›SW Engineering CSC648/848

WhereWeEatinMeow

Restaurant Choice Solution

Section 04, Team 03

Raul Cardenas – *Team Lead / Infrastructure Lead*
Andy Nguyen – *Github Master / Back-end*
David Ye Luo – *Database Admin / Front-end*
Dean De Leon – *Scrum Master / Front-end*
Fred Gunther – *Back-End Master / Front-end*
Jonathan Ko – *Back-End Master / Front-end*

"Milestone 2"

March 29th, 2023

Revision History Table

Revision ID	Revision Date	Revised By
1.0	March 8, 2023	Raul Cardenas
2.0	March 29, 2023	Raul Cardenas

Content and structure for Milestone 1 document for review

Functional Requirements V2

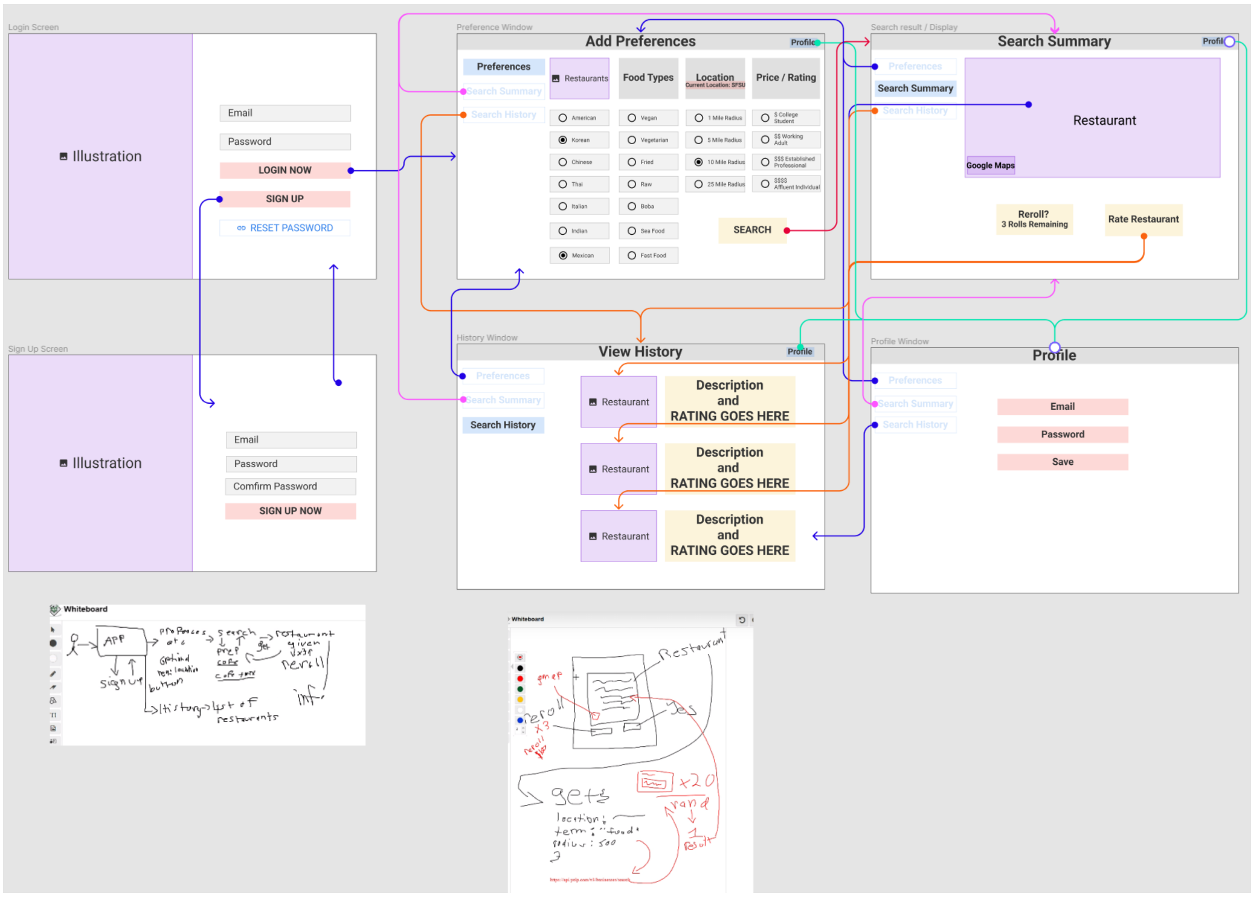
Revised from Milestone 1

🟦	Must-have	🟥	Desired	🟩	Opportunistic

	ID	Functional Requirement Description	Details (As Needed)
🟦	1	Connect to internet: this app requires internet usages	1. User should be able to connect to our website.
🟦	2	Sign-up/login: Users can register for accounts	1. Users can sign up with an email, and password. 2. Users can then login with said email and password.
🟦	3	Choosing Preference: Users should be able pick restaurants, food types, place	1. Users can choose different food types to be considered in the selection (display criteria they want to use). 2. Users can also choose maximum distance from their set location. 3. Users can set the minimum star rating of a restaurant to be considered.
🟦	4	Display preferences: App shows user's selected food, location preferences before selecting a restaurant.	1. Users can update their preferences before submitting a search
🟦	5	Display randomly chosen restaurant based on preferences	1. After preferences are chosen and user is ready to be given a restaurant, the app will provide info on one restaurant based on preferences
🟦	6	Pull from other/similar restaurant provider app (Yelp)	1. App pulls the number of restaurants from other websites/app based on preferences. Third Party API: Yelp ● Name: Business Search ● Path: /businesses/search ● Description: Search for businesses by keyword, category, location, price level, etc.

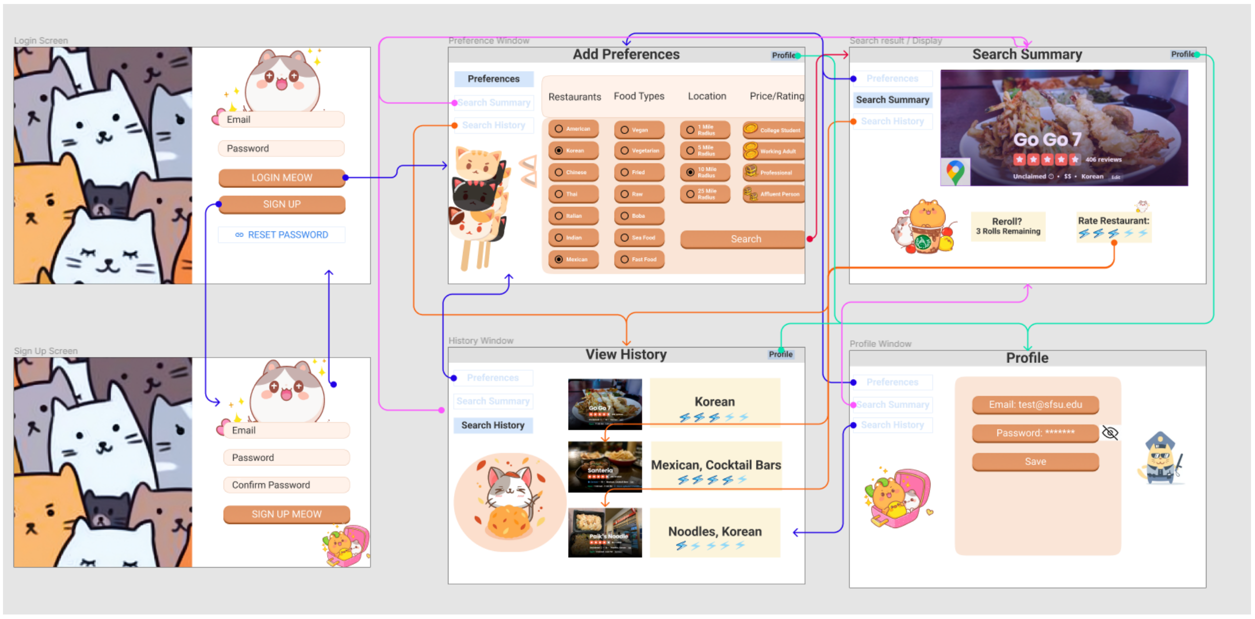
🟦	7	Calculate/find correct restaurant within distance from set location if set.	1. Users should get restaurant(s) within preferred distance from set location
🟥	8	Confirm to be given a restaurant.	1. Users should be able to accept the generated result or regenerate another result.
🟥	9	Allow for “rerolls” and limit how many can be performed.	1. User can be given another random restaurant based on preferences or criteria. 2. User has a maximum of three rerolls before a cooldown.
🟩	10	User can rate each restaurant out of 5 stars.	1. User should be able to rate the restaurants. (Maybe in the future we can use this data as as a user preference)
🟩	11	Search for past restaurants given	1. User can search past rerolls or randomly selected restaurants. 2. The user’s rating of the restaurant will be shown for each restaurant.
🟩	12	Be able to order food from a restaurant	1. App allows users to make an order from a restaurant.
🟩	13	Promote another restaurant in promoted section	1. App displays promoted restaurants in the promoted section for exposure to users.

UI Mockups and Storyboard (High Level)



5

GUI Mockups and Storyboard (High Level)



High Level Architecture, Database Organization

Our main goal of the application is to help users choose a restaurant. The application doesn't need a database to function but it could have better user experience.

So the user puts in the preferences that they want and make a search. Without a database, the user wouldn't have the option to choose their old preferences. They would need to manually input the same preferences as before which requires the users to remember their preferences and clicks.

Our team thought that allowing users to reuse their past result would be a quality of life improvement for the users. So, our database design is to store those result so that users can view and reuse their preferences.

High Level Architecture

The users table serves as an identity to differentiate from users to users. First time users will need to register. When they do, they will insert a row to the User table in the database which contains the Email and Password. The database will handle with generating the user_id and the user_id will be used to get data from other tables. For users and the backend, the user_id doesn't matter much, but it is very important for the database. In the database, the user_id is used to find which data corresponds to that user. This is why every other tables especially *_preferences have a foreign key of the user_id.

One of the question is how can we add multiple user preference choices. We do that by adding ids as a primary key. Again, these id only matters to the database to allow same users to have multiple data.

Now that we can store multiple preferences for the user, they need to retrieve the data and be able to correspond it to other tables. So the, the other important attribute is the save_history_count. This attributes help link between other tables. Same save_history_count value(integer) means that they came from the same search. For example, if the user's last count for a user is 9 and we want the latest area and price preferences. Then we will need the user_id and and find when save_history_count is 9 for area_preference and price_preference.

Note: this is somewhat over-engineered approach for saving many history. We could make the database much simpler in which users can only store 1 history.

APIs

When the user register to our website, they add an entry to the user table. For continuing users, they will just to log in.
Then the user is presented to the preferences view where each preference selection corresponds to their corresponding table. In addition, there will be a variable that we need to prepare. The backend will increment get the save_history_count and increment the value from the history table. This new save_history_count value and the preferences are inserted to their corresponding tables.

User

User
user_id UNIQUE PK
Email STRING
Password STRING

Preference

type_of_restaurant_preference

restaurant_pref_id UNIQUE
user_id FOREIGN KEY
type VARCHAR(32)
save_history_count INT

area_preference
area_pref_id UNIQUE
user_id FOREIGN KEY
radius INT
save_history_count INT

rating_preference
rating_pref_id UNIQUE
user_id FOREIGN KEY
rating INT
save_history_count INT

price_preference
price_pref_id UNIQUE
user_id FOREIGN KEY
price INT
save_history_count INT

food_preference
food_pref_id UNIQUE
user_id FOREIGN KEY
food_type string

save_history_count int

Event

history
history_id UNIQUE
user_id FOREIGN KEY
restaurant_name VARCHAR
date timestamp
save_history_count INT

reroll
reroll_id UNIQUE
user_id FOREIGN KEY
count INT
time_since_last_used INT

Add/Delete/Search architecture	Functional Requirement
Add/Delete for Users	When users register
Search for Preferences for Events	When user set their preferences for restaurant searching/rerolling
Search/Display for Events	When users search for restaurants, history, or want to reroll the search

Data Definitions V2

High Level Data Definition

Revision from Milestone 1

Primary Data Name	Definition	Usage
user	A person who can sign up with an account on the application. Ex: Name: Andy user_id: 1 Preferences: Mexican food	The necessary information of the registered users is collected and stored. This will be used to have default settings so that the user doesn't need to input every time.
history	The last 5 restaurants the app showed to the user	The history of restaurant information. The user can pay for more search history if they want at $10 per extra history list. Ex: 10 search history, costs $50/mo
Reroll	Amount of times a user can reroll. Reset every 24 hours. Ex: Reroll: 3	The user can use reroll to change restaurants to 3 times per day. Can pay for more per day usage at $100.

Data Definition

Revision from Milestone 1

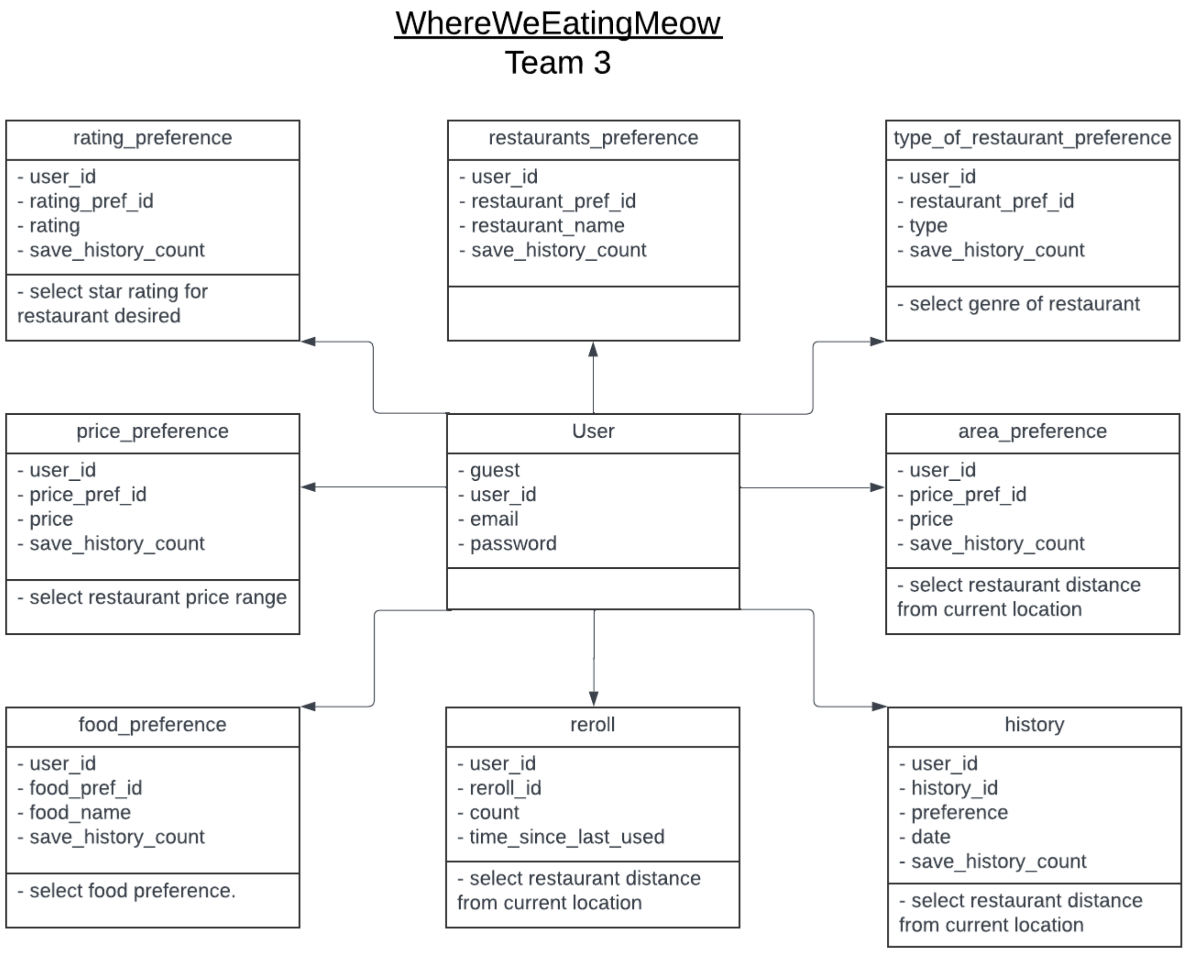
Primary Data Name	Sub-Data
user	● user_id UNIQUE ● email ● password
history	● history_id UNIQUE ● user_id FOREIGN KEY ● restaurant_name VARCHAR ○ This is the result. This is stored so that we can ● date DATE ○ Used to tell the date that the user made the search ● save_history_count

	○ Used to tell how many times the users have searched ○ Also used to correspond how other tables relate to each other. ■ Same number means that they came from the same search
type_of_restaurant_preference	1. restaurant_pref_id UNIQUE 2. user_id FOREIGN KEY 3. Type VARCHAR ○ Korean ○ Chinese ○ Mexican… 4. save_history_count ○ Used to tell how many times the users have searched ○ Also used to correspond how other tables relate to each other. ■ Same number means that they came from the same search
area_preference	1. area_pref_id UNIQUE 2. user_id FOREIGN KEY 3. radius INT ○ unit in miles ■ US is more accustomed to miles ■ Note for backend: the api uses meters so we need to handle conversion. 4. save_history_count ○ Used to tell how many times the users have searched ○ Also used to correspond how other tables relate to each other. ■ Same number means that they came from the same search
rating_preference	1. rating_pref_id UNIQUE 2. user_id FOREIGN KEY 3. Rating INT ○ 1 through 5 rating. ■ 1 for worst

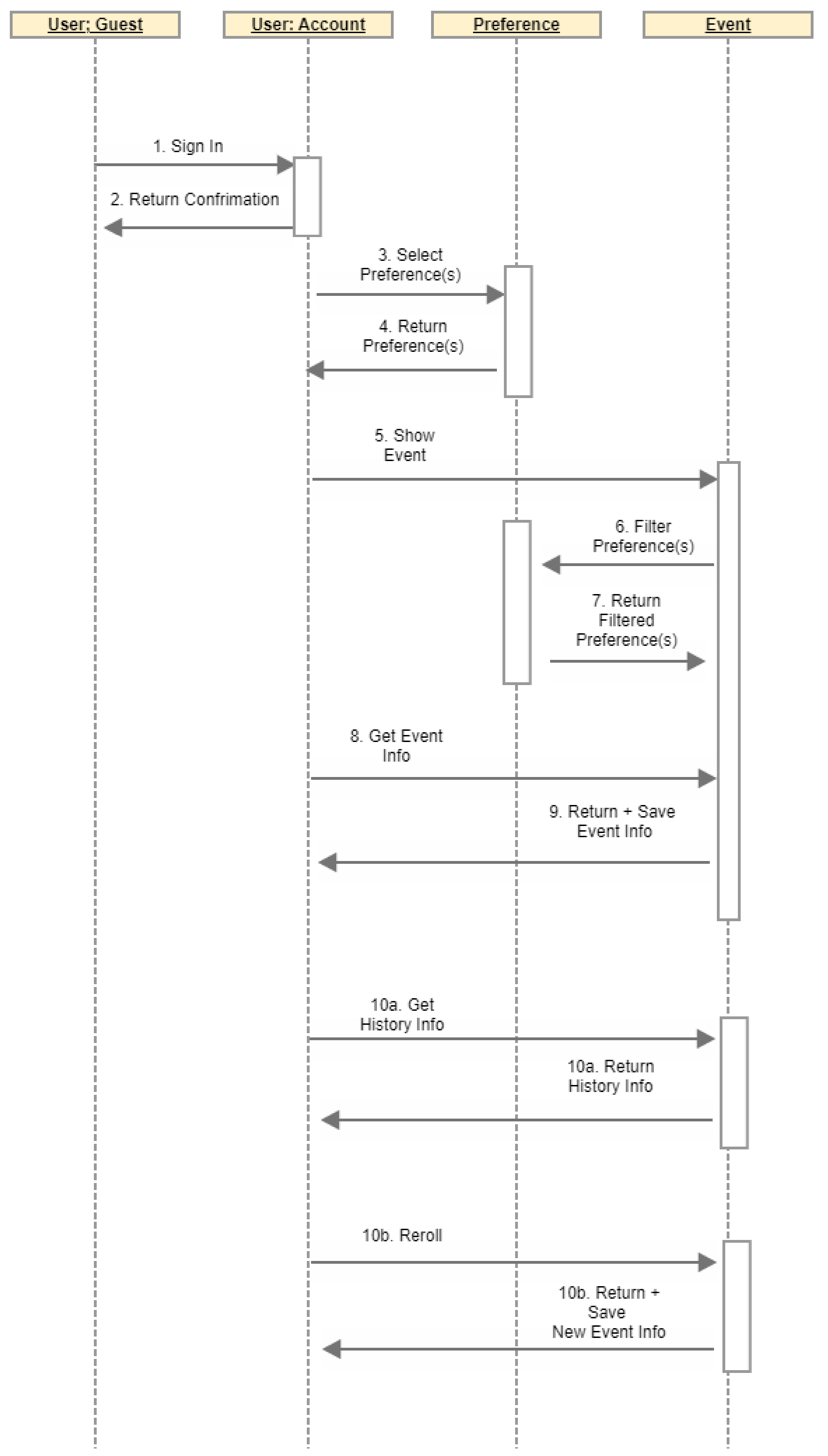
	■ 5 for great 4. save_history_count ○ Used to tell how many times the users have searched ○ Also used to correspond how other tables relate to each other. ■ Same number means that they came from the same search
price_preference	1. price_pref_id UNIQUE 2. user_id FOREIGN KEY 3. price INT ○ 1 for cheap ○ 5 for expensive 4. save_history_count ○ Used to tell how many times the users have searched ○ Also used to correspond how other tables relate to each other. ■ Same number means that they came from the same search
food_preference	1. food_pref_id UNIQUE 2. user_id FOREIGN KEY 3. food_type VARCHAR ○ Vegan ○ Vegetarian ○ Fried… 4. save_history_count ○ Used to tell how many times the users have searched ○ Also used to correspond how other tables relate to each other. ■ Same number means that they came from the same search
reroll	● reroll_id UNIQUE ● user_id FOREIGN KEY ● count INT ○ User have 3 rerolls every hour. These are meant to have users do another search if they don't like their choice. ■ Was meant more of

	accidentally press the search button ■ We also didn't wanted the user to have unlimited choice because it will defeat the point of our app. ● time_since_last_used

High Level UML Diagram



High Level Sequence Diagram



Identify Actual Key Risks For Project At This Time

- Skills Risks and mitigation plan
    - If Django tutorials are not completed, blocks may occur when doing coding tasks for the app.
        - To resolve:
            - Complete Django tutorials 1-6 and mark complete on Jira.
            - Ask for help if tutorials cannot be completed.
    - If a bug or block comes up when working on a task (even after completing Django tutorials), completion of the task will be delayed and may conflict with other members' work.
        - To resolve:
            - Work with team lead and other members to debug.
            - Go back to Django tutorials or look at other online resources for whatever technology is involved.
            - Understand all of the code in the project.
- Schedule Risks
    - If tasks are not completed on time, other members' progress and progress on the project as a whole can be delayed.
        - To resolve:
            - Remember to create tasks on Jira for oneself and look at Jira consistently.
            - Meet with the team lead and other members to make clear what tasks have the highest priorities.
    - If task priorities are not known, important tasks may not be completed before the correct date.
        - To resolve:
            - Team lead must be sure to clarify dates for tasks and which tasks should be in progress.
            - Be sure to move tasks in Jira from in development to in progress.
- Teamwork Risks
    - If backend and frontend teams do not communicate, potential blocks can occur.
        - To resolve:
            - During meetings, be sure to exchange issues and pain points.
            - Be consistent with documentation.
            - Know each others' limitations.
    - If work or other life matters conflict with meetings, upcoming tasks and current tasks will not be known to members.
        - To resolve:
            - Always meet with team lead to cover whatever was missed.
            - Be sure to review meeting minutes after work.

- Communicate with team diligently to stay on top of tasks and be on the same page.

Project Management

Our group contains one infrastructure lead (Team lead), one github master (also devops lead for GitHub Actions), one database lead, one scrum master, two front-end and two back-end devs with each having their own lead. We all play a part in owning a “lead” component of the project. We took the approach of trying to round out our knowledge a bit more during this milestone incase one of us has other duties or get sick. It did cause a bit of momentum loss but we feel its necessary for the rest of the milestones.

We have two weekly meetings as stand ups and check ins. Besides the in-class meeting, we meet every morning at 8am on Saturdays to discuss what needs to be worked on and if needs to be collaborated with we can stay on longer (with every couple hours taking a break). Main collaboration is done thru discord with every meeting having its own meeting minutes incase we need to recall what was discussed

As a team lead, we experimented a bit more during this milestone due to many members having either increased obligations or other factors (such as power outages, floods, family, etc). Planning in jira definitely helped. For future tasks, I plan to get the tasks created earlier so that we can spend more time planning early on to really understand the requirements to give us more time for development work.