Department of Computer Science & Engineering
The University of Texas at Arlington

Architectural Design Specification
CSE 4316: Senior Design I
Fall 2022

Passive Aggressive Solutions
Lateless

Matthew Mcnatt

COLBY WYRICK
NGHIA LAM
GIA DAO
TRIEU NGUYEN

Revision History

Revision	Date	Author(s)	Description
0.1	10.01.2015	CW, MM, NL, GD, TN	document creation
0.1	10.01.2015	MM	Polishing

Contents

List of Figures

List of Tables

1 Introduction

LateLess is a mobile application that allows users to set up events with colleagues or friends and track the location of those colleagues to discern whether they will be late to an event the user is also attending. LateLess Users will be able to see the location, distance, and drive time estimates of these other individuals also attending the same event.

LateLess will be able to rely on location tracking as well as traffic navigation systems to dynamically display the arrival status of individuals attending an event. It can be used either as a remedial solution to the issue of constantly late individuals or as a scheduling utility to minimize wait time in organized meetings. The intended audience of LateLess falls into two categories that are both viable: public, and professional. The first is individuals using the app as recreational software alongside friends and family in a non-constructive or relaxed environment. These individuals would download the LateLess application as a fun way of interacting with other people they know that also use LateLess. The second class of users are individuals that would download LateLess as an efficiency tool to cut down on the dead time generated from poorly communicated meeting times or vague standards for arrival. Though this does not refer specifically to a cooperate environment, a version of the application could also be easily expanded and customized specifically as a cooperate tool and marketed to this category of users as well. Overall, LateLess is intended for general use and the current product is tailored specifically for the public.

The first part of this solution is to be able to add friends (or people who want to meet) and be able to create events at a certain location and time that everyone in that group agrees to meet. Then, a certain amount of time before the event start, the app will begin to track the locations of the people in the event. The app should be able to ping to other people in the group where everyone is at and what distance they are from the event's location. This app should also be able to tell people in the group if somebody slows down sharply (whether because of an accident or large amounts of traffic) through a notification. Lastly, the application needs to be able to stop tracking once a person reaches a certain radius of the event's location. The major components of this app are the ability to track multiple people on a map, notifications based on location and time, a friends list, and event creation based on the location and time that the group wishes to meet.

2 System Overview

There are three overarching layers involved in the LateLess architecture: a database, a react map layer and a react profile layer. The database layer is used for user authentication, profile information storage and acting as a central hub for the location data flow. The React Native Map Functionality is the layer responsible for the rendering, displaying and instantiating events, as well as tracking and displaying the attendants of those events. Finally, the React Native profile functionality layer will handle the registration and authentication of users, as well as updating that information in the database. The system's design centers around the flow of coordinates and React's lack of distinction between a front and back end. All the user information distinct from event management is handled in the React Profile Functionality layer and stored in the Database layer. Meanwhile, The React Native Profile Functionality layer pulls the coordinates of all individuals in a particular event from the database to render it using the Mapbox API.

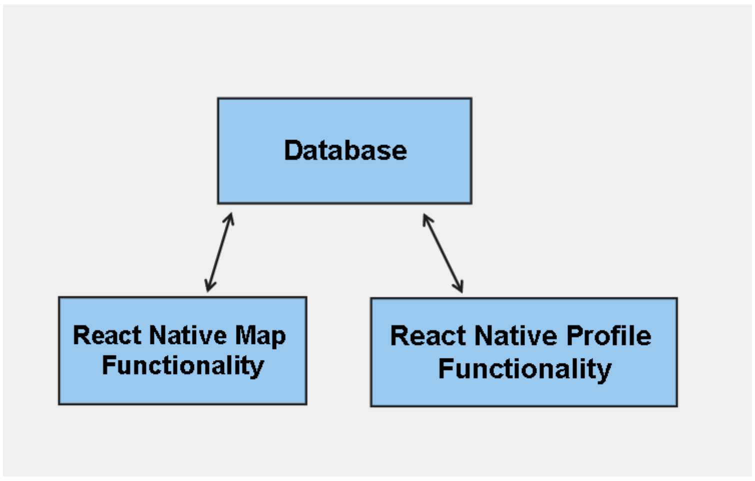
Database
React Native Map Functionality
React Native Profile Functionality

Figure 1: A simple architectural layer diagram

2.1 React Native Map Layer Functionality Description

This layer goes over the Mapbox API implementation in react native. First, this layer needs to allow the creation of an event at a certain location and a certain time. Also, the creator of the event needs to be able to add people to the event so that they can be tracked. Next, this layer should be able to track people's locations which will allow our project to give status updates of the event based on all attendee's locations, time left before the event starts, and other various notifications in regard to the event. The Mapbox API needs to also render a GUI map so that the user can easily create events. Gets coordinates from other user's profile information from the Database layer and will update information on the map frequently. Should use a notification-based system to give out information to users.

2.2 React Native Profile Layer Functionality Description

This layer deals with all information flow pertaining to users and not specific events. This layer handles user registration, user login, user profile update, friend requests and friend acceptance. It relies on the Database Layer to store this information updates as well as pulling all the users information from a database upon a successful login. It will also pull some user specific information from the Map Layer upon the conclusion of events.

2.3 Database Layer Description

The Database layer is purely responsible for storing information for the other two layers. It stores user information, user friend connections for the React Native Profile Layer, as well as coordinates of users and the drive time distance from a specific event for the React Native Map layer.

3 Subsystem Definitions & Data Flow

The image below shows the data flow between all our subsystems. Starting with 1, it shows the User Info CRUD pointing to profile information. The user info CRUD will create information like login info, and then push it to the database to pull to the GUI. Number 2 shows the friend information pointing at profile information because the friend info will be sent to the database. Number 3 has the event CRUD pointing at event information because the event CRUD will create and update information about the event. Number 4 shows the profile information pointing at login systems because the user info CRUD will create the login information and then login systems will pull the info from the database. Number 5 shows profile information pointing at attendant tracking because the coordinates for each attendee will be sent to the database through user info CRUD and pulled from the database. Number 6 shows the arrows pointing both ways between attendant tracking and event status. This is because attendant tracking will see how far each attendee is from the event location, send that information to event status, and event status will send a notification based on the attendant's location. Both 7 and 8 show attendant tracking and event status pointing at mapbox map because both will use the map to get coordinates and get the status of the event. Number 9 shows profile information pointing to event CRUD because event CRUD will need things like friend information and the creators own name. Lastly, number 10 shows event information pointing to event status. This is because event status needs the event information.

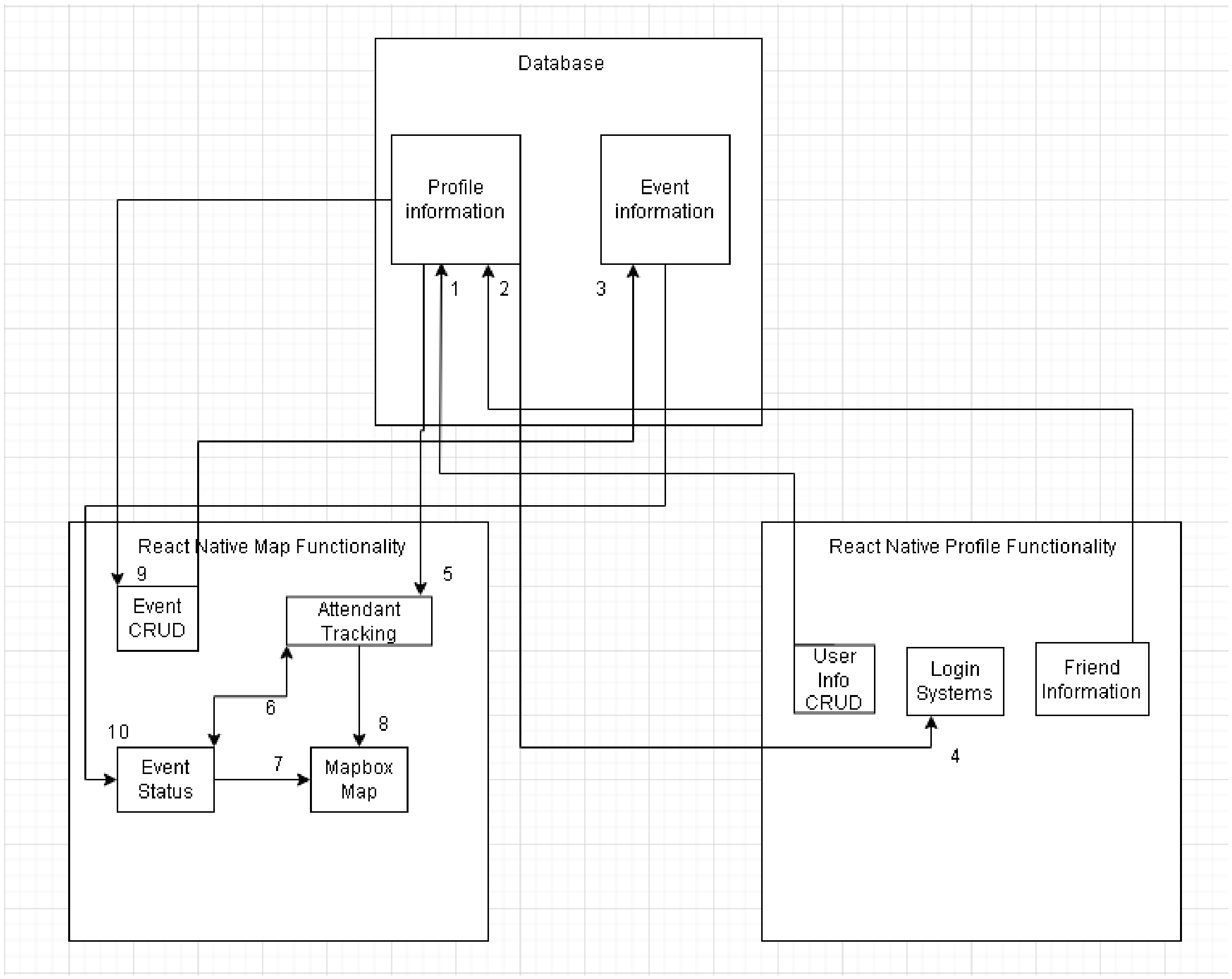

Figure 2: Data flow diagram

4 React Native Map Functionality Layer Subsystems

In this section of the layer it will be describing the React Native Map Functionality Subsystems. This includes Event CRUD, Attendants tracking, Event Status, and MapBox Map

4.1 Event CRUD

In the Event CRUD, users will be able to create, read, update, and delete and events/group.

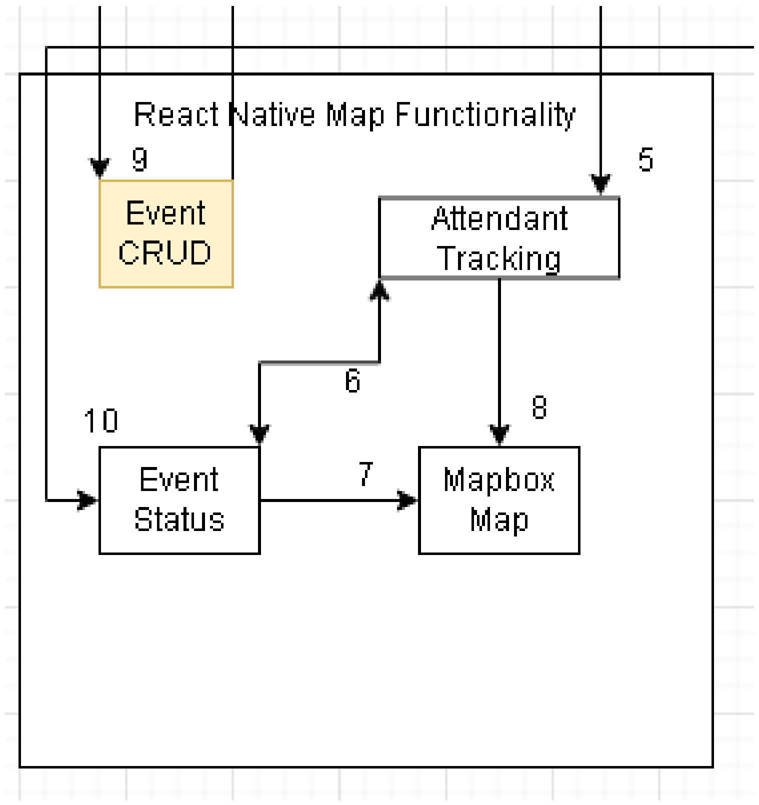

Figure 3: Example subsystem description diagram

4.1.1 Assumptions

The user should have an account and be logged into that account.

4.1.2 Responsibilities

The user would be able to create and read an event. They would also be able to update any information in the event as long as the event has not started yet. The user will also be able to delete any event in order to cancel it. They will only be allowed to update/delete an event only if they are the owner or if they are given permission.

4.1.3 Subsystem Interfaces

Each of the inputs and outputs for the subsystem are defined here. React will be responsible for gathering the user information and input, but all changes will be stored in the database.

Table 2: Subsystem interfaces

ID	Description	Inputs	Outputs
1	Create allows the user to be able to create an event group where they will be able to add people to it.	Name/Description, Members, Date/Time, Location	Event Group
2	Read allows the users to view the event	Gets all the data from database that the user enter when creating the event	Displays: Name/Description, Group Members, Date/Time of the event, Location
3	Update allows the user to update/change the event group as the please, as long as they are the owner/admin of the group.	Name/Description, Members, Date/Time, Location	Name/Description, Members, Date/Time, Location
4	Delete will delete an event, and can only be done by the owner of the group	Press the Delete Button	Delete Group

4.2 Attendants tracking

In the attendants tracking subsystem, it will track the attendants of all the users that are in the group.

4.2.1 Assumptions

The user should have an account and be logged into that account. The user should also be connected to the internet. The user should be in an event group already. And attendants should be tracked automatically.

4.2.2 Responsibilities

Once an event has started, the attendants of everything in that event group will be tracked automatically. And once the event has ended, it will stop tracking them. The data of their attendants will also be stored within the database in profile information. This will allow all other people to see the attendant records others in a group.

4.2.3 Subsystem Interfaces

Attendant tracking is responsible for managing the state of the event relative to user location. Useful information is then computed once and pushed into the database layer to avoid unnecessary API calls.

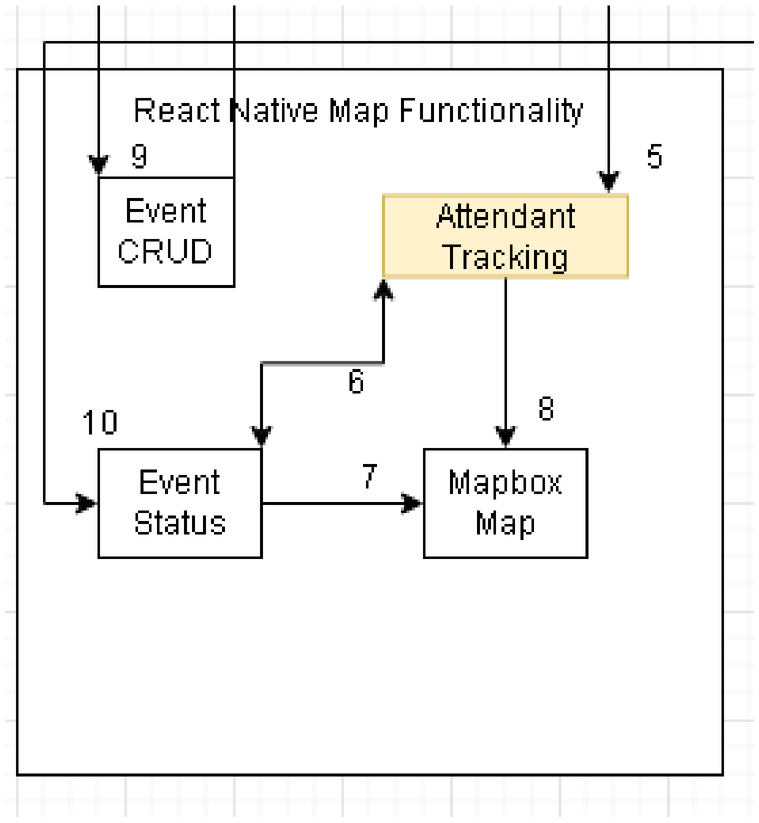

Figure 4: Example subsystem description diagram

Table 3: Subsystem interfaces

ID	Description	Inputs	Outputs
1	Keep track of everyone that has arrived at the event.	Group Member	Arrived, OnTheWay, Late, NoShow

4.3 Event Status

In the event status subsystem, it will keep track of the event status information. Like when the event starts, if it already started, or if the event is over.

4.3.1 Assumptions

The event group is already created. It should have people/users in the event group and it should keep track of the event status automatically.

4.3.2 Responsibilities

Once an event is created, it will start tracking the status of the event. It will then send that information into the database where it will then be stored in the event information section. There are 3 main status to the event: start, currently on going, and ended.

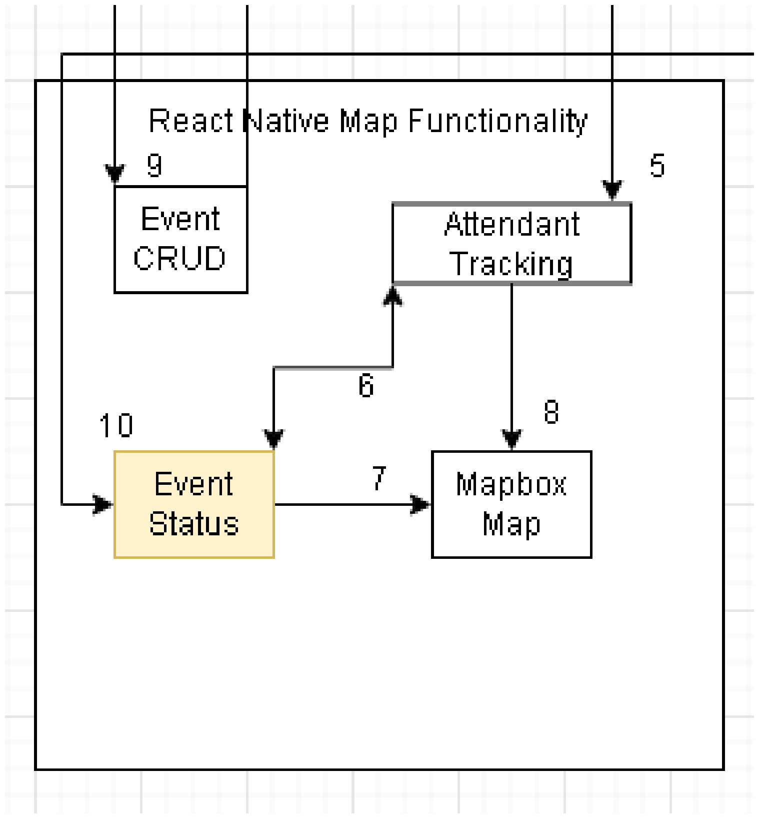

Figure 5: Example subsystem description diagram

4.3.3 Subsystem Interfaces

The current status of an event is incoming remote information unless the user is the owner of the group. This information is pulled to then determine the current running state of the application.

Table 4: Subsystem interfaces

ID	Description	Inputs	Outputs
1	Keeps track of the event status	Get the Event Status from the database	Start, onGoing, Ended

4.4 MapBox Map

In the MapBox Map subsystem will be a map where users will be able to use for navigation, tracking, and etc.

4.4.1 Assumptions

The user already has an account and is logged in. The user much also be connected to the internet.

4.4.2 Responsibilities

This subsystem will just be a displayed map around the user. Where they will be able to interact with it to track others or search for meet places near them.

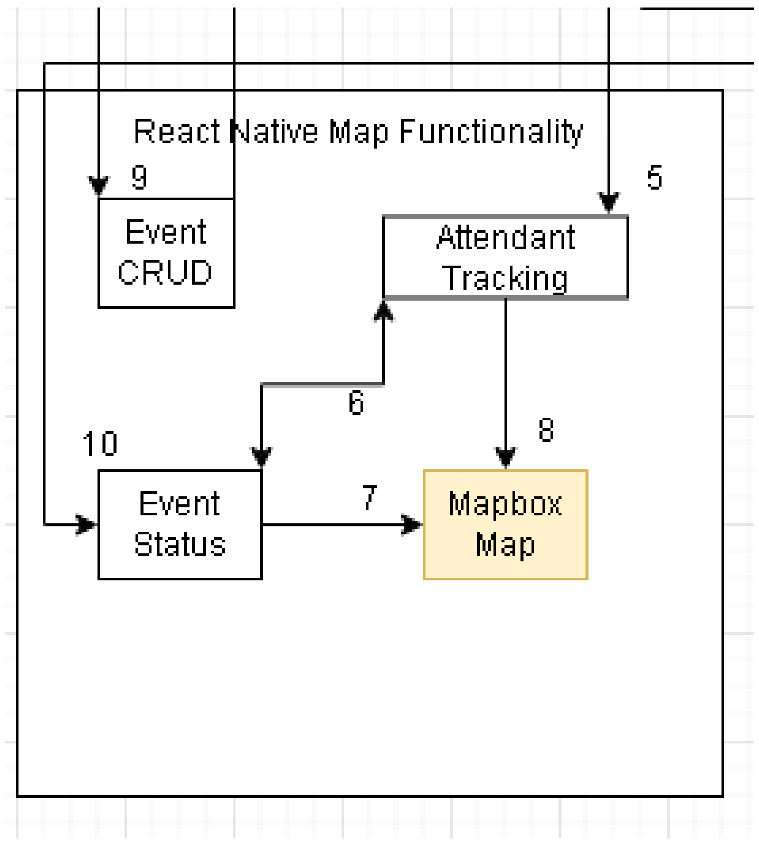

Figure 6: Example subsystem description diagram

4.4.3 Subsystem Interfaces

Mapbox has its own systems for interaction with the Mapbox map. This is just a matter of implementing it into the UI.

Table 5: Subsystem interfaces

ID	Description	Inputs	Outputs
1	This will display a map that the user can use and interact with.	Map from MapBox API	Displays a map

5 React Native Profile Functionality Layer Subsystems

This section describes the Profile Functionality of the application. This functionality includes two core subsystems, Login System and User Create/Read/Update/Delete (User CRUD) that are used to monitor user's account.

5.1 Login System

If the Login System verifies the user account is valid, it will allow users to access the application; otherwise, 'Invalid Account' message will be generated and the user will be blocked from access.

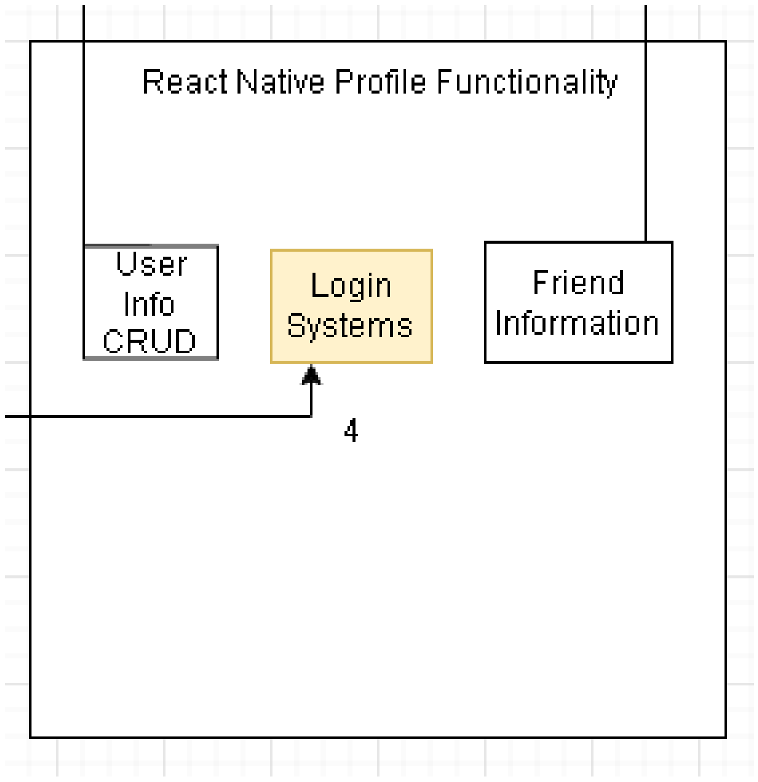

Figure 7: Example subsystem description diagram

5.1.1 Assumptions

Login System assumes that the user has already included the account to login and Login System have two inputs: username and password

5.1.2 Responsibilities

The Login System must be connected with the database. Specifically, the Login System interacts with the Profile Information subsystem. Once the user types in username and password, the Login System will check if that user data is already stored in the Profile Information subsystem. If the retrieved information is matched, the user is authorized to use the application.

5.1.3 Login System Interfaces

Table 6: Subsystem interfaces

ID	Description	Inputs	Outputs
1	Input for username typing	Input from user (username)	Access or Denial
2	Input for password typing	Input from user (password)	Access or Denial
3	A login button	On press	Success or Fail Login

5.2 User Info CRUD

User Info CRUD functionality helps user to create/read/update/delete profile. In case the user has not created an account, the application automatically prompts the user to create one. User Info CRUD also helps user to update their account, such as email, phone numbers, address. If the user requests to change username or password, the User Info CRUD will send a reset password email and verify the identity of the user.

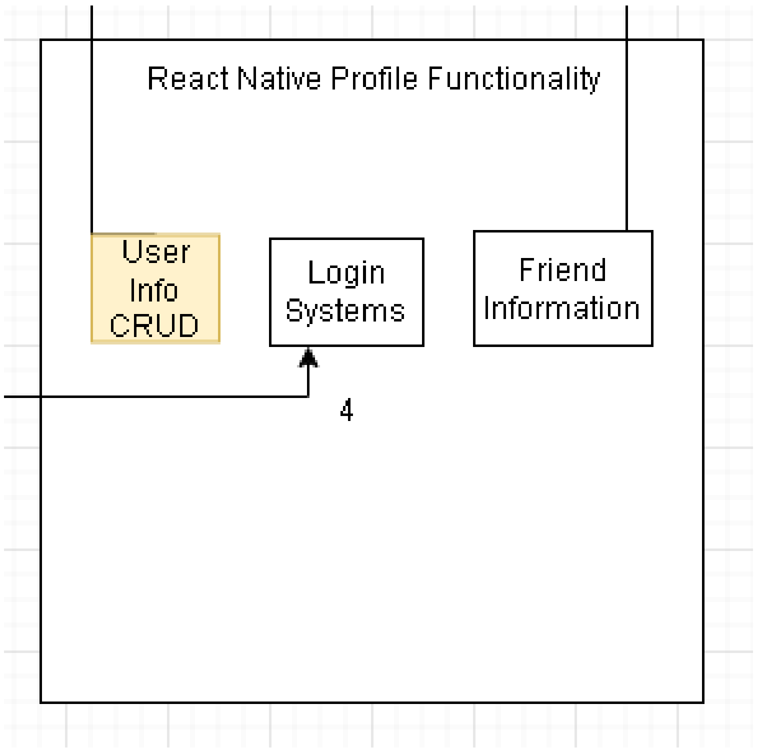

Figure 8: Example subsystem description diagram

5.2.1 Assumptions

The user must enter a valid email when registering for a new account and the entry for password requires 8 characters in length, including numbers, special characters, lowercase and uppercase letters for security purpose

5.2.2 Responsibilities

The User Info CRUD subsystem must be connected with the database. Once the user creates, updates, or deletes the account using User Info CRUD functionality, the Profile Information functionality must also act correspondingly, which is constantly checking for any changes of information from the user so that the Login System will have sufficient data from the data base to check for user's inputs.

5.2.3 User Info CRUD Interfaces

Table 7: Subsystem interfaces

ID	Description	Inputs	Outputs
1	First name	Input from user	Accepted
2	Last name	Input from user	Accepted
3	Email	Input from user	Success or Fail
4	Password	Input from user	Success or Fail
5	Password Confirmation	Input from user	True or False
6	Create button	On press	Allow user to register for account
7	Update button	On press	Allow user to update profile
8	View Profile button	On press	Allow user to view information
9	Delete account button	On press	Ask user if he/she wants to delete account, if press 'YES' then delete

5.3 Friend information

Allows users to add people via a friend list and will send this information up to the database for storage in the profile information subsystem.

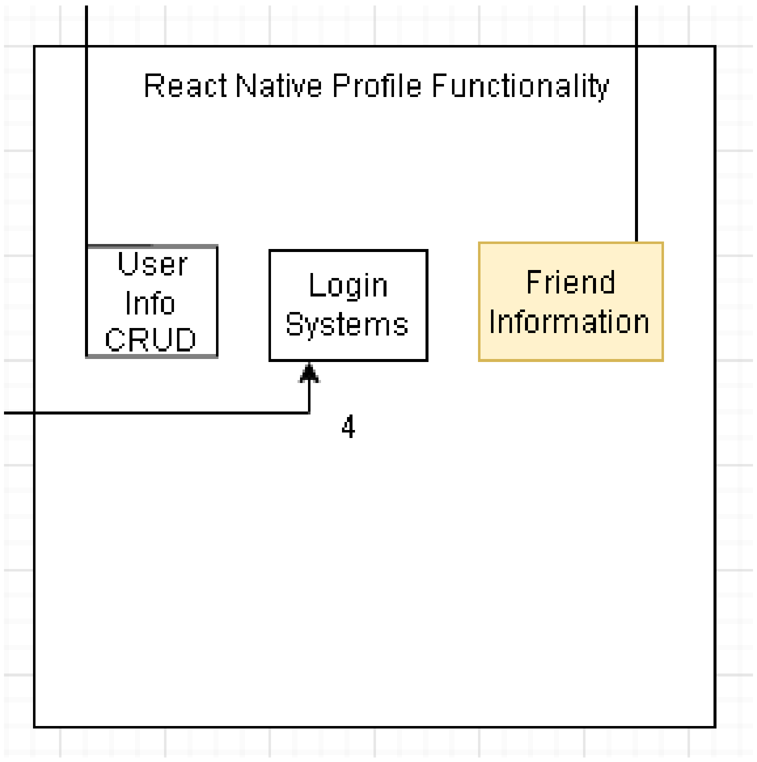

Figure 9: Example subsystem description diagram

5.3.1 Assumptions

Assumes the person being friended is a real person.

5.3.2 Responsibilities

The friend information subsystem must be able to send information to the database, or specifically the profile information subsystem. Users should be able to add or remove friends and that needs to be updated in the database as well. The user should also be able to see the friends that they have already added.

5.3.3 Friend Information

Table 8: Subsystem interfaces

ID	Description	Inputs	Outputs
1	Text Input for username of friend	Input from user	Success or fail
2	View friend list	Button press	Show friend list
3	Remove friend	On press	Remove Friend

6 Database Layer Subsystems

This is the database system. In this system, we will have the subsystem Profile information and the event information.

6.1 Profile Information

The profile database will store the user's personal information like age, name, Email, phone number, friends, and GPS coordinates.

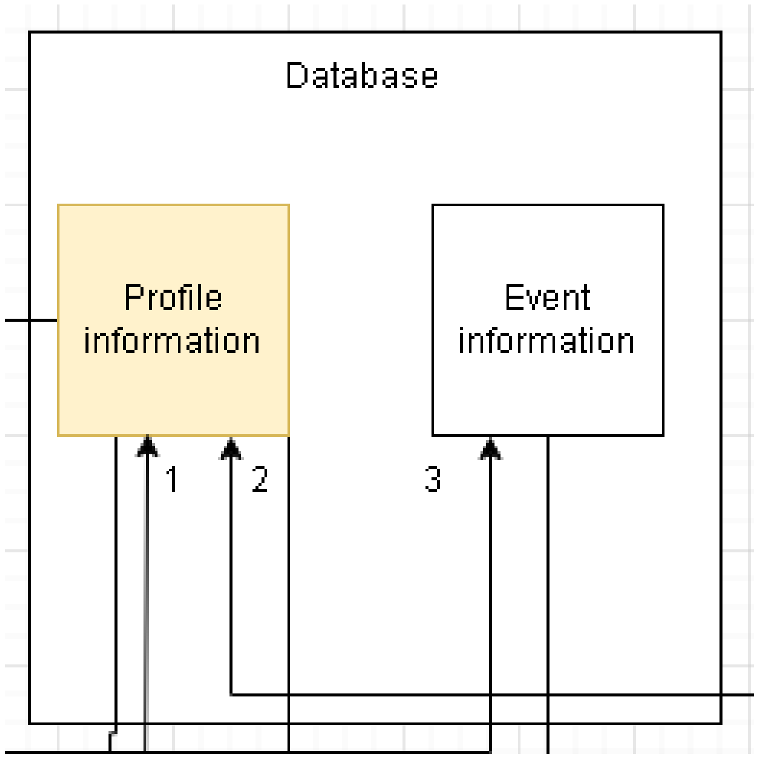

Figure 10: Example subsystem description diagram

6.1.1 Assumptions

Assuming the user has an account and they are also logged in.

6.1.2 Responsibilities

The "Profile Information" responsibility is to store the information of the user so that it can push it into the "React Native Map" layer.

6.1.3 Subsystem Interfaces

This subsystem will communicate with the "Event CRUD" and "Attendant Tracking" as it will push the GPS coordinates of the user into "Attendant Tracking" and it will push the name as well into the "Event CRUD". It will get its information from the "User info CRUD" and the "Friend info" subsystem where it will then be stored in this subsystem.

Table 9: Subsystem interfaces

ID	Description	Inputs	Outputs
#1	This will hold the data of the user like GPS coordinates and personal information	User CRUD	Successfully store user information into database

6.2 Event Information

The profile database will store the event's information like the attendants, time the event starts, and location of the event.

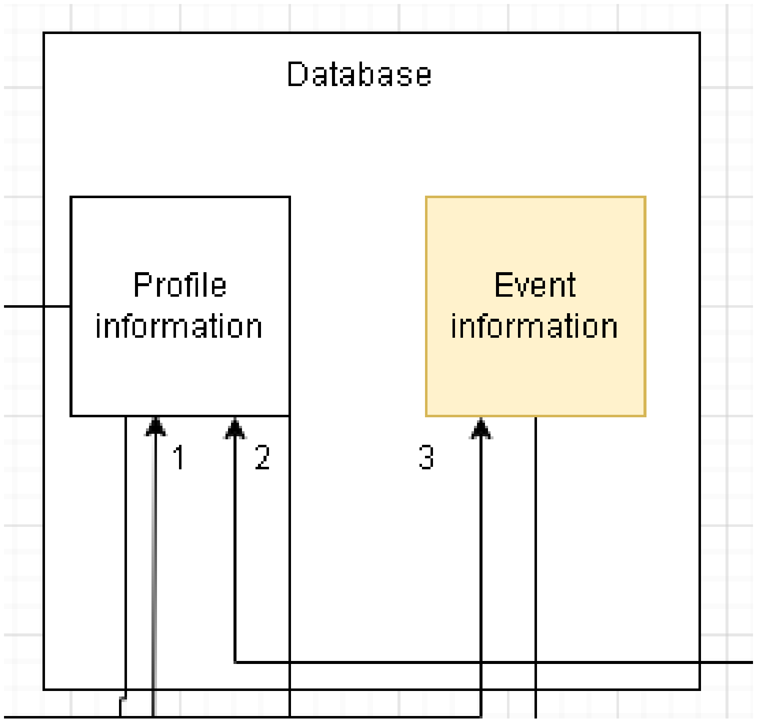

Figure 11: Example subsystem description diagram

6.2.1 Assumptions

Assuming the user has an account and they are also logged in. There is an event in progress.

6.2.2 Responsibilities

The "Event Information" responsibility is to store the information of the event so that it can push it into the "React Native Map" layer.

6.2.3 Subsystem Interfaces

Table 10: Subsystem interfaces

ID	Description	Inputs	Outputs
#1	This will hold the information of the event	Event CRUD	Successfully stores event on database

References