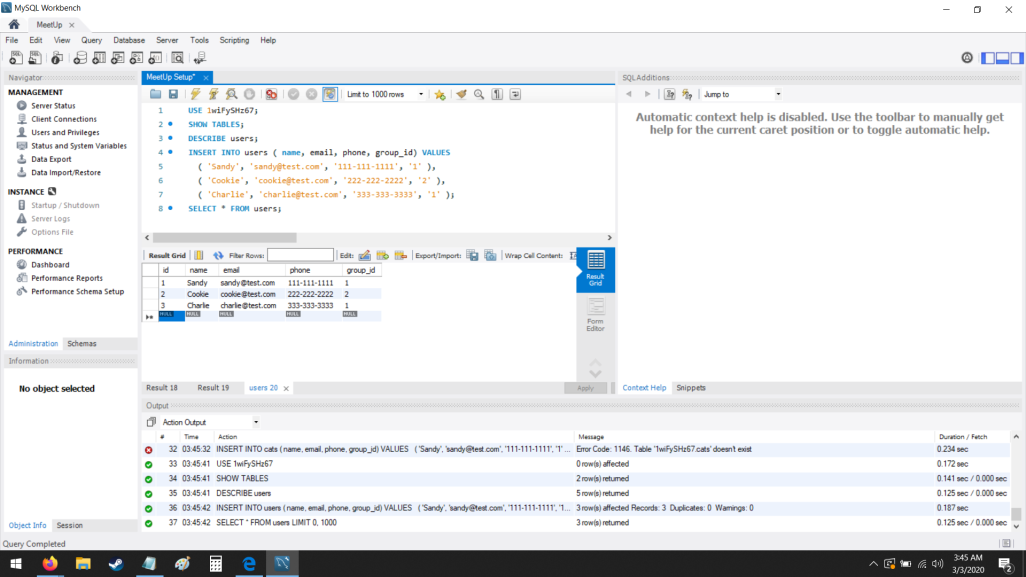
Implementation

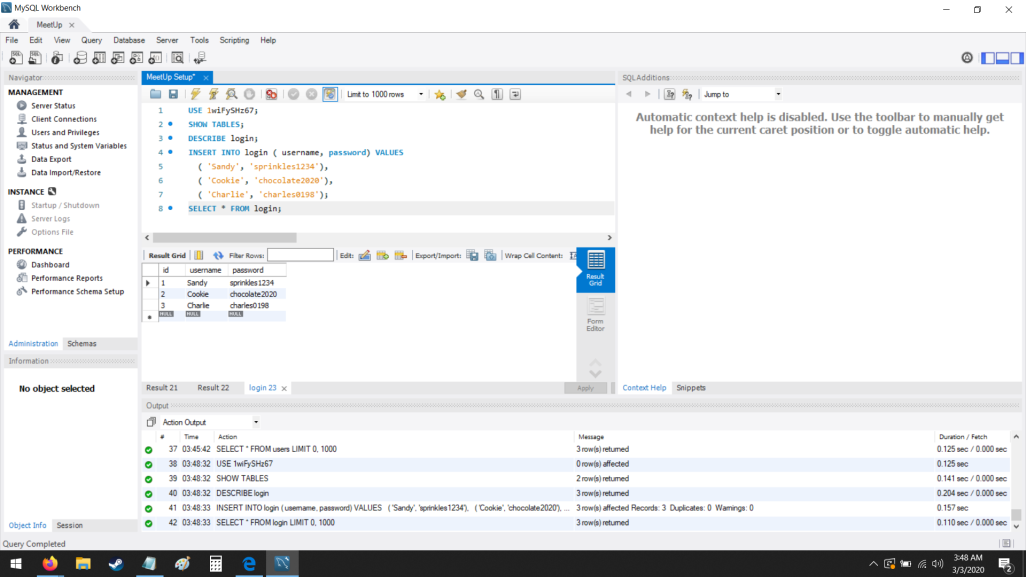
Database:











The database is used to store usernames, emails, phone numbers and passwords for use in the login and register use cases. In order to test the database, we used a MySQL workbench server. This will not be the final hosting service, but it was useful for initial implementation. Since we did not received feed back from professor Mussa on the Database set, we kept it at it was before and work on top of that.

|  |
| --- |
| Project A4 queries: |
| use CEDAH\_MEETUP; |
|  |  |
|  | CREATE TABLE IF NOT EXISTS logins ( |
|  | id int auto\_increment PRIMARY KEY, |
|  | username varchar(255), |
|  | password varchar(255) |
|  | ); |
|  |  |
|  | CREATE TABLE IF NOT EXISTS users ( |
|  | id int auto\_increment PRIMARY KEY, |
|  | name varchar(255), |
|  | email varchar(255), |
|  | phone varchar(255), |
|  | group\_id int |
|  | ); |
|  |  |
|  | CREATE TABLE IF NOT EXISTS meeting ( |
|  | id int auto\_increment PRIMARY KEY, |
|  | meeting\_day varchar(255), |
|  | meeting\_start\_time time, |
|  | meeting\_end\_time time, |
|  | manager\_user\_id int |
|  | ); |
|  |  |
|  | CREATE TABLE IF NOT EXISTS user\_availability ( |
|  | id int auto\_increment PRIMARY KEY, |
|  | day varchar(255), |
|  | start\_time time, |
|  | end\_time time, |
|  | user\_id int |
|  | ); |
|  |  |
|  | CREATE TABLE IF NOT EXISTS meeting\_possibilities ( |
|  | id int auto\_increment PRIMARY KEY, |
|  | day varchar(255), |
|  | start\_time time, |
|  | end\_time time, |
|  | vote\_count int, |
|  | group\_id int |
|  | ); |
|  |  |
|  | CREATE TABLE IF NOT EXISTS files ( |
|  | id int auto\_increment PRIMARY KEY, |
|  | description varchar(255), |
|  | data longblob, |
|  | file\_name varchar(255), |
|  | file\_size varchar(255), |
|  | file\_type varchar(255), |
|  | group\_id int |
|  | ); |
|  |  |

The database is used to store usernames, emails, phone numbers and passwords for use in the login and register use cases. It is also used to store meeting informations, user availabilities, meeting possibilities and files informations.

Frontend and Logic:

# How to run this project locally

## Meetup-API

#### Dependencies

* MySQL
* (Optional) MySQL Workbench
* Java

### Setup

* Download & Install MySQL on your machine
  + Mac:
  + $ brew install mysql
  + $ mysql.server start
  + $ mysqladmin -u root password ‘<YOUR\_PASSWORD>’

<YOUR\_PASSWORD> is the password you wish to set for the root user. Leave blank for no password

$ mysql -u root -p

$ Enter your password

$ CREATE USER ‘<YOUR\_USERNAME>’@‘localhost’ IDENTIFIED BY ‘<YOUR\_PASSWORD>’;

$ GRANT ALL PRIVILEGES ON \*.\* TO ‘<YOUR\_USERNAME>’@‘localhost’;

* + Windows: //TODO
  + Ubuntu: Installation Documentation [Here](https://www.digitalocean.com/community/tutorials/how-to-install-mysql-on-ubuntu-18-04)
  + Download: [Here](https://www.mysql.com/downloads/)
* Create a local Database
* $ mysql -u <YOUR\_USERNAME> -p
* $ <YOUR\_PASSWORD>
* $ CREATE DATABASE <YOUR\_DATABASE>
* Add SpringBoot **application.properties**
  + create folder 'resources' in /meetup-api/src/main/
  + create file 'application.properties'
  + Enter the following:
  + server.port = 8080
  + spring.jpa.hibernate.ddl-auto=create
  + spring.datasource.url=jdbc:mysql://localhost:3306/<YOUR\_DB>?useJDBCCompliantTimezoneShift=true&serverTimezone=UTC
  + spring.datasource.username=<YOUR\_USERNAME>

spring.datasource.password=<YOUR\_PASSWORD>

* + Verify server = 8080
  + Enter your Database Name you created into this url string
    - spring.datasource.url=jdbc:mysql://localhost:3306/<YOUR\_DATABASE>?useJDBCCompliantTimezoneShift=true&serverTimezone=UTC
  + Set your username you created during 'CREATE USER...' mysql script
    - spring.datasource.username=<YOUR\_USERNAME>
  + Set your password you created during 'IDENTIFIED BY...' mysql script
    - spring.datasource.password=<YOUR\_PASSWORD>
* Start API
  + Navigate to /CSC-SWE-CEDAH/
  + Start API
    - CSC-SWE-CEDAH connor$ ./mvnw spring-boot:run

## App (React-Frontend)

#### Dependencies

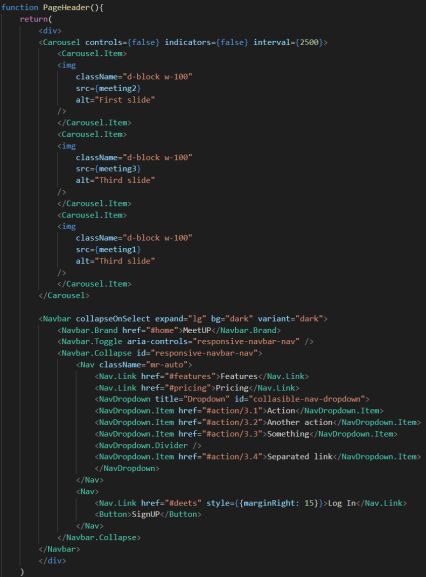
* yarn
* Install yarn or npm
  + If you chose npm, you must convert the yarn statements to npm
* Navigate to: /CSC-SWE-CEDAH/app/
* Install yarn dependencies:

yarn install

* Run Front end:

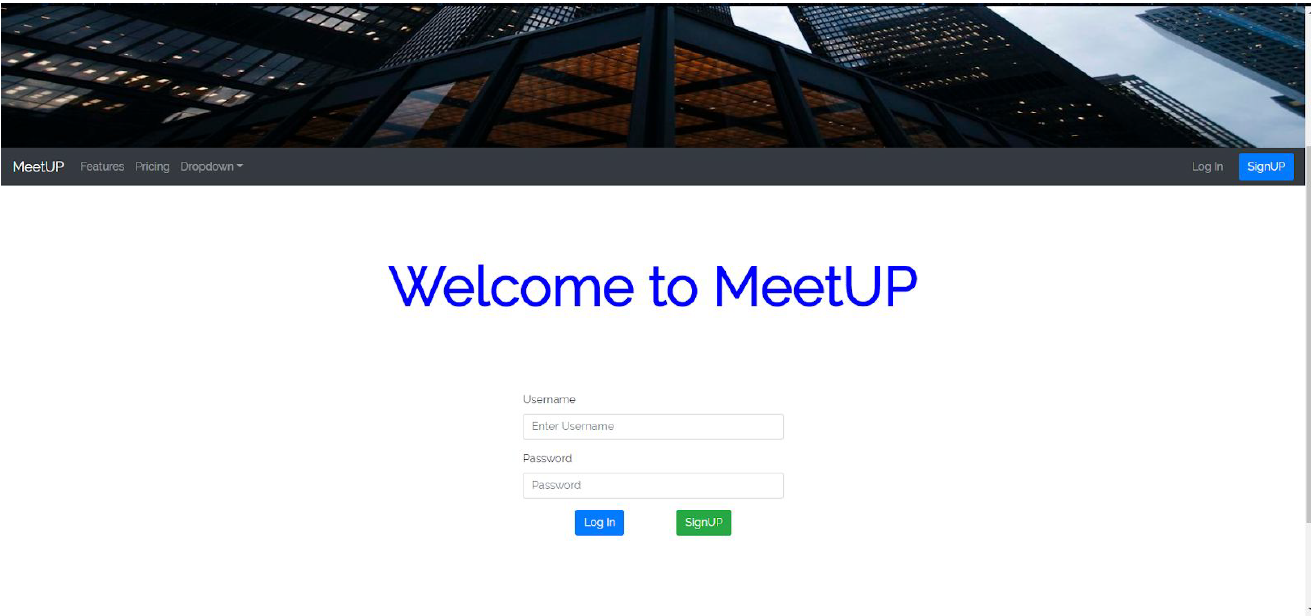
yarn start







The two use cases that we developed a front-end and database for Assignment A3 where the register and login use cases.



The two use cases that we developed a front-end and database for Assignment A4 where the create group and add group members use cases.

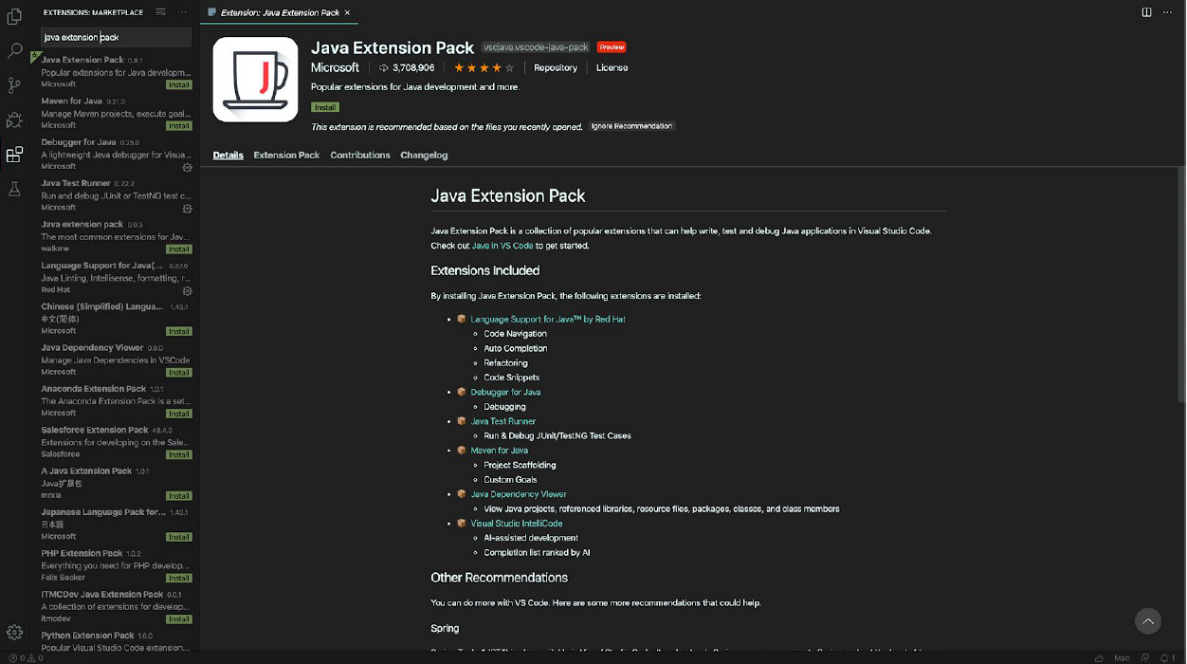
Testing

Testing Java in Visual Studio Code can be enabled by the Java Test Runner extension. The extension supports JUnit 4, JUnit 5, and TestNG. It can run/debug test cases with customizable configurations and show test logs and reports.

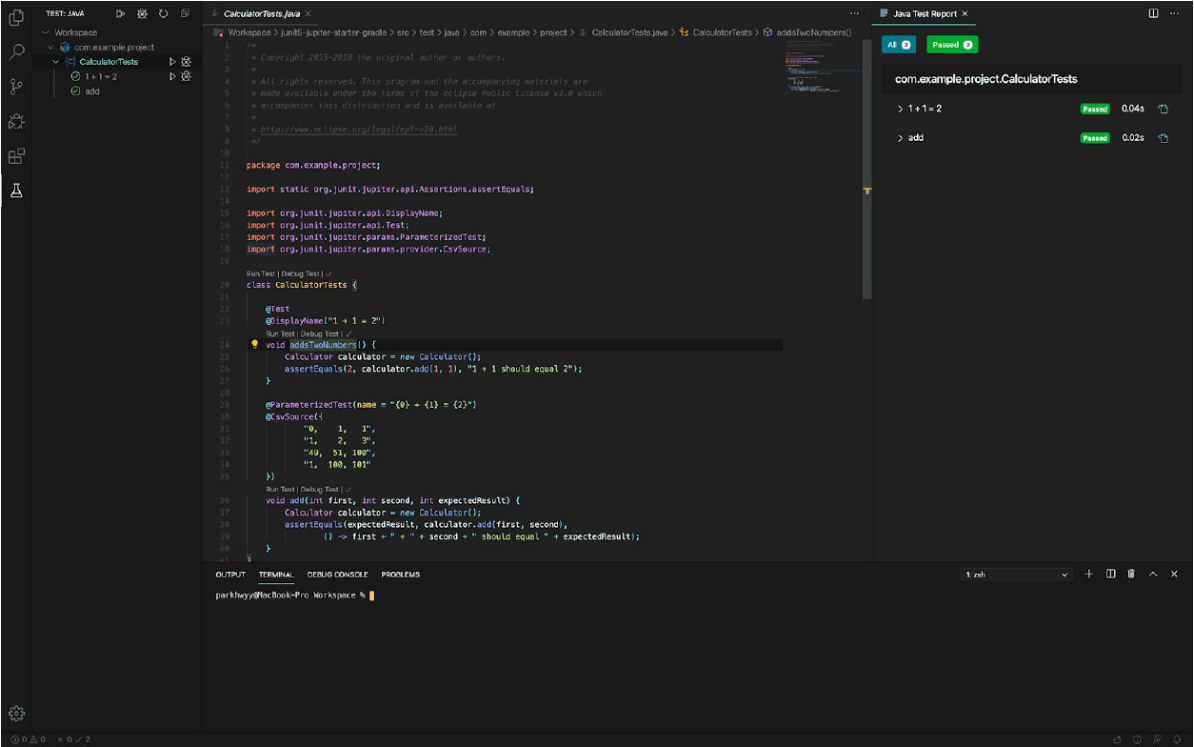
Assuming Visual Studio Code is ready, the following extension package can be installed, which

includes all of these necessary extensions:

* 1. Language Support for Java by Red Hat
  2. Debugger for Java
  3. Java Test Runner



The following screenshot captures a sample test using JUnit in Visual Studio Code.



**Use Case 1: Register Account**

Function: register (String username, String password)

Input partition: unique | existing | empty (for username)

valid | invalid | empty (for password)

Test specifications:

register (unique username, valid pw)

register (existing username, valid pw)

register (unique username, invalid pw)

register (existing username, invalid pw)

register (unique username, empty pw)

register (existing username, empty pw)

register (empty username, valid pw)

register (empty username, invalid pw)

register (empty username, empty pw)

Test cases:

register (“user00”, “User001\*”)

register (“admin”, “User001\*”)

register (“user00”, “user00”)

register (“admin”, “user00”)

register (“user00”, “”)

register (“admin”, “”)

register (“”, “User001\*”)

register (“”, “user00”)

register (“”, “”)

**Use Case 2: Login to Account**

Function: login (String username, String password)

Input partition: correct | incorrect | empty (for both username and password)

Test specifications:

login (correct username, correct pw)

login (incorrect username, correct pw)

login (correct username, incorrect pw)

login (incorrect username, incorrect pw)

login (correct username, empty pw)

login (incorrect username, empty pw)

login (empty username, correct pw)

login (empty username, incorrect pw)

login (empty username, empty pw)

Test cases:

login (“admin”, “Admin001\*”)

login (“admit”, “Admin001\*”)

login (“admin”, “Admin00”)

login (“admit”, “Admin00”)

login (“admin”, “”)

login (“admit”, “”)

login (“”, “Admin001\*”)

login (“”, “Admin00”) login (“”, “”)

**Use Case 1: Register Account**

Function: register (String username, String password)

Input partition: unique | existing | empty (for username)

valid | invalid | empty (for password)

**Test #1 testValidReg ()**

|  |  |
| --- | --- |
| **Test ID** | **Test #1 testValidReg ()** |
| Purpose of Test: | To verify if a user will be able to register with a unique username and valid password. |
| Test Environment: | Mac OS using JUnit |
| Test Input: | Unique username and valid password (ex. “admin”, “Admin001\*”) |
| Expected Result: | Should display the confirmation message and redirect to the login page. |
| Likely Problems: | It might not redirect to the login page after a successful registration. |

**Test #2 testExistingUser ()**

|  |  |
| --- | --- |
| **Test ID** | **Test #2 testExistingUser ()** |
| Purpose of Test: | To verify if a user cannot register with a username that already exists. |
| Test Environment: | Mac OS using JUnit |
| Test Input: | Existing username and any password (ex. “user00”, “User001\*”) |
| Expected Result: | Should display the error message to have the user to choose another username. |
| Likely Problems: | The same username with different capitalization might be considered unique. |

**Test #3 testInvalidPw ()**

|  |  |
| --- | --- |
| **Test ID** | **Test #3 testInvalidPw ()** |
| Purpose of Test: | To verify if a user cannot register with a password that does not meet the requirements. |
| Test Environment: | Mac OS using JUnit |
| Test Input: | Unique username and password that does not meet the requirements (ex. “admin”, “admin”) |
| Expected Result: | Should display the error message to remind the password requirements. |
| Likely Problems: | A password that does not meet the requirements might not be detected |

**Test #4 testEmptyReg ()**

|  |  |
| --- | --- |
| **Test ID** | **Test #1 testValidReg ()** |
| Purpose of Test: | To verify the registration page when either field is empty and submit button is clicked. |
| Test Environment: | Mac OS using JUnit |
| Test Input: | Empty username and/or empty password (ex. “”, “User001\*” / “admin”, “”) |
| Expected Result: | Should display the error message to remind all fields should not be empty. |
| Likely Problems: | The empty string might be accepted as a valid input. |

**Use Case 2: Login to Account**

Function: login (String username, String password)

Input partition: correct | incorrect | empty (for both username and password)

**Test #1 testValidLogin ()**

|  |  |
| --- | --- |
| **Test ID** | **Test #1 testValidLogin ()** |
| Purpose of Test: | To verify if a user will be able to log in with a correct username and password. |
| Test Environment: | Mac OS using JUnit |
| Test Input: | Correct username and password (ex. “admin”, “Admin001\*”) |
| Expected Result: | Should redirect to the homepage. |
| Likely Problems: | It might not redirect to the homepage after a successful login. |

**Test #2 testInvalidUser ()**

|  |  |
| --- | --- |
| **Test ID** | **Test #1 testValidLogin ()** |
| Purpose of Test: | To verify if a user cannot log in with a username that is not registered. |
| Test Environment: | Mac OS using JUnit |
| Test Input: | Invalid username and any password (ex. “user99”, “User001\*”) |
| Expected Result: | Should display the error message to have the user to check the username. The message will also remind the user to register first with the link provided. |
| Likely Problems: | The same username with different capitalization might be considered invalid. |

**Test #3 testWrongPw ()**

|  |  |
| --- | --- |
| **Test ID** | **Test #1 testValidLogin ()** |
| Purpose of Test: | To verify if a user cannot log in with an incorrect password. |
| Test Environment: | Mac OS using JUnit |
| Test Input: | Valid username and incorrect password (ex. “admin”, “user00”) |
| Expected Result: | Should display the error message to have the user to check the password. |
| Likely Problems: | An incorrect error message might be displayed to the user. |

**Test #4 testEmptyLogin ()**

|  |  |
| --- | --- |
| **Test ID** | **Test #1 testValidLogin ()** |
| Purpose of Test: | To verify the login page when either field is empty and submit button is clicked. |
| Test Environment: | Mac OS using JUnit |
| Test Input: | Empty username and/or empty password (ex. “”, “User001\*” / “admin”, “”) |
| Expected Result: | Should display the error message to remind all fields should not be empty. |
| Likely Problems: | The empty string might be treated as a valid input. |

Appendix

Youtube link

<https://www.youtube.com/watch?v=JsBMbk5mbAU&feature=youtu.be>

or link to video:

<https://youtu.be/0cXyuNclhN0>

Discord link

https://discordapp.com/channels/670083895097688087/670083895097688099

Github Repository Link

<https://github.com/ConnorLacy/CSC-SWE-CEDAH/tree/master/src>