jrios777 Sixto Josue Rios
xalicex1 Alice Cheng
tadrip Tadrill Perry
jli93 Jessica Li
cho1993 Michelle Cho
jhzhu10 Junhao (Ian) Zhu
kkbrandt Kaleo Brandt
thaond Thao Nguyen Dang

Architectural Design

System Architecture

Major Modules/Classes

- Account (Abstraction)
    - User account object that represents a single user account of the application. It stores all the required/optional information of the user (such as name, email, gender, etc)
- AccountManager (Utility Class)
    - Has the methods that will be used to manage user's current accounts (editing, adding, deleting information).
    - Acts between the activity classes that listen to events and the Database that stores all of the data
- DatabaseManager (Utility Class)
    - Has the methods that will be used to manage the data and information needed for this application. Allows other classes to look up, edit, add information to the database.
- MealPlan (Abstraction)
    - An object that represents a single MealPlan that exists in the application. This stores all of the information of the event (data, time, host, etc.).
- MealPlanManager (Utility Class)
    - Has the methods that will be used to manage the MealPlans. This allows MealPlans to be changed or updated or deleted.

User Interaction Classes

- CreateMealPlanActivity
    - Listens to see if the user wants to create a new MealPlan and notifies the MealPlanManager to carry out the appropriate work to add the meal plan for the user.
- JoinMealPlanActivity
    - Listens to see if the user wants to join an exisiting meal plan. It notifies the MealPlanManager to carry out the appropriate work to join the meal plan.
- SignInActivity
    - Listens to see if the user needs to sign in. It notifies the AccountManager to verify any information to allow the user to sign in to their account with all their information and MealPlans.
- SignUpActivity

    - Listens to see if the user does not have an existing account and notifies the AccountManager to create a new Account that interacts with the DatabaseManager to add the new information to the application.
- ManageAccountFragment
    - Listens to see if the user wants to update any of their information and notifies the AccountManager to carry out the respective events.
- HomeFragment
    - Listens to the interaction of user and the Home Page of the application to allow different views to appear. This notifies the UI to switch tabs when necessary and the DatabaseManager to retrieve the appropriate data to be displayed to the user.
- MyMealFragment
    - Displays all the lists of meal plans that user is hosting or joining
    - Listens to add meal plan button to start CreateMealPlanActivity
- SettingsActivity (Extra feature)
    - Listens to see if the user decides to change/update the settings.
- HelpActivity (Extra feature)
    - Listens to see if the user clicks on a help button that will display information about the application to the user.

Data

- Our system stores user information such as name, gender, year, email, and phone, account information such as email and password, and meal plan information, such as all the meal plans created and their hosts, food, date, time, location, attendees, and description. For account authentication we'll be using Amazon's Android SDK to connect to Amazon Web Services' (AWS) Cognito/Sync service and for the databases storing user and meal plan information we'll be using Java's JDBC library to talk to AWS' RDS databases. The schema is as follows:
    - Account(id, name, gender, year, email, activated, validationCode, major)
    - Attending(planID, attendeeID)
    - MealPlan(planID, hostID, food, location, dateTime, description)

Alternative Designs

- One alternative design for displaying the Meal plans' feeds/homepage, user's current meal plans, and user's account information was to have separate activities instead of one activity with fragment tabs. However, we chose to go with tabs to allow the user a more efficient, quick way of maneuvering between frequent visited sections of the application.
    - Pros of alternative: It can potentially simplify the user interface because there would not be any tabs showing and it is easier to implement.
    - Cons of alternative: It usually does not allow the client to switch between different sections in one click.
    - Pros of chosen design: It helps the user switch from screen to screen in an efficient manner and gives them the freedom to choose which screen they want to look at.
    - Cons of chosen design: More difficult to implement.
- Another alternative design was to store and query all information from the database when needed from whatever class. The design we chose was to store in memory account information and query information, providing a refresh action to reload data up to date.
    - Pro of alternative: this would reduce coupling between classes and whatever the query returns will be up to date.

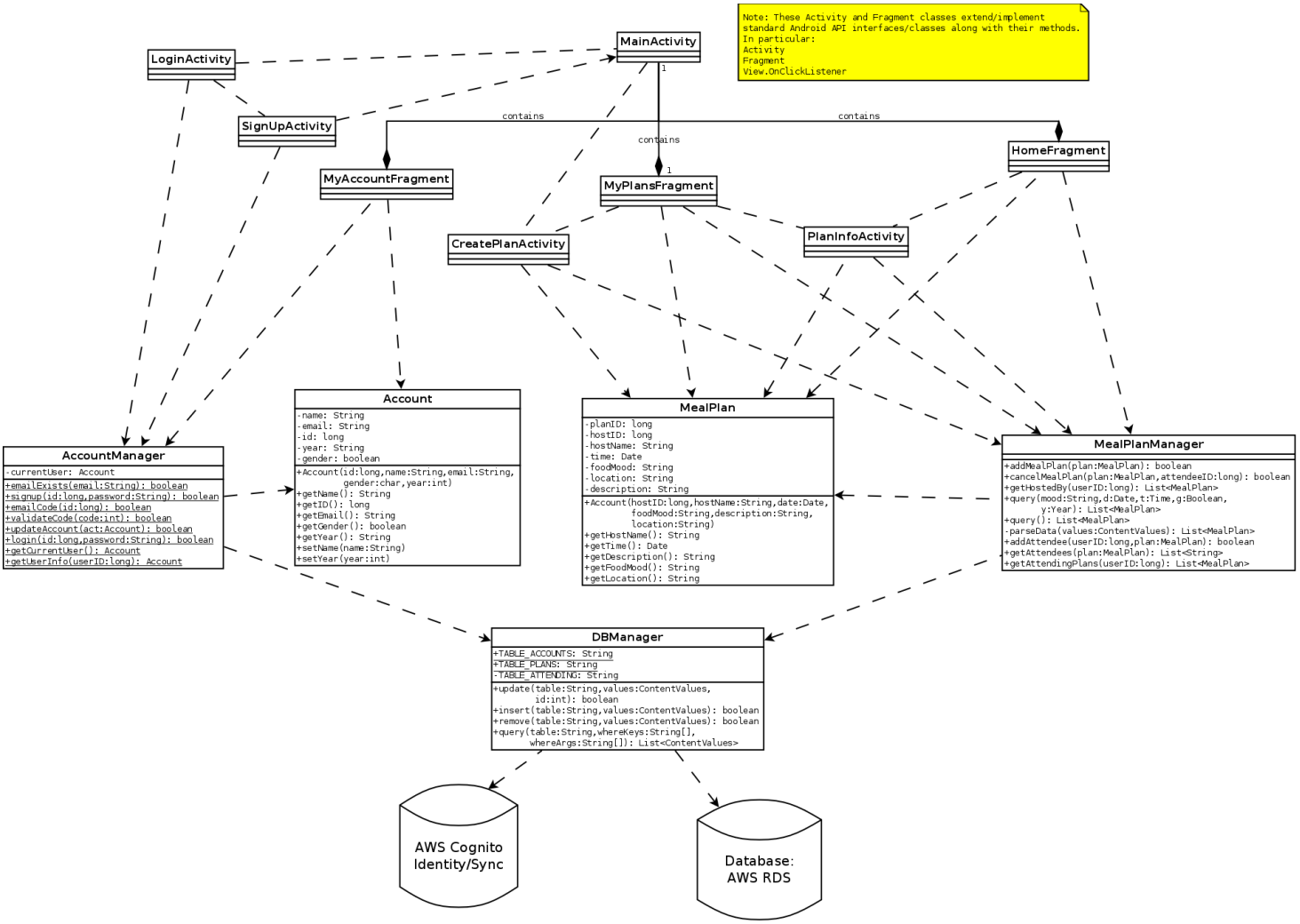
- Con of alternative: Expensive and slower to receive information. Needs to load dynamically from database server.
- Pro of chosen design: easy and fast access to information.
- Con of chosen design: More dependencies. Chance of returning out of date information.

Assumptions

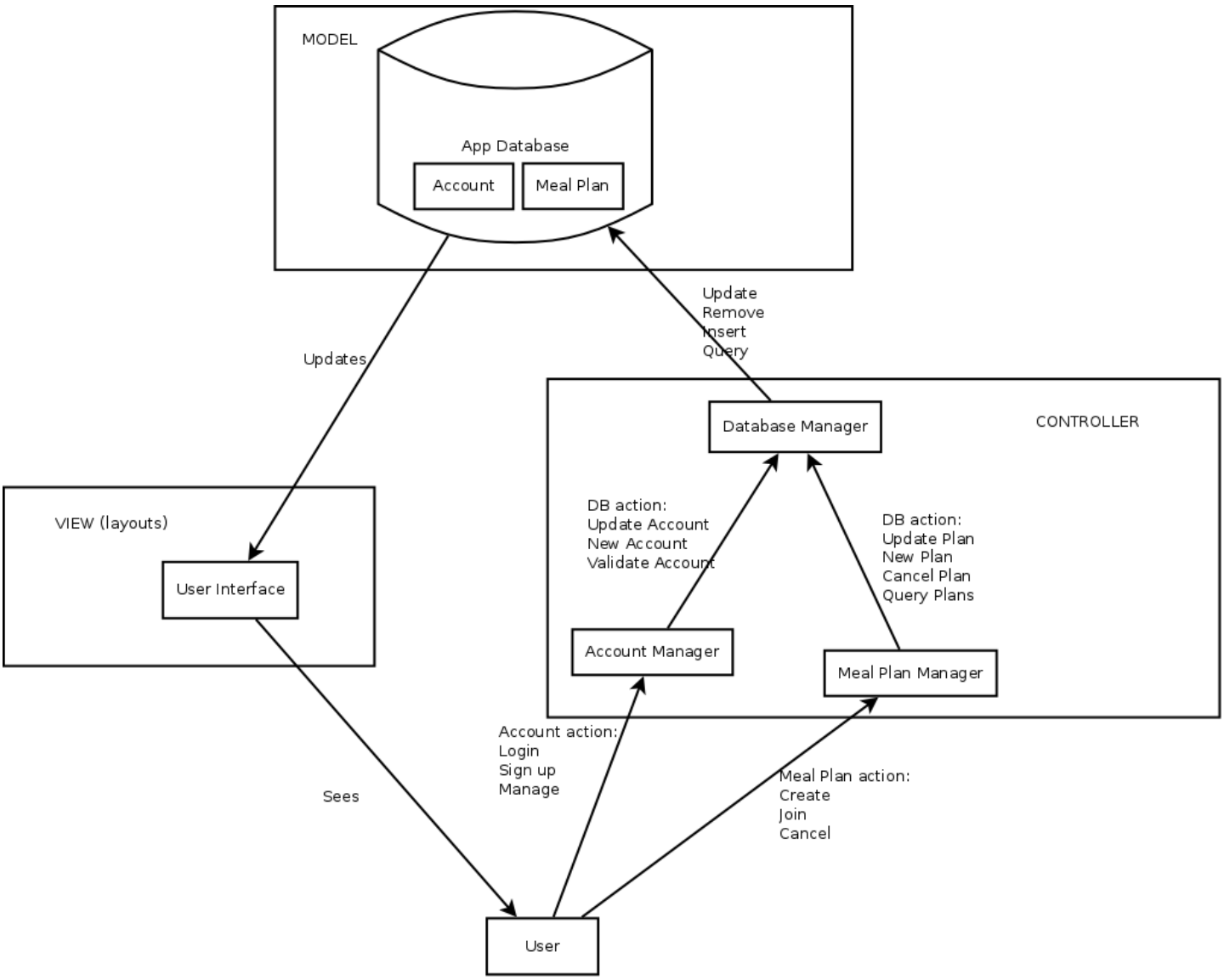
- User has internet connection for application to function properly
- User will need to have an account to be able to use the application
- User must have UW (CSE) netID

Diagrams

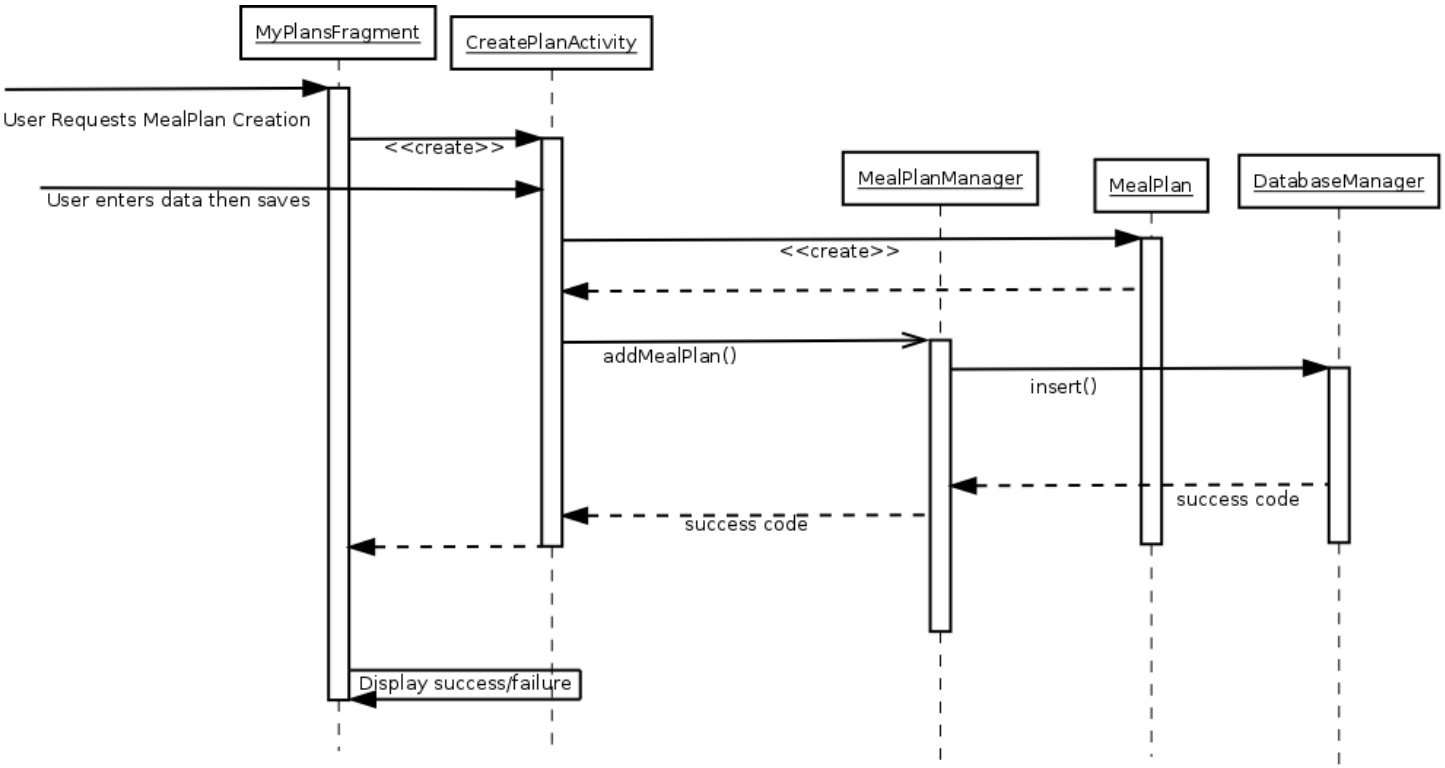
UML Class Diagram:



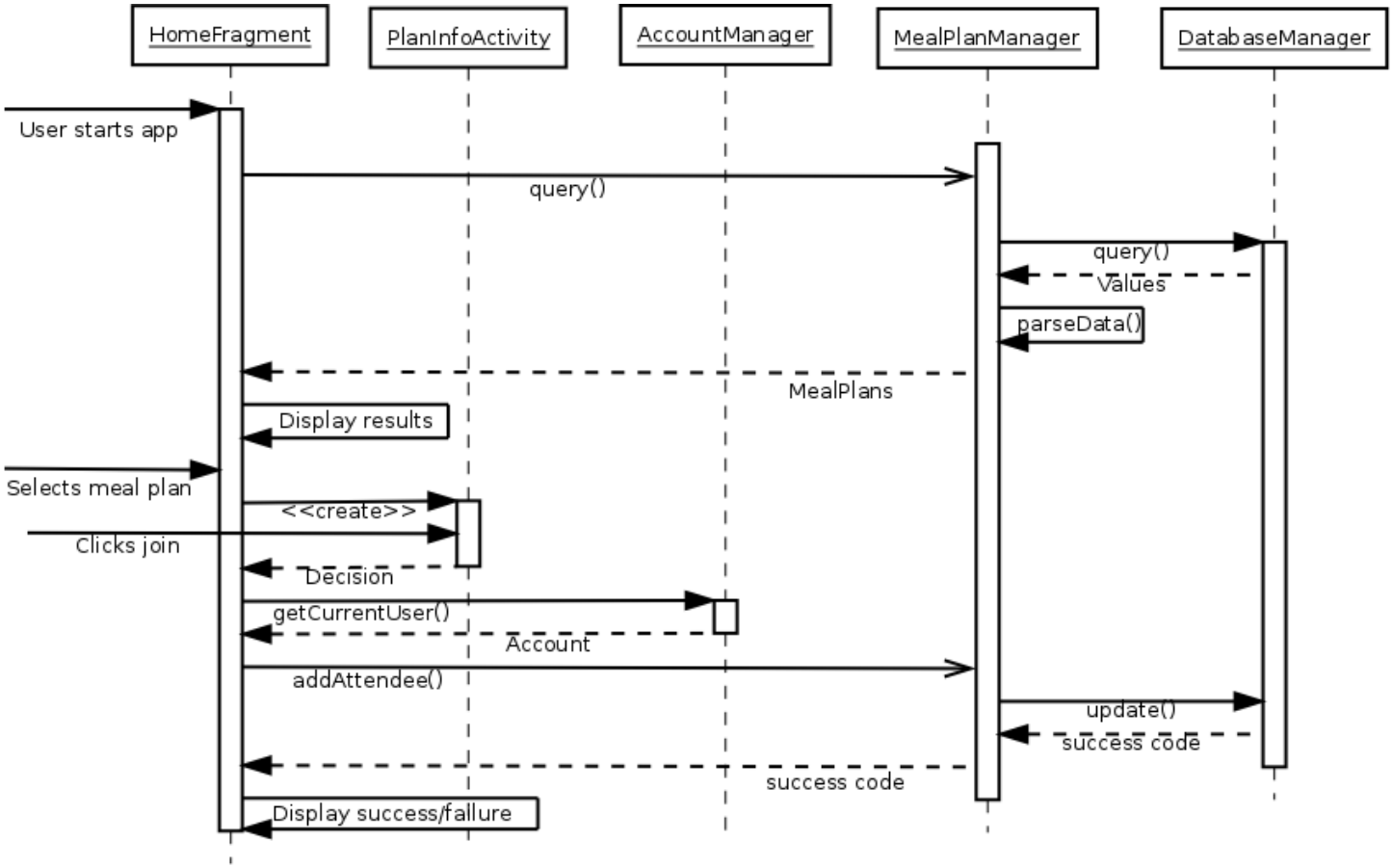
A second view at our system from an MVC design:



Sequence Diagram for Creating a Meal Plan



Sequence Diagram for Joining a Meal Plan



Process

Risk Assessment

The most serious risk is if the integration between the back-end and front-end components of the system fails. The likelihood of this occurring is high since the teams working on the front-end and back-end are different. Therefore, the possibility of a bug or problem occurring substantially increases. The impact of this risk can range from low to high depending on what part of the system fails. If the integration completely fails, then the app will not work. If there is a very small integration error, then the error could go undiscovered. In order to minimize this risk, we've assigned the integration team to facilitate communication. Additionally, the testing team will carry out black box experiments concerning everything from documentation to app usability in order to detect whether this problem occurs. If a problem is detected, it will be posted on the bug issue tracker, where a team member will be delegated to fix the problem.

Another major risk is the failing to provide and update information instantaneously. The likelihood of this occurring is high, because many group members have not worked with networking before. Furthermore, a bug may occur if the device using the app fails to connect to the Internet in the time between when the user submits their data and the time it reaches the server. The impact of this risk may also range from low to high, depending on how often the failure occurs. We know these estimates are true because in an earlier group meeting, we proposed an idea of how to handle the networking, but then discovered many possible places where a bug may be introduced within our plan. In order to reduce the likelihood of this risk, a few group members will be required to read guides on networking and we thoroughly discussed how we plan to approach this problem in our group meeting. We have also decided to try to run network operations on a separate thread from the user interface as recommended by the Android development guide. In order to discover potential bugs, the testing team will develop black box tests that attempt to create and retrieve information when there is no Internet connection or the Internet connection breaks down in the middle of the app. To mitigate this risk, we plan to test for networking bugs early, so that we have time to find a solution for the bug.

Implementing the software as an Android app is another major risk since most group members have no experience with Android development or SQL. Therefore, there is a high possibility that the software may fail to work due to lack of knowledge about how to implement the app. This would highly impact the project if it occurs. For this, all members have already done an Android tutorial and will be required to read guides on Android development. If this problem occurs, it will be very apparent, so we should not need to do anything additional to detect the problem. To mitigate the impact of this risk, we may need to end up simplifying the interface or functionality of our app.

Most members have also not used the external software and libraries before, so there is also a high possibility that the software may fail to work due to not knowing how to use the external software. For example, we have considered using external software to manage login information to make our database more secure. The impact of failing this software is medium, because there are many alternative software products that implement this functionality. However, our database may be at risk to SQL injection or other unsavory practices. In order to reduce this risk, a few group members will be delegated to do related tutorials for the external software. The testing team will run black-box tests to check whether the functionality of the external software works, but we will not be able to detect any internal bugs within the external software. If external software fails for login, we can attempt to implement our own login database using SQL.

Another risk which was slightly mentioned in the previous one is the possibility that our database and network will not be secure. The likelihood of this occurring is very high, because our group members have limited knowledge about security. The impact of this is low regarding to the completion of this project since an unsecure app still works, but would have an extremely high impact if this software becomes available to the public. In order to reduce the likelihood of this occurring, a few group members will be required to read guides on security. We will not be able to detect whether this problem occurs, but we will know that it is always likely that our database and network are not very secure. In order to mitigate this risk, we will try use reputable external software for private account information, and hope that people who created them have a strong knowledge in security. We will also limit user input by preventing them from using symbols that may cause SQL injections.

Project Schedule

Week – Group goals	Back End – Alice and Josue	Front End – Jessica and Thao	Integration – Ian and Michelle	Testing – Kaleo and Tad
4/27 – Zero feature release: Login System complete 5/1 – due	**Alice**: Implement Account login/signup methods in DBManager **Josue**: Setup the AWS instances /databases needed	**Jessica:** Implement MainActivity & MyAccountFragment **Thao:** Implement LoginActivity & SignUpActivity	**Michelle:** Implement Account **Ian:** Implement core AccountManager methods	**Tad**: Write black box tests for Account and AccountManager. **Kaleo**: Write UI/Integration tests for signing up, logging in & logging out
5/4 – Beta Release: MealPlan System complete 5/8 – due	**Alice**: Implement insert and remove for DBManager. **Josue**: Setup tables in database and implement query	**Jessica**: Implement MyPlansFragment & HomeFragment **Thao**: Implement CreatePlanActivity & PlanInfoActivity	**Michelle**: Implement MealPlan **Ian**: Implement core MealPlanManager methods	**Tad**: Write black box tests for MealPlan and MealPlanManager. **Kaleo**: Write UI/Integration tests for creating meal plan, joining meal plan, & deleting meal plan
5/11 – Stretch Feature: Implement account verification and forgot password validation	**Alice**: Implement DBManager methods needed for verifying accounts and validating forgotten password codes **Josue**: Setup the any necessary tools to complete the back end for the feature	**Jessica**: Design and implement the user interface for account verification **Thao**: Design and implement the user interface for forgotten password codes	**Michelle**: Implement AccountManager methods needed for account validation **Ian**: Implement AccountManager methods needed for forgotten password recovery	**Tad**: Write tests for forgotten passwords **Kaleo**: Write tests for account verification
5/18 –	**Alice**: Research and	**Jessica**: Design and	**Michelle**:	**Tad**: Write

Feature-Complete Release: Stretch Feature: Push Notification Services 5/22 - due	implement how to use Amazon SNS services for push notifications. **Josue**: Setup necessary settings on AWS to enable push notifications	implement the notification received in the notification bar **Thao**: Design the color scheme and app icon for the app	Implement the necessary data abstractions to facilitate push notifications **Ian**: Implement any architectural design changes in order to promote modularity	integration tests for push notifications **Kaleo**: Write a system test that simulates the usage of a user during a session
5/25 - Recovery week: Fix any issues, catch up				
6/1 - Prepare for Presentation	**Alice**: Presentation Prep **Josue**: Presentation Prep	**Jessica**: Presentation Prep **Thao**: Presentation Prep	**Michelle**: Presentation Prep **Ian**: Presentation Prep	**Tad**: Presentation Prep **Kaleo**: Presentation Prep

Team Structure

Josue will be the project manager because he is the only one with Android development experience. The roles of all members are to work on their assigned weekly tasks in pairs and collectively contribute to the group assignments. The weekly tasks are divided into four teams:

- Front End
    - Members: Jessica and Thao
    - Responsibility: Designing and implementing user interfaces and interaction activities.
- Back End
    - Members: Alice and Josue
    - Responsibility: Setting up, managing, and connecting to the AWS Cognito service and RDS database service.
- Integration
    - Members: Michelle and Ian
    - Responsibility: Providing data abstractions needed to promote modularity and facilitating communication between the front end and back end.
- Testing
    - Members: Tad and Kaleo
    - Responsibility: Writing black box tests to ensure proper documentation, integration/UI tests to ensure functionality, and system tests to ensure system behavior.

Group meetings will be held every Tuesday 9:30~10:30 and Friday 3:30~4:30 to coordinate efforts and resolve issues. The main outline of the weekly status reports will be discussed on Tuesday meetings, where one member of the group will be delegated with the responsibility of compiling the report. The final report will be uploaded into the GitLab project repository for final inspection, before being turned in. Our main method of communication outside of the weekly meetings will be through the Telegram app, Google docs, and Gitlab, which allow group discussions, issue tracking, and status reports.

Test Plan

We will be using unit testing to test each class as it's written. The authors of each class will write white-box tests for that class. In addition, the testing team will be writing weekly black-box unit tests for the classes to ensure that they follow the specs. All of our tests will be run with continuous integration to detect integration errors as they appear. We will be using Cruise Control to run builds that run as members and parts of the system.

To test the system and its integration the testing group will come up with black box tests involving multiple classes as they would interact in our system. For instance, we will need to test that network failures provide a dialogue with a helpful error message. We will also need to test that general use cases work as expected, such as login/logout and creating/ deleting/joining meal plans. Some of these tests will need to be simulated programmatically, such as network failures. The system will also be tested by using the MeetAGenius application on a Genymotion emulator on screens of varying sizes as well as physical mobile devices. This makes sure that the UI is integrated correctly and all buttons and interactions seem to do what they are supposed to. The system testing will be done every week during meetings and written by the testing team between meetings.

To test how intuitive our interface is and the simplicity of our program we will ask people to try out our UI and see what they say. Since the team has extensive knowledge of the system it would help to have someone else chime in on the ease-of-use of our application. These interactions with potential users would need to happen at least once after we come up with a complete GUI.

To be sure that our testing is adequate, we will enforce strict testing protocols for each team to minimize bugs and ensure system reliability, such as testing before pushing and adding regression tests. Teams will be required to write detailed JavaDoc comments before pushes to ensure ease of testing. Finally, we will use GitLab's issue tracker to track bugs and delegate fixing responsibilities.

Documentation Plan

When users set up new accounts, we will provide a user's guide integrated in the user interface, demonstrating how to use the app. As for external documents, we will write out an online user manual consisting of two sections. The beginner section will include basic functionality while the advanced section will include descriptions of additional features. Moreover, we could also create a promotional video to serve as a user guide and an advertisement if there is remaining time.

Coding Style Guidelines

Since this app will be developed for the Android platform, we have decided to use the "Android's Code Style for Contributors" located at URL: http://source.android.com/source/code-style.html as the coding style guideline for Java. In order to enforce these guidelines, members will be required to read the guideline and adhere by them. Each week we will have everyone briefly look at each other's code to ensure coding style is consistent with the guidelines. Before turning in the project, there will be a group meeting to review the entire project for proper coding style.