LIFETRACKER

WILD KARRDE DEVELOPMENT

Martin Fritz

Jacob Roquemore

Huy Le

Jonathan Wallis

SOFTWARE DESIGN DOCUMENT

Version 1.0

INDEX

TITLE Page No.

1. Introduction 1
2. Overview of Architecture 2
3. Use cases 3
   - A. Use Case Diagram
   - B. Use Case Specifications
     - i. LTUC1 – View Profile
     - ii. LTUC2 – View Reminders
     - iii. LTUC3 – View Missed Tasks
     - iv. LTUC4 – Edit Tasks
     - v. LTUC5 – Login
     - vi. LTUC6 – Add Task
4. Classes and their interactions 4
   - A. Class diagram
   - B. Details of classes
     - i. Cl – Class name
     - ii. C2 – Class name
     - iii. ….
   - C. Sequence diagrams
     - i. S1 – Scenario 1
     - ii. S2 – Scenario 2
     - iii. …
5. State transition diagram 5
6. User interface design 6
   - A. Sample screen 1 – Screen name
   - B. Sample screen 2 – Screen name
   - C. Sample screen 3 – Screen name
   - D. …
7. Conclusion 7

References

Appendices (Optional)

Glossary (Optional)

1. Introduction

This document covers the development, testing, and the operation of the Android application known as LifeTracker.

LifeTracker is an Android Application that allows its users to develop healthy habits by tracking activities and events within their life. Using analytics, the user is able to see where they have succeeded and fallen short of their goals. By tracking these, the user will begin to develop lifelong habits that improve their individual success.

The user will have the ability to create different types of events within the application: They will have the ability to create daily tasks that require completion before the end of the day, the ability to create "one-off" events that are similar to calendar events in many modern calendar applications found on all operating systems, and have the ability to create custom events that allow them to specify a user defined amount of time before the event is triggered.

This application will utilize a mySQL database and Google APIs as supplementation. Events will be stored within a database that defines a key that is unique to the user logging into the database. This will store their tasks and recall them when requested. The Google Maps API will allow users to plan and plot their location for geolocation specific events. Users will also have the option to export their events to their individual Google Calendars using the Google Calendar API.

The design of this application, and any application in general, allows its authors to envision and execute on an application that is aimed at aiding individuals overcome problems encountered. In the case of LifeTracker, we aim to aid consumers with overcoming the hurdles of procrastination and the hopeless feeling when faced with seemingly insurmountable odds and deadlines. This software is designed to be driven solely by the consumer making it a user centered software. Understanding this, Wild Karrde Development developed specific requirements that aim to minimize the intellectual distance the consumer requires to use the application and create a user experience that is intuitive.

The following document outlines Wild Karrde Development's LifeTracker application. Subsequent is the architecture used for the application, selected use case diagrams along with their corresponding specifications, the class diagrams of the application along with their related sequence diagrams and class explanations, the state transition diagram of the application, and finally mockups of the anticipated final project.

2. Overview of architecture:

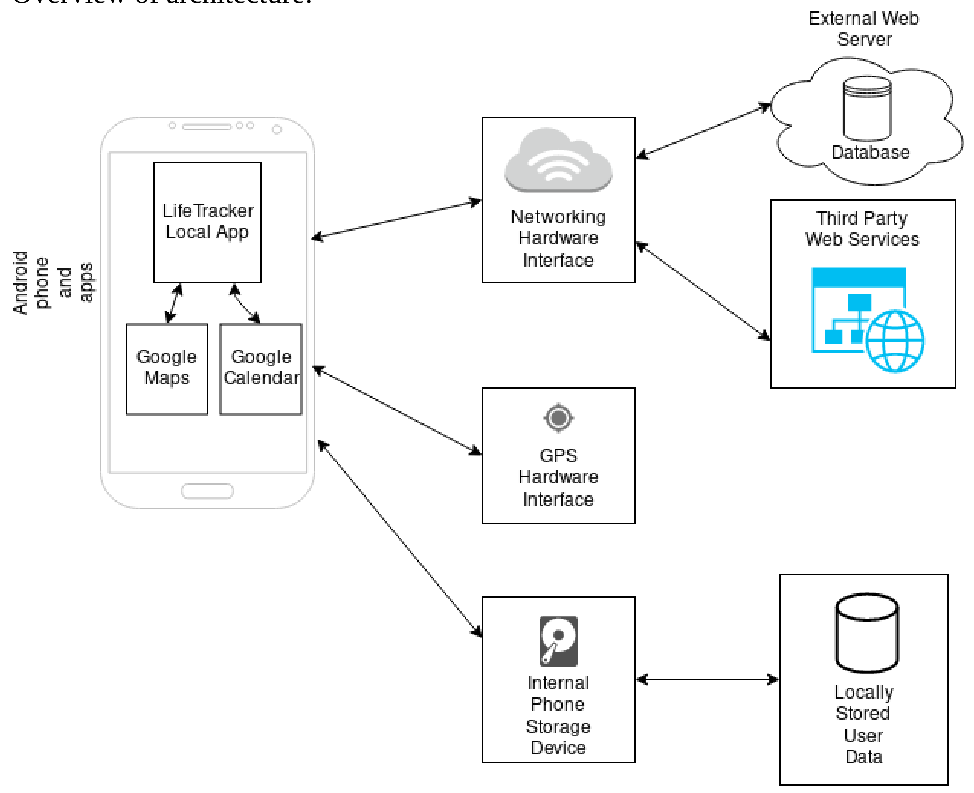
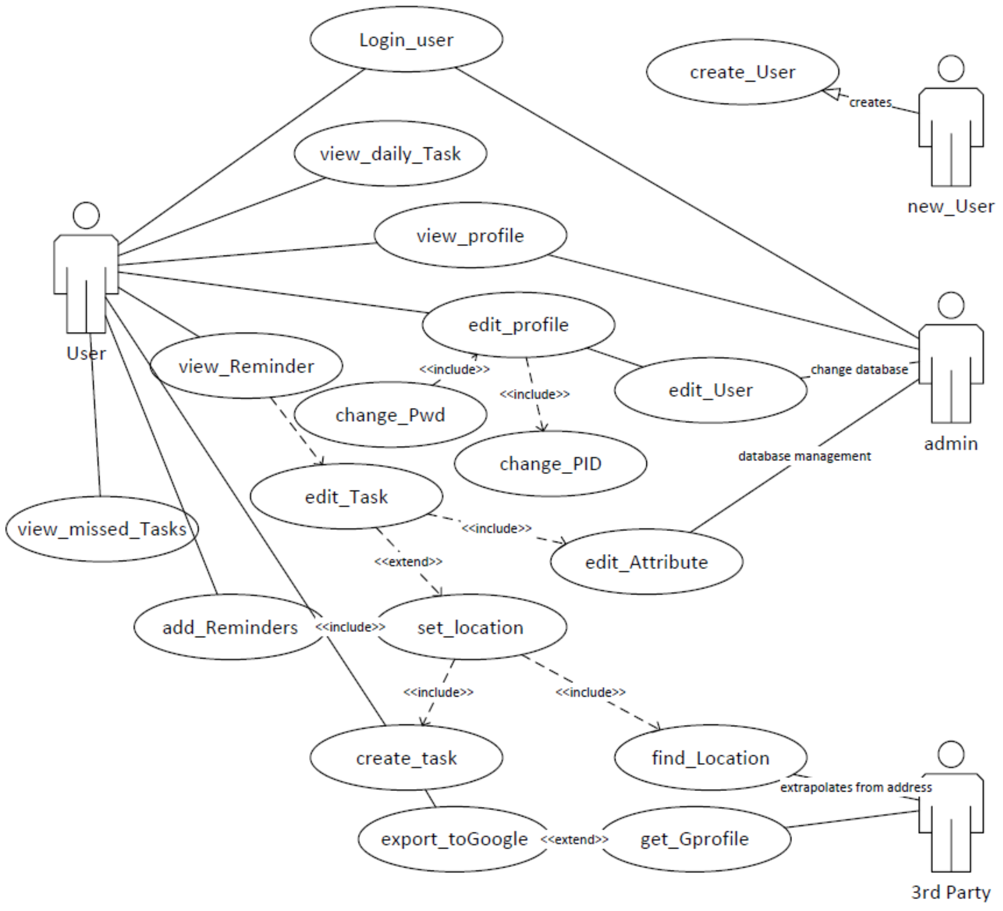


*Figure **Error! No text of specified style in document.**-1 General diagram of system architecture*

The overall architecture that is used is a client-server architecture along with components of a cloud based architecture. The client-server portions are used for HTTPS requests made from an end-user client, as seen by the Android phone object in the Figure above, while the server is used to process these requests, store information in the database, and return any requested information.

3. Use cases
   A. Use case diagram



B. Use case specifications

i. LTUC1 – View Profile

Use Case:	LTUC1
Use Case Name:	View Profile
Overview:	Open the user's profile to view configurations such as user info.
Type:	Primary
Actors:	User
Pre-Condition:	User has another tab open The application is on and logged into
Post-Condition:	The user's profile is open and the user can view their profile.
Basic Path:	1. The user opens the application 2. The user is logged in and the daily reminders page opens 3. The user selects the profile image at the top of the screen. 4. The user selects "view profile" 5. The application opens up the profile page for the user to interact with.
Alternative Paths:	2b. The user logs into the application and the daily reminders page opens.

ii. LTUC2 – View Reminders

Use Case:	LTUC2
Use Case Name:	View Reminders
Overview:	Look at the list of recurring reminders and view what is available.
Type:	Primary
Actors:	User
Pre-Condition:	User has another tab open The application is on and logged into
Post-Condition:	The page of reminders is open and the user's reminders are listed in rows.
Basic Path:	1. The user opens the application. 2. The user logs in and the daily reminders page opens. 3. The user selects the "View Reminders" button 4. The application opens up the page of reminders, soring them by preference.
Alternative Paths:	2b. The user logs into the application and the daily reminders page opens.

iii. LTUC3 – View Missed Tasks

Use Case Number:	LTUC3
Use Case Name:	View Missed Tasks
Overview:	User views the missed tasks of the week
Type	Primary
Actors	User
Pre-Condition	User has selected to view their missed tasks for the week User has used application for more than 1 week
Basic Path:	1. User logs in 2. User selects "View Missed Tasks" at <location TBD> 3. If the user missed any tasks within the past 168 hours (7 days), they will be displayed
Alternative Paths:	3a. If the user has no missed tasks, UI displays "No missed tasks! Keep it up!"

iv. LTUC4 – Edit Tasks

Use Case Number:	LTUC4
Use Case Name:	Edit Task
Overview:	User edits previously created task
Type	Primary
Actors	User
Pre-Condition	User has created at least one task
Basic Path:	1. User logs in

	2. User selects a task from their dashboard 3. User selects “Edit Task” 4. User edits selected attribute 5. User saves changes and closes out of “Edit Task”
Alternate Path:	3a. If user chooses to edit the Task Name, they will enter a new string of their choosing and select save.
	4a If the user choose to edit the reoccurrence of the task, they will choose “Single Task”, “Daily Habit”, or “Reoccurring Task”

v. LTUC5 – Login

Use Case Number:	LTUC5
Use Case Name:	Login
Overview:	A registered user will be able to login to their account and be led to the daily tasks screen.
Type	Primary
Actors	User
Pre-Condition	Customer has opened the mobile application and is not currently signed into their account
Main Flow	1. User is shown a form with textboxes for the “Username or Email” and “Password”. 2. The user enters their account details in these textboxes and then submits this information by pressing the “Sign In” button. 3. After the button is pressed, an HTTPS POST request with the filled form information is sent to the external server. 4. If the username and password are valid according to the server, then the server will send a user back and appropriate session key or other form of authentication. 5. If the server response is received in 2 seconds or less, then the user will be taken to their daily tasks screen.
Alternate Flow	4a. If the username and/or password are invalid, the server will send a message back accordingly, and an error message will be displayed on the user’s screen stating that, “Login failed due to invalid username and/or password. Please try again”.
	5a. If the response from the server takes longer than 2 seconds, then the user will receive a timeout error message on their screen stating, “Could not login to your account, the connection timed out! Please try again.”

vi. LTUC6 – Add Task

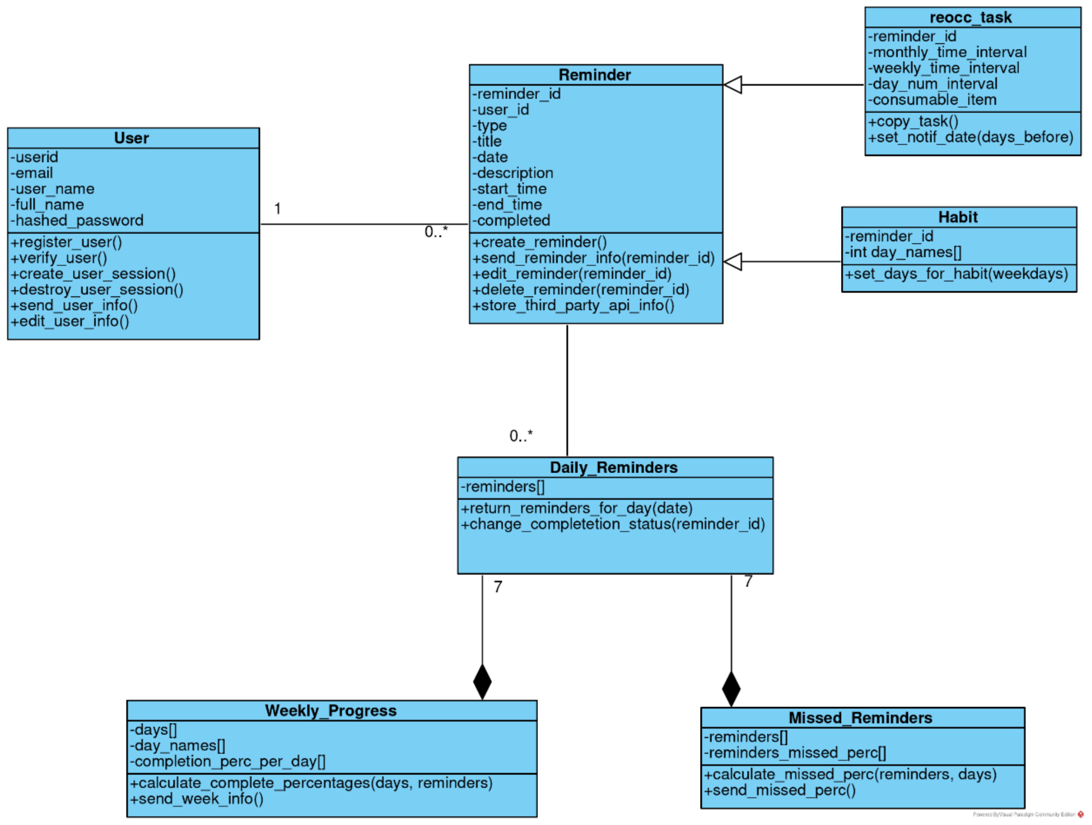
Use Case Number:	LTUC6
Use Case Name:	Add Task

Overview:	A registered user will be able to add a task with options for the task to be a one-time task for the current day, a task
Type	Primary
Actors	User
Pre-Condition	Customer has opened the mobile application and is not currently signed into their account
Main Flow	1. User presses the circular plus icon at the bottom riht of the "Daily Tasks" page 2. User fills out, "Title", textbox with a descriptive name for their task 3. The user will then select the type of task 4. The user will then select and fill out options related to the type of task chosen in Step 3. 5. The user can optionally use a tool like Google Maps to add additional information and functionality to a task, resulting in an API Call to a third-party service. 6. The user can then press a "Submit" button after any necessary information is filled out. 7. After the button is pressed, an HTTPS POST request with the filled out form information is sent to the external server. 8. If a server response is received in 2 seconds or less, then the user will be taken to their daily tasks screen and a small success message will be displayed at the bottom of the screen stating "Task Successfully Added!". The task is now stored on both local storage and external server.
Alternate Flow	1-6 At any time during these steps, a user can cancel the addition of a task by pressing a back arrow that is located on the bottom left of the page. This will redirect to the Daily Task Screen.
	5a. If the response is not received from the third-party API within 5 seconds, the user will be taken back to the task addition screen and notified that an error with the plugin was encountered and to retry again later.
	8a. If the response from the server takes longer than 2 seconds, then the task will be stored on the user's local storage but will display an error regarding the timeout. "Could not add task to server. Task will be synchronized with the server later."

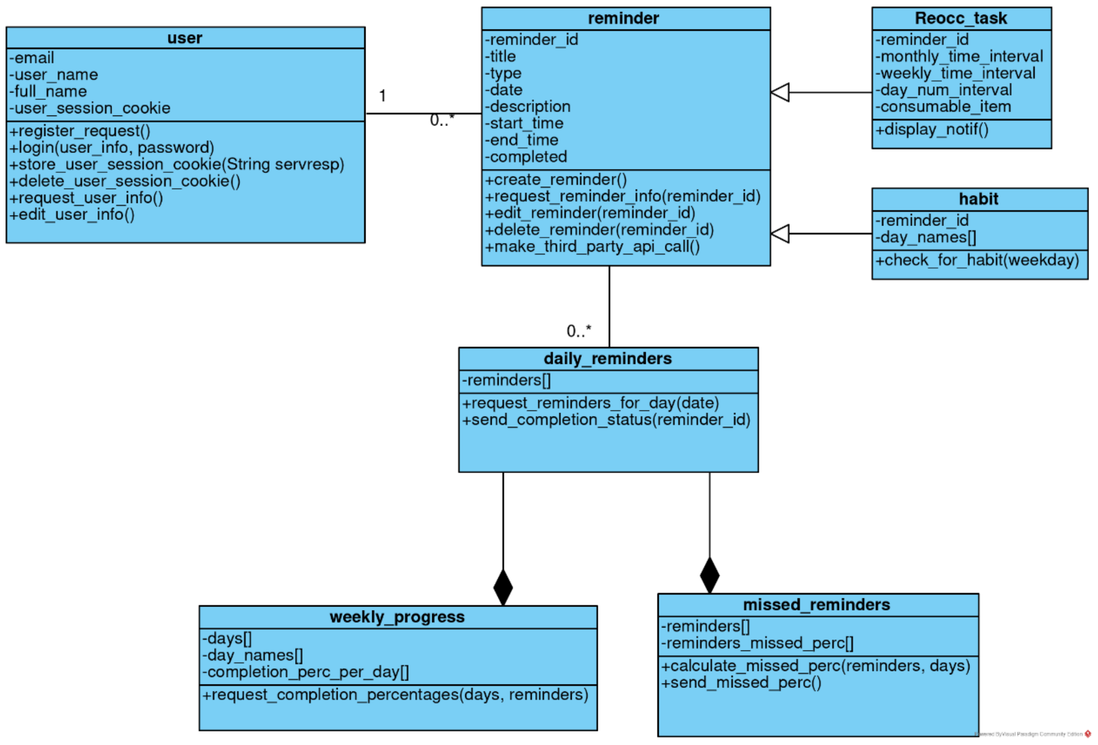
4. Classes and their interaction

   A. Class diagrams

Server-side class diagram:



User-side diagram:



B. Details of classes

Server side classes:

i. C1 –User Class

Attributes:

Attribute name	Type	Description
userid	Int	A variable that will store userid information. This will act as a primary key in the user table within the database that autoincrements with each new user (the specific type in the mysql database is bigint(20))
email	String	A variable that will store a user’s email. This will be a unique attribute in the database, and can also be used whenever a user is logging in.
user_name	String	A variable that will store a unique username for a user of the system. This can be used whenever a user is logging in, and is also a part of a user’s profile information.
Full_name	String	A variable that will store the full name associated with a person for profile information, as well as for demographic purposes.
Hashed_password	String	This will be a hashed version of the password a user gives when registering, and will also be used for verifying a user when they login.

Operations/Methods:

Method Name	Arguments passed	Expected return value	Description
Register_user()	User_name, email, password, full name (optional) (these are gathered from an HTTPS POST request's parameters)	String	A function that will be used whenever a user is registering for the first time. After the server receives a registration request, it will first make sure that the user_name and email do not mass any previous user_names or passwords. If it doesn't, then the server will add the user to the database and send back a success message. Otherwise, an error message will be sent back.
verifyuser()	User_name (or email), password (these are gathered from an HTTPS POST request's parameters)	String	A function used to verify a user when a login request is made. If both the user_name/email and hashed+salted password match what is in the database, then the create_user_session function is called and the server gives the user_id to the session creation function. Otherwise, the server will wait 2 seconds, and then return an error message.
create_user_session	user_id	Cookie	If this function is called from verify_user, then a session based on a user's id is created on the server, and a cookie with the session information is returned to the client.
destroy_user_session	user_id	none	If a user logs out of the application, then the server will find the user_id associated with that user and destroy any current server sessions that user has.

Send_user_info	User_id	JSON	If a user requests to view their profile, then the server will retrieve the information based on their user_id, and return their associated profile information in a Javascript Object Notation format (JSON) for the user client to parse and display
Edit_user_info	User_id, JSON	None	If a user sends a request to edit their information, then the server will parse the user's new information in the JSON object, and then make changes in the SQL database accordingly.

ii. C2 – Reminder

Attributes:

Attribute name	Type	Description
reminder_id	Int	A variable that will store the unique id for a reminder. This will act as a primary key in the reminder table within the database that autoincrements with each new reminder (the specific type in the mysql database is bigint(20))
User_id	Int	A variable that will store the userid associated with the user who created this reminder. This will act as a foreign key in the reminder table, and since a reminder has to be created by a user, this means that both the reminder_id and the user_id are composite keys.
type	String	A variable that will describe the type of reminder, which is either an single event, reoccurring task, or a habit.
title	String	A variable that will store the name, or descriptive, title of a reminder.
date	Date	This will be a variable that stores the date of the day in which the reminder will take place.
Description	String	A variable that will store any descriptive info about a reminder the user includes.

Start_time	Time	A variable that stores the starting time for a specific reminder if needed for the reminder.
End_time	Time	A variable that stores the starting time for a specific reminder if needed for the reminder.
Completed	Boolean	A variable that will indicate whether or not a task has been completed.

Operations/Methods:

Method Name	Arguments passed	Expected return value	Description
Create_reminder()	JSON data that contains the title of a task, it’s reminder type, and optional information such as it’s start_time and end_time, it’s time intervals if it’s a reoccuring task, and it’s Day names if it’s a habit	None	A function that will parse JSON information given to it from a user requesting to make a new task. It will then store the parsed information in the database.
Send_reminder_info(reminder_id)	Int (reminder_id)	JSON	A function that will return specific reminder information when a request is made using the reminder’s id. This will commonly happen when a user actually selects a task in the daily tasks screen.
Edit_reminder(reminder_id)	reminder_id, JSON	None	If a user sends a

			request to edit a reminder's information, then the server will parse the reminder's new information in the JSON object, and then make changes in the SQL database accordingly.
Delete_reminder	Int (Reminder_id)	none	When a user requests to delete a reminder based on it's id, then the server will remove the reminder from the databased accordingly, as well as any copied reminders that match it.
Store_third_party_api_info	JSON	None	This will take any information returned from a third party api call made by an end user and then sent to the server, parse it, and then store it in a table designed for that third party api,

			and associate it with the proper reminder_id

iii. reocc_task

Attributes:

Attribute name	Type	Description
reminder_id	Int	The reminder_id that a reoccuring task subclass will be attached to in the database.
Monthly_time_interval	Int	A variable that represents the amount of months that a task will happen again at.
Weekly_time_interval	int	Represents the number of weeks in which a reminder will happen again.
Day_num_interval	int	Represents the number of days in which a task will occur again.
Consumable_item	bool	A Boolean that indicates whether a reminder involves a consumable item or not. For example, if a person buys an air filter, then task will know to potentially store amazon api related information for the task so the person can easily buy the item again when the reminder comes up.

Operations/Methods:

Method Name	Arguments passed	Expected return value	Description
Copy_task()	JSON data that contains any information a reoccurring task object has	None	A function that will take information a reoccurring task contains and then copy the

			information to future dates based on the time interval information stored in the task.
Set_notif_date(days_before)	Int (days_before)	None	A function that will set the days before a task happens given by a user to notify the user ahead of the task

iv. Habit

Attributes:

Attribute name	Type	Description
reminder_id	Int	The reminder_id that a reoccuring task subclass will be attached to in the database.
Int day_names[]	Integer array	An integer array that represents the days of the week. For instance 1 = Monday , 2 = Tuesday … 7 = Sunday

Operations/Methods:

Method Name	Arguments passed	Expected return value	Description
Set_days_for_habit(weekdays)	Integer array	None	A function that will take the days of the week that a habit will reoccur on through an integer representation (1 = Monday , 2 = Tuesday … 7 = Sunday, 8 = Everyday)

v. Daily_Reminders

Attributes:

Attribute name	Type	Description
Reminders[]	Reminder object list	A list of reminders and their attributes that are associated with a date. This will be temporarily stored on the server when a query for a specific day is made.

Operations:

Method Name	Arguments passed	Expected return value	Description
Return_reminders_for_day(date)	Date representing a day	JSON data that contains the following information: reminder id, title, and completion status	A function that will take a user's request for reminders either on the user's current day, or on another specified date, and return a JSON document containing core information for each reminder that day.
Change_completion_status (reminder_id)	Int (reminder_id)	None	A function that will occur when a user checks a checkbox next to a reminder. When the user performs this action, it will send a

			HTTPS POST request with a Boolean represented in text form indicating the completeness of a reminder, and the server will then change the information in the database accordingly.

vi. Weekly_Progress

Attributes:

Attribute name	Type	Description
days[]	Dates list	A list of dates making up an entire week
day_names[]	String list	A list representing the day name associated with a date in the days list above
Completion_perc_per_day[]	Double list	A list representing the percentage of tasks that were completed on a particular day

Operations:

Method Name	Arguments passed	Expected return value	Description
Calculate_complete_percentages(days, reminders)	Dates list, reminders on each date	Double list	A function that will count the number of reminders on each day in a week, and then count the number of reminders that were complete. It will then perform basic math to determine a percentage

			of completion for that day, and then return an Double type list representation of the week, with each index of the list containing the completion percentage
Send_week_info()	date	JSON document representing the date representing the start of the week (on Sunday), the weekdays, and the completion percentage for each weekday	A function that occurs when a user selects the, “Weekly”, page. Once the user has made a request for the information, the server will call the calculate_complete_percentages function based on the appropriate reminder id’s, and then return a JSON representation of the user’s overall task completion percentage per day.

vii. Missed_Reminders

Attributes:

Attribute name	Type	Description
reminders[]	reminders list	A list of reminders that can be found throughout a week.
Reminders_missed_perc[]	Double list	A list representing the percentage that a reminder has been missed throughout a week.

Operations:

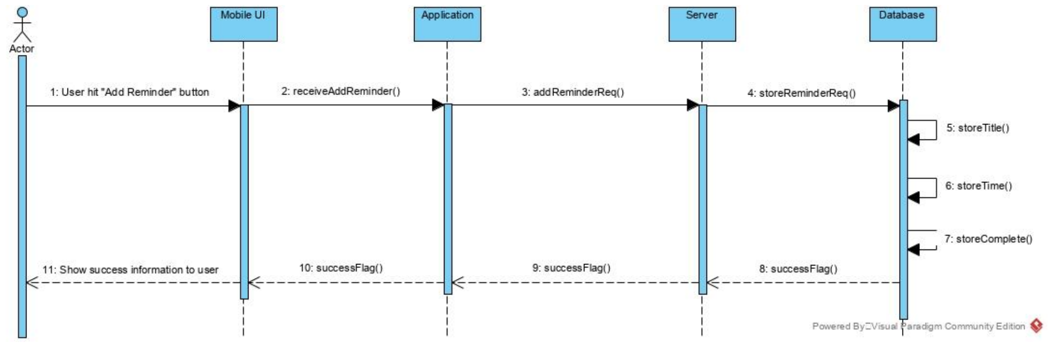
Method Name	Arguments passed	Expected return value	Description
Calculate_missed_perc(reminders, days)	Dates list, reminders on each date	Double list	A function that will count the number of times a specific reminder will appear in a week. It will then use basic math to determine how much of this reminder has been missed through a week and then calculate a percentage from this
Send_week_info()	date	JSON document representing each task that can occur in a week, and the percentage representing the amount of times they missed that task	A function that occurs when a user selects the, “Missed tasks”, page. Once the user has made a request for the information, the server will call the calculate_missed_perc function based on the appropriate reminders, and then return a JSON representation of the user’s overall percentage of the times they missed a particular task in a week.

User side classes:

The attributes representing data and their types for both the server-side portion and end user portion are similar. The only notable differences for a user is that they will not have their user id for security purposes, and they also will have a session cookie object stored in their User data object, rather than a hashed password, for security purposes. Also, a user will have functions for requesting information, such as, “request_completetion_percentages(days,reminders)”, while the server will perform the majority of the retrieval and returning of information based on requests. The only exception are create and edit requests, which will form new, or edit, previous information on the user side, and then the server will make the proper data adjustments accordingly.

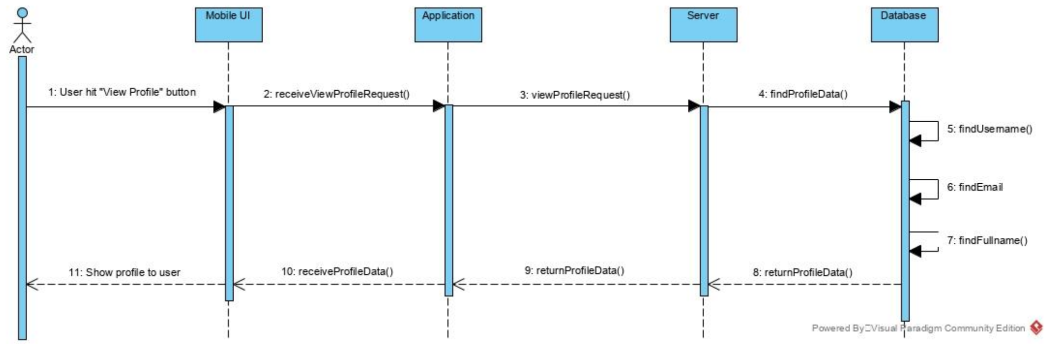
C. Sequence diagrams

Add Reminder:



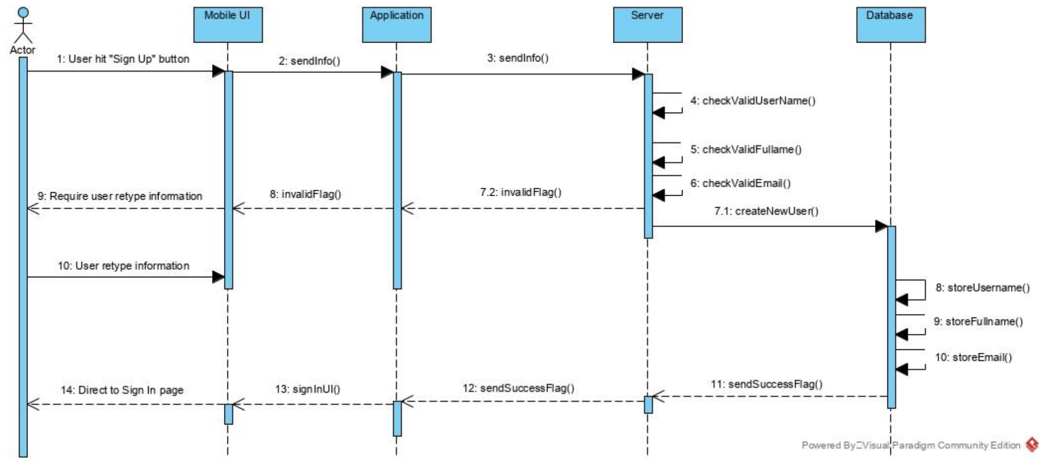
After the user fills out the information and taps the "Add Reminder" button, the UI will prompt application to send addReminderReq() to the server. Once the server receives the request, the database will then store the information. After successfully storing it, the database will send successFlag() to the server that is then forwarded back to the application. This success notification is then displayed to the user.

View Profile:



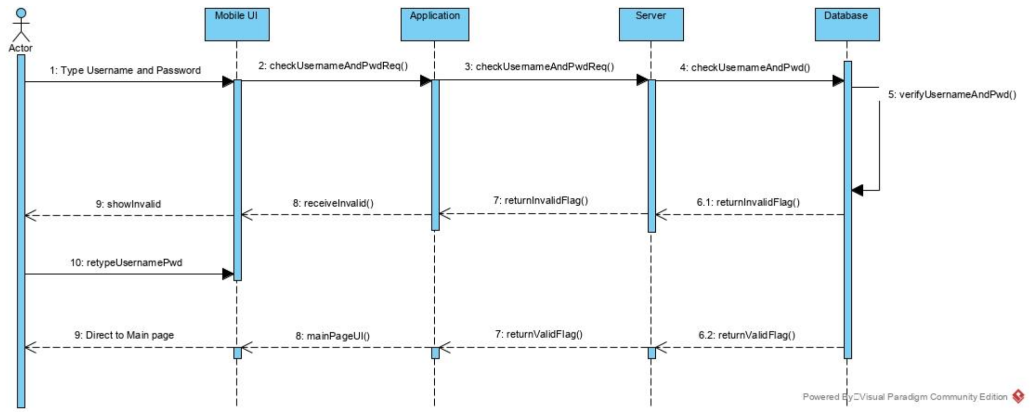
When the user taps on "View Profile", the application will send a viewProfileRequest() function to the server. After the server receives the request, it will call the function findProfileData(). The database will query itself for the username, email, and full name. If the user is found, the database will return the profile data with the returnProfileData() function. If no user is found based on the search terms, it will return a "User Not Found" message.

Create User:



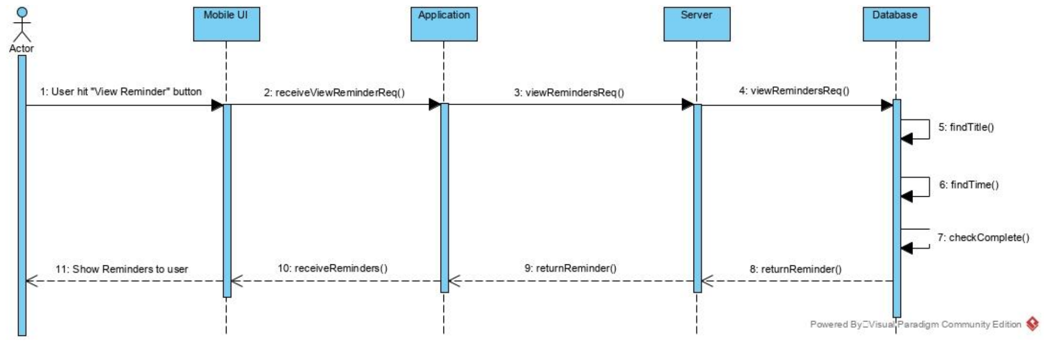
After a user fills out the information and selects "Sign Up", the application will forward to the server to verify the validity of the information. It will ensure there are no duplicate entries. If the information is invalid, the server will return a invalidFlag() function back to the application prompting the user to re-enter their information. If the information is valid, the server will send the information to the database for storage. After this, the database will send a sendSuccessFlag() function back to the server. The server will forward to the application to call signInUI() function to allow the user to sign in.

Login User:



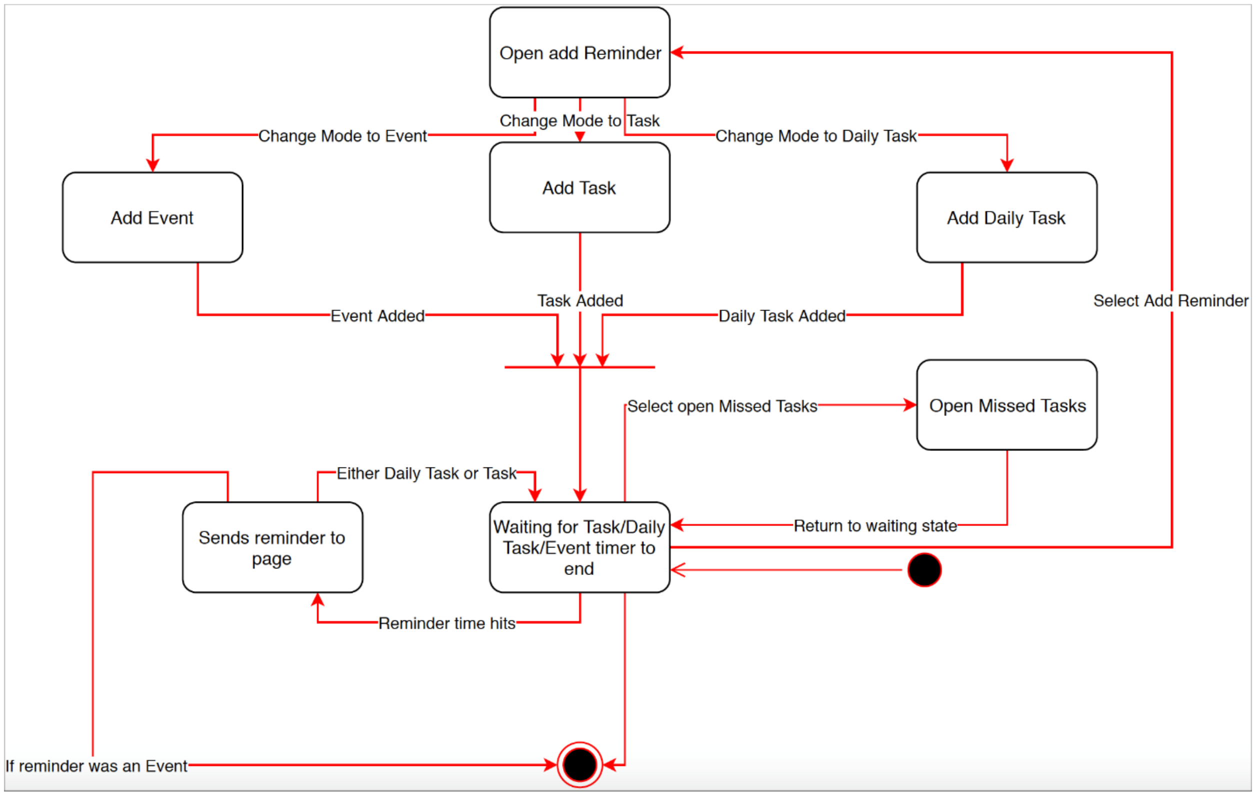
After the user types their username and password, the application will send a checkUserNameAndPwdReq() to the server. When the server receives the request, the server will forward to the database to verify the parameters. If the username and/or password are incorrect, the database will send returnInvalidFlag() function back to the server. This flag is forwarded to the user's interface and a message is displayed prompting the user to re-enter their information. If the information is correct, the database will send a returnValidFlag() function and redirects the user to the main page of their dashboard.

View Reminders:



After the user selects “View Reminder”, the UI will ask the application to send viewRemindersReq() to the server. When the server receives the requerst, it will query the database to look up every reminder based on username. This will return all information back to the server to be forwarded to the application, where it is then displayed on the screen.

5. State transition diagrams



Open add reminders - Opening add reminders will open up the page on the application to add reminders. The page will let the user select what kind of reminder they will add, which will open up a different page that will show things for the user to enter so they can create said reminders. The button to open add reminders is in the bottom right of the application and will always be able to access, unless the user is already in the process of creating a reminder.

Add Event - Add event will be a page that will show the parameters for adding an event for the user to enter. The 'Event' will be a one time task that will not refresh after it goes off once, so it will disappear after the reminder is complete. The user will enter date, name, and optionally time, location, and notes.

Add Task - Add task will be a page that will show the parameters for adding a task for the user to enter. Add task is a reminder that will iterate at any possible pattern that the user wants. The user will enter frequency for reminder for the task, name, optional time, location, and notes.

Add Daily Task - Add daily task will be a page that will show the parameters for adding a daily task for the user to enter. A Daily task is one that goes everyday, and has a set time it goes off. This is where the user will set a weekly pattern for daily reminders, for their schedule. The user will be asked to enter in Time, frequency in the week, name for the task, optional location, and notes on the task as well.

Open missed tasks - open missed tasks will open a page of previous tasks that were missed for either the daily, or task/event. It will also output user specified parameters for what analytics they may want to use. (i.e. amount of things missed on Monday, how many times they missed brushing their teeth in a week, etc.)

Waiting state - the waiting state is waiting from any page for reminders alarm the user, or waiting for user input on the application. Any page can be open for this process to be running. This is the default state for the state transition diagram. All functions will fork off of this state.

Send reminder to page - This state is when the reminder sends a notification to the page/user. This will create whatever notifications the user had specified and engage the users with said reminders. This process with run with the app both opened and closed.

6. User interface design
    A. Screen 1 – Login screen
       Image:

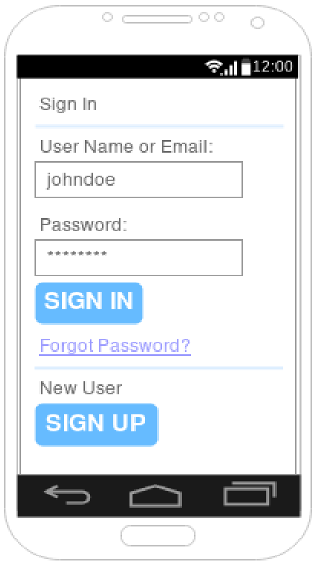
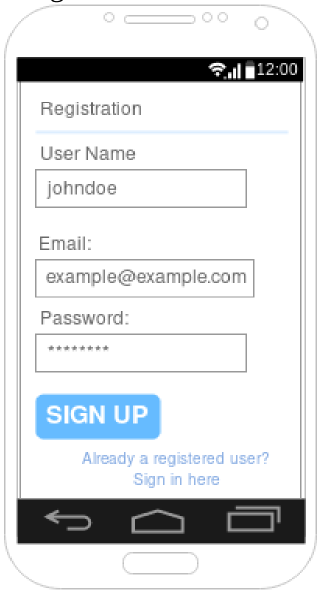


*Figure **Error! No text of specified style in document.**-1 - Login screen Source: draw.io*

Description: Upon opening the app for the first time after installation/reinstallation of the app, or if a user has deleted the application's cache and data (explain what this means), the user will be shown a login screen, as can be seen in Figure 2-2 below. The user then has the option of logging into a preexisting account using either an email/username as well as their password if they have one. If a user does not have a preexisting account or needs a new account, then they can select the Sign-Up button if they are a new user. If this button is selected, then the user will see the registration screen which is described in Screen 2 below.

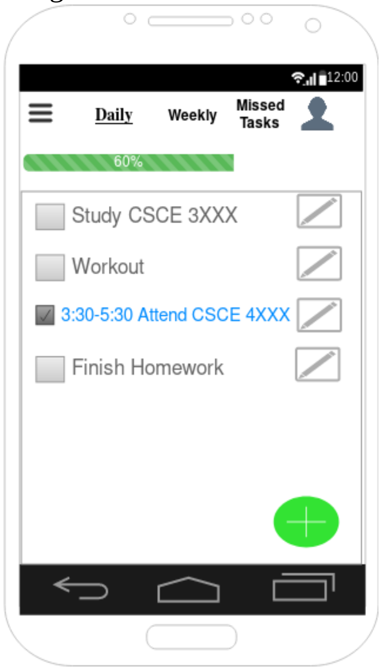
B. Screen 2 - Registration screen

Image:



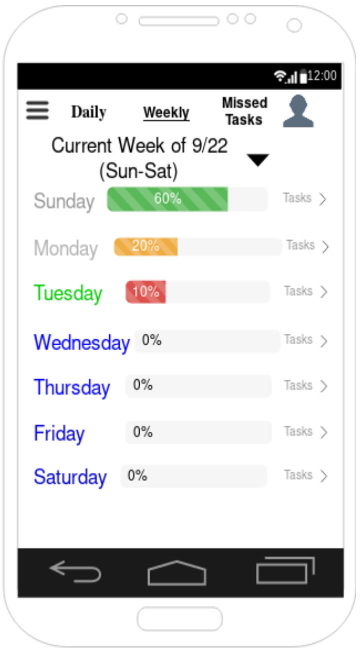
Description: This screen will allow a user to enter an email, password, and username to create a new account, with the option to enter additional details later.

C. Screen 3 - Daily tasks screen
Image:



Description: Upon a user's repeated opening of the app, or after they have logged in, they will start at this daily task screen. This screen will allow a user to view the tasks they have to accomplish today, as well as a progress bar showing their progress for the day. The user can indicate they have finished a task by checking it off the list using the check mark buttons to the left of athe task. Additionally a user can select, or tap on some tasks to retrieve information such as location of an event on Google Maps, or places to buy an item that is needed that day through third-party APIs such as Amazon.

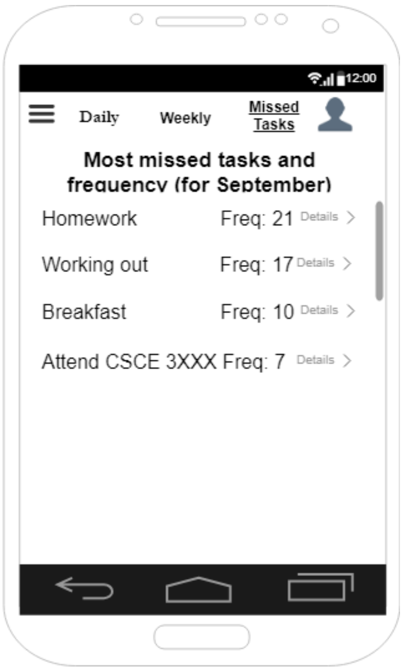
D. Screen 4 – Weekly progress page
Image:



Description: This screen above can be accessed by simply selecting the text, "Weekly", menu option at the top center of the screen. This screen can show task completion progress of the days in the current week, as well as tasks in previous weeks. By default, the progress of the current week will be shown to the user. If the user wants to look at task completion summaries from previous weeks (or further weeks, depending on the currently selected week), then they can do this by selecting the downward facing arrow in the top left of the screen next to the currently displayed week. By clicking on Tasks next to a day of the week, it will show them the specific tasks they marked as completed for that week, similar to the daily tasks screen.

E. Screen 5 – Missed tasks page

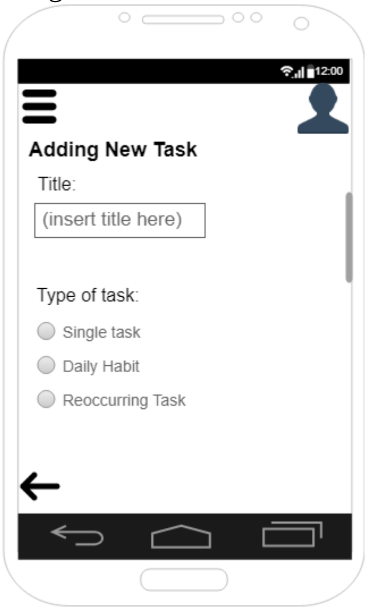
Image:

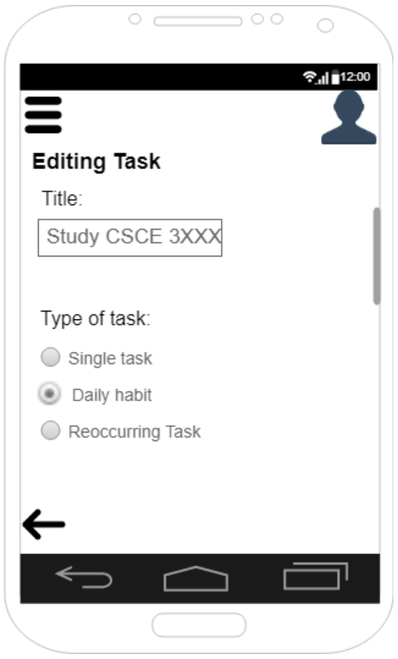

12:00
Daily
Weekly
Missed Tasks
Most missed tasks and frequency (for September)
Homework Freq: 21 Details
Working out Freq: 17 Details
Breakfast Freq: 10 Details
Attend CSCE 3XXX Freq: 7 Details

Description:

F. Screens 6 and 7 – Add and edit tasks pages

Images:





7. Conclusion
   Overall, the group has learned the importance of defining what we want from the software as well as clearly defining the goals of our program from designing this document. We have learned that aspects such as class diagrams and ui design are important for giving a broad outlook on the design of the software, while state transition diagrams and sequence diagrams are important for clearly defining the functionality that supports the broad design of the system.