M2: Design Specification

Study Buddy

Nima Nasiri, 36969475
Joe Puthenkulam, 49190135
Yuyi Wang, 60661196
Yiyi Yan, 40800161

CPEN 321, 2019W
October 6th, 2019

PART 1: Status Update

Progress made since M1

- Front-end: Started integrating Google Calendar API into app framework. Implementing chat room and push notifications using FCM, but found socketIO would be a better choice.
- Back-end: Find the API to get student calendars from Google Calendar and UBC CWL. Implemented the chatroom in socketIO, tried to set up the database with both MongoDB and MySQL, decided to use MySQL. Set up AWS EC2 with free option.

Plans until M3

- Complete the database setting in backend, and more tests on the chatroom.
- Complete UI designs in storyboard and implement major components on mobile.
- Finish the calendar importing feature, and meeting event creation in backend.

Decisions and Changes in Project Scope

- The non-functional requirement: “Friends list changes updated after refreshing page (leaving and coming back to it)” should be functional requirement since it’s visible.
- Remove the non-functional requirement “Push Notifications received within 30 seconds of starting a free block “. From the meeting with TA, we should aim to get the notifications instantly.
- Functional requirement: “Location update”, changing from calling Google Map API to use the location service on the mobile.

Individual Contribution

- Joe: Currently integrating Google Calendar API for retrieval of Calendar/Events and Facebook authorization for retrieval of friends list.

- Nima: Finishing research for backend frameworks, architecture, and design.
- Yuyi: Implementing chat room and push notifications using FCM, front-end design for M2.
- Yiyi: implementing chat room with socketIO. Configure the database and completed the diagram components and backend design for M2.

PART 2: Design

1) ***List the main modules of your front-end and back-end components. Explain in one to two sentences the purpose of each module. [10 points]***

Front-end:

- **User setting view** : Allow user to view the current settings of their account.
- **Calendar view:** Allow user to view and edit their schedule, and find available friends in the selected time slot.
- **Friends and group view**: Allow user to view and add/remove friends and groups
- **Message view**: Allow user to chat with friends and groups.
- **Map view**: Allow user to view their current location and friends' locations.
- **Notification pusher**: Push notification for new friends suggestions, meeting time and locations, and messages received.
- **Push notification controller:** Receive notification from the back-end, push it to the notification pusher.
- **Request controller**: Receive requests from the view modules, communicate with modules in the back-end and return the result to the request module.

Back-end:

- **UserController**: Management unit for the user database. Provides logic for user login, user setting and calendar settings.

- **CalendarController**: Logic for the personal and group's calendar creation, maintenance. It also includes the logic for events in the calendars,
- **ChatRoomController:** SocketIO logic for chatroom, and also the main module that will manage the groupDB. It includes the chat bot that will suggest events and make the meeting planning experience more friendly.
- **SuggestionController:** The controller module where the complex logic belongs. It contains the suggestions features that will pull the whole user and calendar database to give user suggested friends and groups.
- **userDB**: Include user information, such as user login name, passwords, API tokens, user calendars, user preferences setting for notifications.
- **groupDB**: Databases for all the groups. It includes each group's userID, the group calendars and group's notification settings.
- **calendarDB**: Database for all calendars. Each calendar has all the events, which has the time, location, content and repeat options.
- **chatDB**: Database for all the chatroom information. Each chat has the groupID along with all the messages in the chat. Each message has the userID of the sender, the timestamp of the message and the location of the user when it was sent.

2) ***Define interfaces of each component, including parameters and return values. Explain in one to two sentences the purpose of each interface. [20 points]***

Front-End:

User Setting Views:

- NotificationState interface : Contains a local variable to indicate if the notification is turned on/off and functions to check/change the states of the notification setting.
    - Attributes:
        - Int state;
        - Int notification;
    - Operations:
        - Boolean checkstate(User user);
        - Void changeState(User user, Int state);

- User interface : Object representation of a user, contains calendar, account info, and friend and group list.
    - Attributes:
        - String id
        - String userName
        - String password
        - Image profilePhoto
        - Calendar calendar
        - FGlist list
        - NotificationState state
    - Operations:
        - Boolean changeUserName(String newName);
        - Boolean changePassword(String oldPassword, String newPassword);
        - Boolean changeProfileImg(Image newPic);
        - Calendar getCalendar();
        - FGlist getFGlist();

Calendar view:

- Calendar interface: Contains list of events in a calendar, can add/remove/edit event in a calendar.
    - Attributes:
        - Event[] eventList;
    - Operations:
        - Event[] getEvents(String date)
        - Boolean addEvent(String eventTitle, String eventLoaction, String startTime, Int duration);
        - Boolean removeEvent(Event event, String date);
        - Boolean changeStartingTime(Event event, String startTime);
        - Boolean changeEventDuration(Event event, Int duration);
        - Boolean changeEventLocation(Event event, String location)
- Event interface: event object that contains the event name, location, starting time, and the duration.
    - Attributes:
        - String eventTitle
        - String eventLocation
        - Sting eventTime
        - Int duration
    - Operations:
        - String getStartingTime(String eventTitle);
        - String getEndingTime(String eventTitle);
        - String getLocation(String evenTitle);

Friends and Groups view:

- FGlist interface: Keeps track of friends associated with the user. Mainly used in creating groups and setting one-on-one meeting.
    - Attributes:
        - Group[] groupList;

        - Friend[] friendList;
    - Operations:
        - Void deleteFriend(Friend friend);
        - Void addFriend(User newFriend);
        - Void deleteGroup(Group group);
- Group interface : Keeps track of all users that are part of a group.
    - Attributes:
        - Friend[] groupMembers;
        - String groupName;
    - Operations:
        - Friend[] getGroupMembers();
        - Void startGroupChat();
        - Event[] getFreeBlocks();
- Friend interface: User abstraction defining association to another user.
    - Attributes: User friend;
    - Operations: Void startChat();

Message view:

- Messaging Interface: Receives input text and sends to a specific user or group.
    - Attributes:
        - String buffer;
        - User recipient;
        - Group Recipi
    - Operations:
        - String waitForInput();
        - Int sendMessage();
        - Void sendError();

Map view:

- Maps Interface: Shows recent location of a certain amount of friends.
    - Attributes:
        - List friends[];
        - List locationsOfFriends[];
    - Operations:
        - Void showFriends(friends[], locationOfFriends[]);
- Notification pusher:
    - MessageCenter Interface: Receives messages from the notification controller
        - Attributes:
            - String[] messages;
        - Operations:
            - Boolean pushNotification(User[] target, String message);
            - Void newMessage(User target, String newMessage);

Push notification controller:

- NotificationController Interface: Receives messages and notifications from the back-end, manage the notifications and decide which to send to the messageCenter
    - Attributes:
        - Queue<String> notifications

- Operations:
    - Boolean receiveNotification(String newNotification);
    - Void manageNotifications();
    - Boolean sendNotifictionInfo(String message, User target);

Request controller:

- RequestController Interfacer: Handles the requests sent from the views by indicating the request type and calling the corresponding controller in the backend.
    - Attributes: N/A
    - Operations:
        - void execute(String requestContext);

Back-End:

User Controller:

- User Controller Interface: Handles adding/removing friends, creating/deleting accounts, user settings, and user authentication.
    - Attributes:
        - String UserID
    - Operations:
        - bool addFriend(string idUser, string idFriend);
        - bool removeFriend(string idUser, string idFriend);
        - bool createAccount(string id, string userName, string password, Image profilePhoto, Calendar calendar);
        - bool deleteAccount(string id);
        - bool userLogin(string username, string password);
        - bool userLogout(string username);
        - bool notificationSettings(bool notificationsOn);

Calendar Controller:

- Calendar Controller Interface: For storing calendars and authenticating the google calendar if it's being used:
    - Attributes:
        - string CalendarID
    - Operations:
        - bool getCalendarInfo(Calendar calendar);
        - bool shareCalendarInfo(Calendar calendar);
        - void storeCalendar(Calendar calendar);

Chatroom Controller Interface:

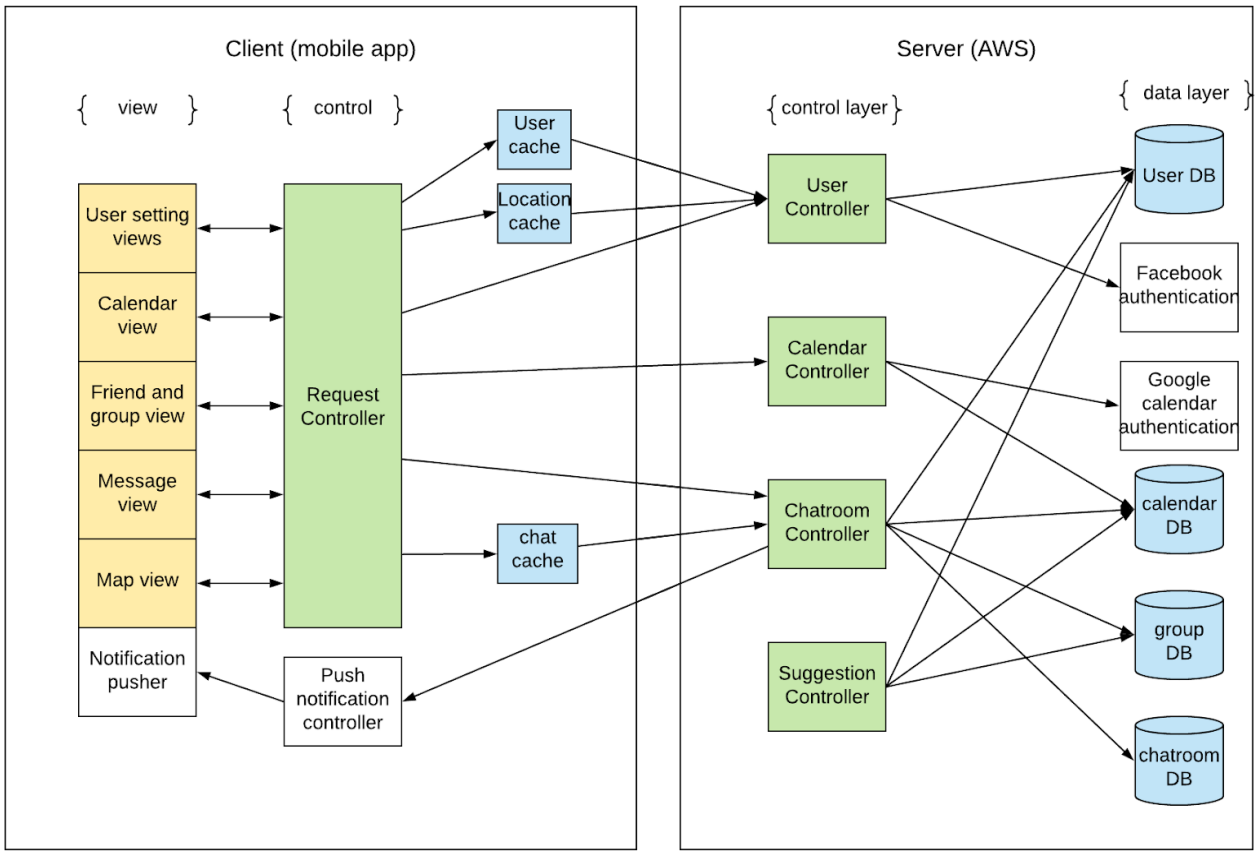
- Chatroom Controller Interface: Handles chat notifications and chat room messages/history.
    - Attributes:
        - String ChatID
    - Operations:
        - void notifyUser(string userID, int numChatsActive);

- void updateChatHistory(string userID, string message, int time, double userLocationLong, double userLocationLat);
- struct updateChat();

Suggestion Controller:

- Suggestion Controller Interface: Handles friend suggestions when user is looking to add friends.
  - Attributes:
    - List friends[]
    - List groups[]
  - Operations:
    - List[] suggestFriends();
    - List[] suggestGroup();

3) *Draw a diagram showing dependencies between the modules. The "boxes" in this diagram should correspond to the identified modules; the links should show calls between the modules and should have directionality. [15 points]*



4) Specify which architectural patterns are used for the front-end and back-end components and explain in one to two paragraphs why. [10 points]

Front-end: Model-View-Controller

With integrating so many APIs in addition to our own functions coupling between modules increases quite drastically so to mitigate this issue we believe a MVC architectural pattern will adhere to the fan-in/fan-out principle best. We assume that the integrated APIs will have low coupling and high cohesion and we aim for our own modules to do the same.

Back-end: Client-Server

We decided on a client-server architectural pattern as it keeps the structure of the app rather simple, and helps in distinguishing what work should be done in the front and back end. Using basic logic in the front end can make some operations more efficient by having computation done on the user's device; an example of this would be for push notifications. Keeping most data transfer in the back end allows the front end developers more time to focus on making the app interface usable and presentable.

5) Specify the languages and frameworks used (React-Native, Xamarin, Express, etc.), explain in one to two paragraphs why you picked them and what the alternatives were. [10 points]

Front-end: Java, Android Studio, Facebook SDK, Maps SDK and Calendar SDK

Since we are developing an Android application and both of the front-end developers are new to the mobile app development process, we chose to use Android studio. There are three additional reasons for that decision:

- With all group members being iPhone users, the GUI tools in Android Studio such as Layout Editor, APK Analyzer and Vector Asset Studio streamline our development process.

- The Gradle build system in Android Studio is integrated well and has great support for our external API calls (Facebook Login, Google Calendar, Google Maps).
- Android Studio is the official IDE for the Android platform and provides a more stable performance when coding and testing. We will be using Java since it is the official language of Android development. Furthermore, Facebook SDK, Calendar SDK and Maps SDK for Android will be included in the application as the external APIs. Since Facebook, Google Calendar and Google Maps are popular apps, using these external APIs will allow us to access all the information needed for the back-end implementation.

The alternative we came up with was React-native for the UI design which can be used to develop applications for Android, iOS, Web and UWP. Since we are only going to implement an Android app we think Android Studio will give us more targeted libraries and tutorials.

Back-end: **Javascript, NodeJS, Express, AWS EC2 for hosting**

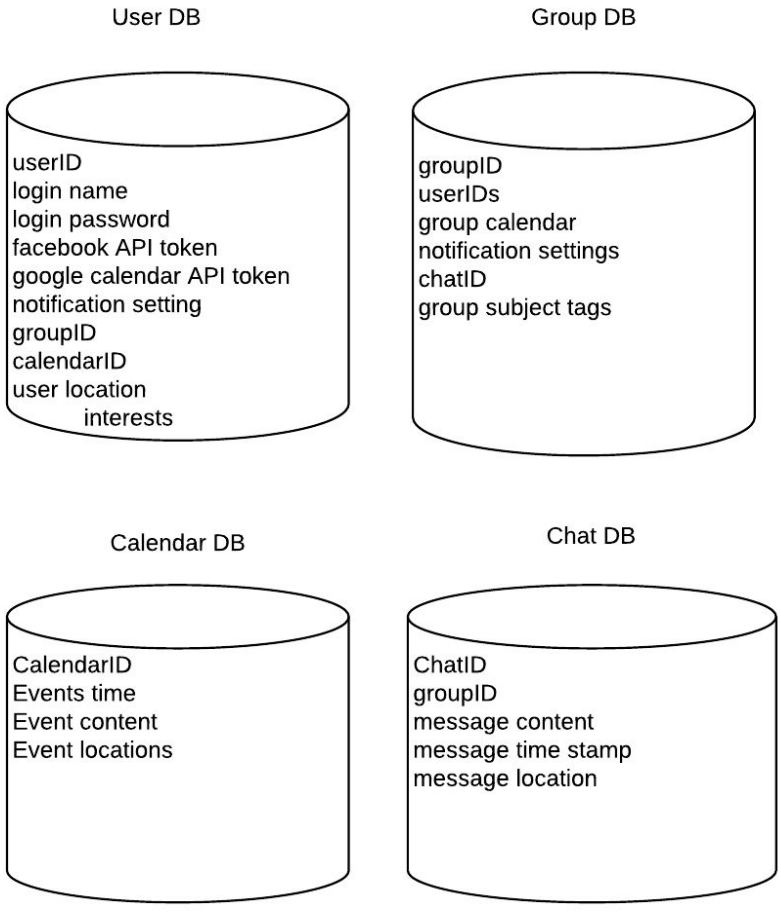
The framework that we chose for backend are NodeJS (javascript) with Express, which simplified the workload of setting up a server from scratch. We didn't have a choice for the backend framework, since the project requirement stated NodeJS, we used Express to set up a prototype, but might consider other frameworks if the performance doesn't meet the non-functional requirements. We would have considered Django for it's flat learning curve and python, but it's not NodeJS.

For hosting the backend, we currently are using AWS EC2 since some of our members have previous experiences with this platform. Other alternatives include Microsoft azure, digital ocean, but it doesn't make much difference. Lastly, since we are using Docker, the remote server platform no longer matters.

6) ***Specify the databases used (MongoDB, MySQL) and the schema of the data you store. Explain in one to two paragraphs why you made that choice and what the alternatives were. [10 points]***

We decided to use MySQL due to its structure and the fact that some team members have prior experience using it. MongoDB is easier to change as it is less rigid, given that all information is stored in documents; which allows the developers to choose how to structure their database. Although MongoDB is more fluid and apparently easier to scale, we chose MySQL for the better structure and security. Since there is prior MySQL experience in the team it also allows us to look into learning how to make it more scalable.

Data Schemas



7) ***Explain in one to two sentences how you plan to realize each non-functional requirement from M1. [10 points]***

- **Messages received within 500 milliseconds**

SocketIO is the framework we are planning to use to achieve real time messaging. There will be almost no delay when sending and receiving messages in the chatroom after the socket starts. The critical path is when we start the socket, which is well below 500ms based on our measurements.

- **Pop ups and page switches execute within 100 milliseconds**

Most of the pages doesn't have any logic or data polling. The only pages that will experience noticeable delays are: chatroom view page, suggestion page and map view page. We are planning to meet the requirements by adding cache on client, as well as use the native mobile location service instead of Google Map API.

Opening additional modules would probably require a context switch which increases latency. So we aim to reduce this by adhering to the fan-in/fan-out principle with our modules.

- **Filtering friends takes effect after typing each letter within 100 milliseconds**

We are planning to use a User cache on the client to meet this requirement. Since polling the friend list from remote server will definitely exceed 100ms, by caching all the friends and certain amount of user information along with it (such as chatID, calendarID and interests), it will meet this requirement and not increase the mobile app storage usage significantly.

8) ***Describe the main algorithm used in your "complex logic" use case. Specify its inputs, outputs, and main computational logic. [15 points]***

Friend Suggestion Algorithm: our complex logic use case is suggesting new friends by considering location, schedules, and mutual friends.

The input to this use case are: **User, Calendar and Group DB.**

The output to this use case are: **suggested friends list**

Main computational logic: this module will take the user's current friend list from User DB, and calendar from Calendar DB. Based on this information, we are going to use statistical analysis algorithms such as PCA to get the suggesting subjects and pull a list of users that are tagged with this subject, and then use feature matching algorithm such as Linear Regression. We will then recommend the users these suggested friends, which should have similar calendars, mutual friends, and location to the users.