AllinWallet

All-in-one Personal Finance App

Team 26 Design Document

Team Members

Junyi Zhang

Ray Chen

Rohith Sridharan

Shaan Hoda

Yufei Xu

Purpose

Without a proper budget or indication of how much to spend, people on average will most likely spend more money than they thought they actually did. The problem is that people can easily go over their budget if they don't know how much money they've already spent and how much money they have left. The target users of our project are the people who fail to properly manage their budget on their own.

Due to this people have a hard time managing how much money they should spend and save. Therefore we are creating an app to allow people to enter budgets and keep track of much money they plan on spending. The app will have features which let the user also budget and save incomes for future use. The first and foremost feature of this application is a finance management which indirectly should benefit user spendings.

Functional Requirements

1. As a user, I would like to register for an AllinWallet account so that I can synchronize my profile and data across devices.
2. As a user, I would like to log in with a username and password.
3. As a user, I would like to log out of the app so that I can prevent other people from accessing my profile.
4. As a user, I would like the ability to permanently delete my account profile.
5. As a user, I would like the ability to report issues to the project developers.
6. As a user, I would like the ability to backup and restore my data so that I can undo any potential mistakes or prevent data loss.
7. As a user, I would like to add my email to my AllinWallet account so that I can reset my password through email.
8. As a user, I would like to set a picture as my profile avatar and change it if I wish.
9. As a user, I would like to use this application with a language of my choosing.
10. As a user, I would like to choose the currency I am using.
11. As a user, I would like to see some money saving tips while the app is on the loading screen.
12. As a user, I would like to have many different themes to choose from so that I can customize the look of the app.
13. As a user, I would like to quickly access the app features that I use the most.
14. As a user, I would like to enter the amount of money I have just spent and create a purchase in my account profile.

15. As a user, I would like to add a title to my newly created purchase.
16. As a user, I would like to choose a category for my newly created purchase.
17. As a user, I would like to add location and time tags to my purchase.
18. As a user, I would like to take a picture of the sales receipt and add it to my purchase.
19. As a user, I would like to delete unwanted purchases that I entered in accidently.
20. As a user, I would like to create a weekly, monthly, or annual budget so that I will be notified when I'm about to go over/exceed my budget.
21. As a user, I would like to see my weekly, monthly, or annual spend analysis report.
22. As a user, I would like to be able to sort the purchases in my spend analysis report by time/category/title/cost.
23. As a user, I would like to see a spend analysis report for a specific period of time.
24. As a user, I would like to add recurring charges (e.g. cell phone bills) to my profile so that those spendings appear in my spend analysis report automatically.
25. As a user, I would like to see what category I have spent money on the most.
26. As a user, I would like to view my purchases on a calendar so that I can see the amount of money I spend each day.
27. As a user, I would like the ability to search a specific purchase by title.
28. As a user, I would like to limit my search results by category and/or range of cost/time.
29. As a user, I would like the start screen/landing page of the app to display as much relevant information (e.g. budget, income, spending, etc) as possible so that I don't have to navigate to other places for these information.
30. As a user, I would like to see a graph visualization (e.g. percentage bar, pie chart) of categories that I have spent money on.
31. As a user, I would like to know how much money I was under my budget for a giving period of time.
32. As a user, I would like to know if my money spending habits have improved since I started using the app.
33. As a user, I would like to share my activities and progress I made on social media.
34. As a user, I would like the ability to know how many people are currently using this app.

Non-Functional Requirements

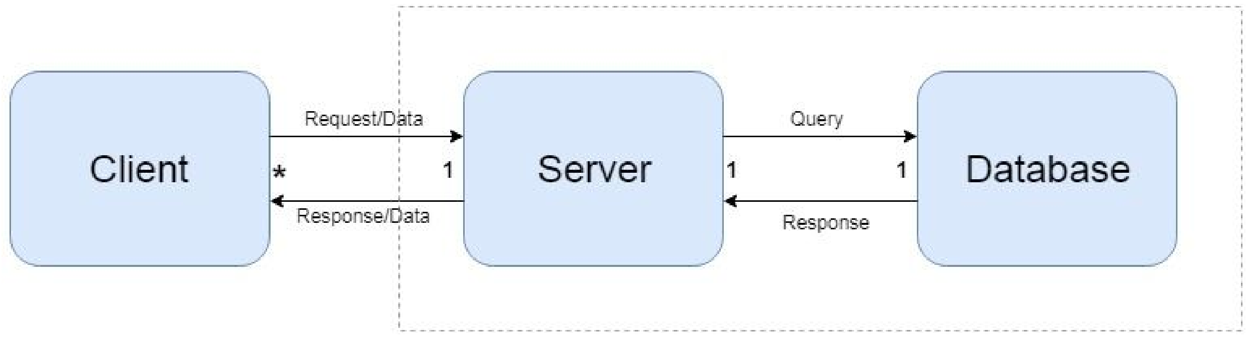
1. As a first-time user, I would like to learn a sufficient amount about the app in a relatively short amount of time.
2. As a user, I would like to see a friendly UI.
3. As a user, I would like my personal information to be secure.
4. As a user, I would like to be able to voice my opinions and offer feedback that could potentially improve the app (e.g. make any additions or subtractions that would be beneficial).
5. As a developer, I would like our software to be easily maintainable.
6. As a developer, I would like to protect the privacy of users and developers of this app by adding extra layers of security.
7. As a developer, I would like our software to be used only in its intended context so that it will not cause harm to the people or environment.
8. As a developer, I would like to obtain plentiful feedback from our users in hopes of improving the application.

Design Outline

Our project will be an Android app that helps users to manage their budgets. We will use a Client-Server model in this project. There are three major system components: client, server, and database. We will build a fat-client because most of the user interactions are handled locally in the Android client. The server and database components will add profile synchronization, access restriction, and data recovery functionalities to this project.

1. Client
    a. The majority of the user interactions is handled directly in the Android client.
    b. The Android client sends data/requests to the server in JSON format.
    c. The Android client may need to download data from the server in some cases.
2. Server
    a. The server is the bridge between the Android client and database.
    b. The server processes data/requests from the Android client.
    c. The server sends queries to the database.
    d. The server receives data from the database and sends appropriate responses back to the Android client in JSON format.
3. Database
    a. The database stores the data needed for user authentication and profile synchronization.
    b. The database handles queries from the server and responds with data.

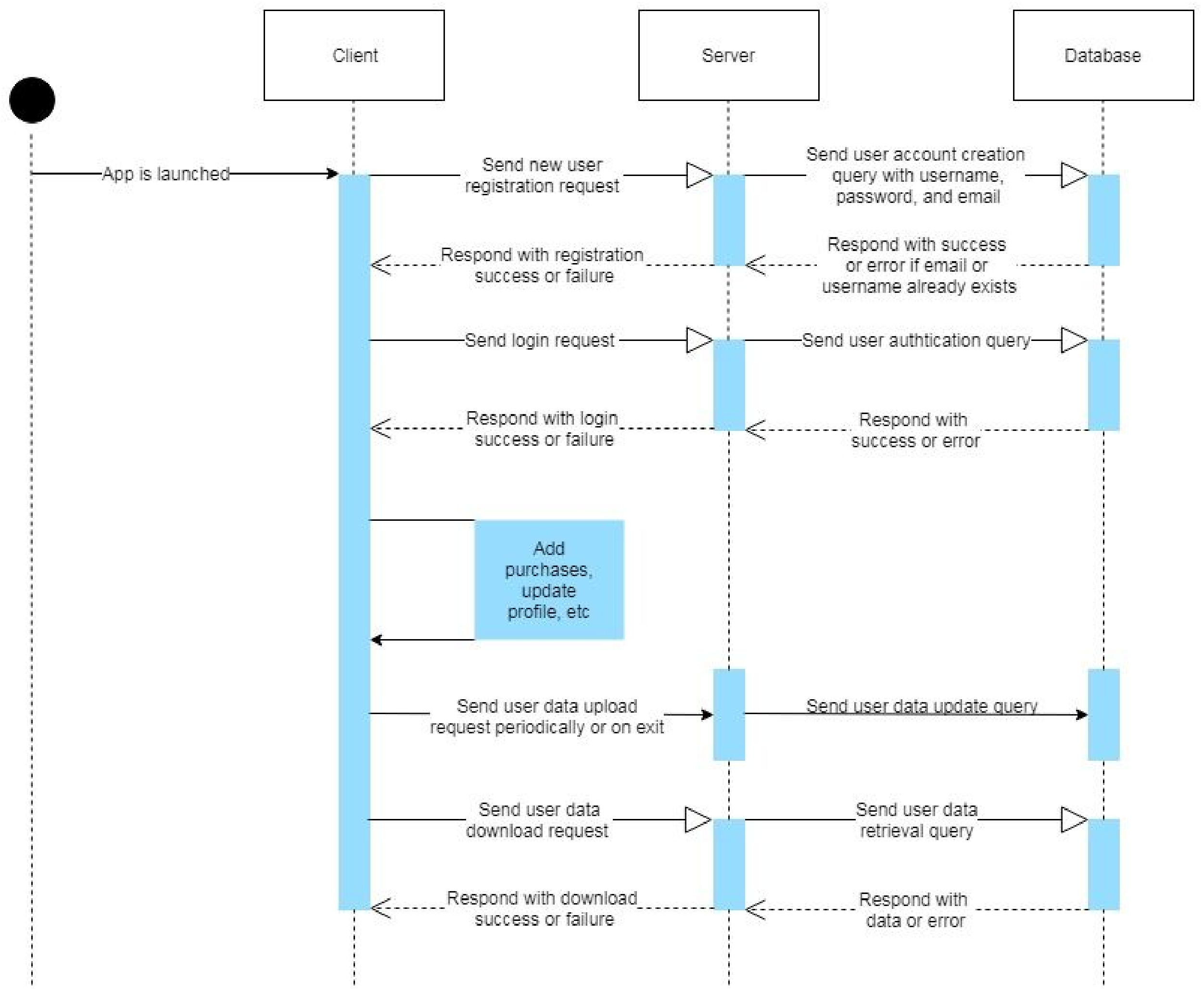
High-Level Structure View of Our System



Sequence Diagram of Interactions between Components

The diagram in the next page shows some typical flow of events among clients, server, and database.

- In order to use the app, a user must first create an AllinWallet. The server receives the user's request and queries the database. The database responds with success if the username or email doesn't exist yet, and error otherwise. The server will then forward an appropriate response to the client.
- Once the account is created and the user is trying to log in, the client will send out a login request to the server. The server and database will process this request and respond with success if a valid combination of username and password is provided.
- Once the app is up and running, the client can examine the user data for recent changes and upload the data to the server if necessary. This examination can be done either periodically or once on the exit of the app. The server will then query the database to update the data belongs the specific user. In some cases, the client may have to download user data from the server and database.


Client
Server
Database
App is launched
Send new user registration request
Send user account creation query with username, password, and email
Respond with registration success or failure
Respond with success or error if email or username already exists
Send login request
Send user authtication query
Respond with login success or failure
Respond with success or error
Add purchases, update profile, etc
Send user data upload request periodically or on exit
Send user data update query
Send user data download request
Send user data retrieval query
Respond with download success or failure
Respond with data or error

Design Issues

Functional Issues

1.) What kind of information do we require for creating an account?

- Option 1: Username and password
- Option 2: Username, password, and email address
- Option 3: Username, password, email address, and phone number

Decision: Option 2

Reasoning: In order to set up/log into an account, a username and password will be necessary for the user. Since each username will be unique to each user, it will used to represent his or her data stored in the database. If a user happens to forget their password, the provided email may be used to reset the password. The email address should also prove to be a nice addition where users can receive emails regarding the weekly/monthly information, updates, new additions, etc. of the application. A phone number can be added if they wish, but not needed for signing up for the app.

2.) What kind of/how many currencies will be available in this app?

- Option 1: $ US Dollars
- Option 2: Several different national currencies
- Option 3: None, the user simply inputs their costs of purchases made regardless of the currency they're using

Decision: Option 3

Reasoning: While this application is generally geared towards college students to help them with expenses and save money, it can certainly be used by anyone looking to improve their financial spending. In that case, it shouldn't really matter what currency one uses...it's really about what one's numerical goals/limits are and entering in the costs of purchases made by an individual. For example, if you set a limit to a monetary value of 500 per week, the app will simply inform you of progress and how close you are to that particular limit (Of course, the user can manually enter in a currency symbol before the number, but the point is that the user is meant to already be aware of the currency he or she is using).

3.) How often should the user receive status updates/warnings regarding goals and limits?

- Option 1: Only warnings/updates when the goal and/or limit is nearly fulfilled
- Option 2: Option 1 + occasional progress updates (ex. Once every two days)
- Option 3: Daily updates regardless of progress

Decision: Option 2

Reasoning: This choice should be a good balance. The user most likely cares most about how close they are to meeting their goal or reaching their set limit. It might be overkill to provide the user with daily updates if their progress isn't even close to their set goals/limits. Therefore, it would likely make most sense to offer very occasional updates, so that the app isn't sending the user everyday notifications but sending out reminders sporadically. Ultimately though if time is available, it might be most preferable to allow the user to choose how often he or she receives these status updates and warnings regarding their goals/limits.

4.) Of course, we wish for the users to save their data and not allow for anything to be lost unless they choose to permanently close their account. How should we save their data?

- Option 1: Just app data (linked to your smart device)
- Option 2: Data is stored in a database with the help of their username

Decision: Option 2

Reasoning: Even if a user is only utilizing the application on a single device, it is preferable to store their data in a database, in case they accidentally uninstall the app or their phone is wiped/reset. But this option also allows users to access their data on any other device instead of one, as they would simply just log in with their credentials. It is also a nice feature to be able to retain all purchases/pieces of information since Day 1 of using the app (we will have to be wary of storage space however), which is more easily done by storing everything in a database instead of just forcing their respective device to be in charge of the app data.

5.) What should be displayed on the loading page (between logging in and the landing page)?

- Option 1: Money-saving tips/advice
- Option 2: Brief facts regarding your expenses and/or history
- Option 3: Nothing, just the throbber.

Decision: Option 1

Reasoning: It's nicer to display some kind of information on the loading screen instead of just a simple throbber that the user looks at. Having some short facts on a user's expenses or history could be an option, but we want to be considerate of sensitive content, so perhaps those facts are better placed in a location the user can access if he/she wishes to. We think that displaying brief tips on improving expenses on the loading screen upon logging in would be quite helpful to the user.

Non-Functional Issues

1.) What programming language will we use?

- Option 1: Java (Android Studio)
- Option 2: Django (Python web framework)
- Option 3: Node.js (JavaScript)

Decision: Option 1

Reasoning: We agreed that this tool is best suited to be a mobile application as opposed to a web application. Since more of us are familiar with Android development, we decided that using Android Studio to build the app would make the most sense. Java is the language used to build Android apps, which is also what many of us are already familiar with (such as from CS 18000).

2.) What database should we use?

- Option 1: AWS
- Option 2: Firebase

- Option 3: MongoDB

Decision: Option 2

Reasoning: MongoDB is very feature-rich and likely overkill for our application. Firebase is more focused on light-weight data as opposed to larger data storage (AWS) such as images, videos, etc. Since the application is primarily intended for storing data values, Firebase likely makes the most sense, especially since it is built to be fast.

3.) What should we use for the front-end/framework?

- Option 1: Corona SDK
- Option 2: Xamarin
- Option 3: React Native

Decision: Option 3

Reasoning: These three (and other frameworks not mentioned here) might actually be fine options, but it seems that Corona SDK is more appropriate for game development. Xamarin is a solid choice, but is written in C# which we aren't very familiar with. React Native on the other hand seems to be an exciting choice due to its popularity and relevance. Ultimately, this question may be discussed in greater detail this weekend.

4.) What Android IDE should we use?

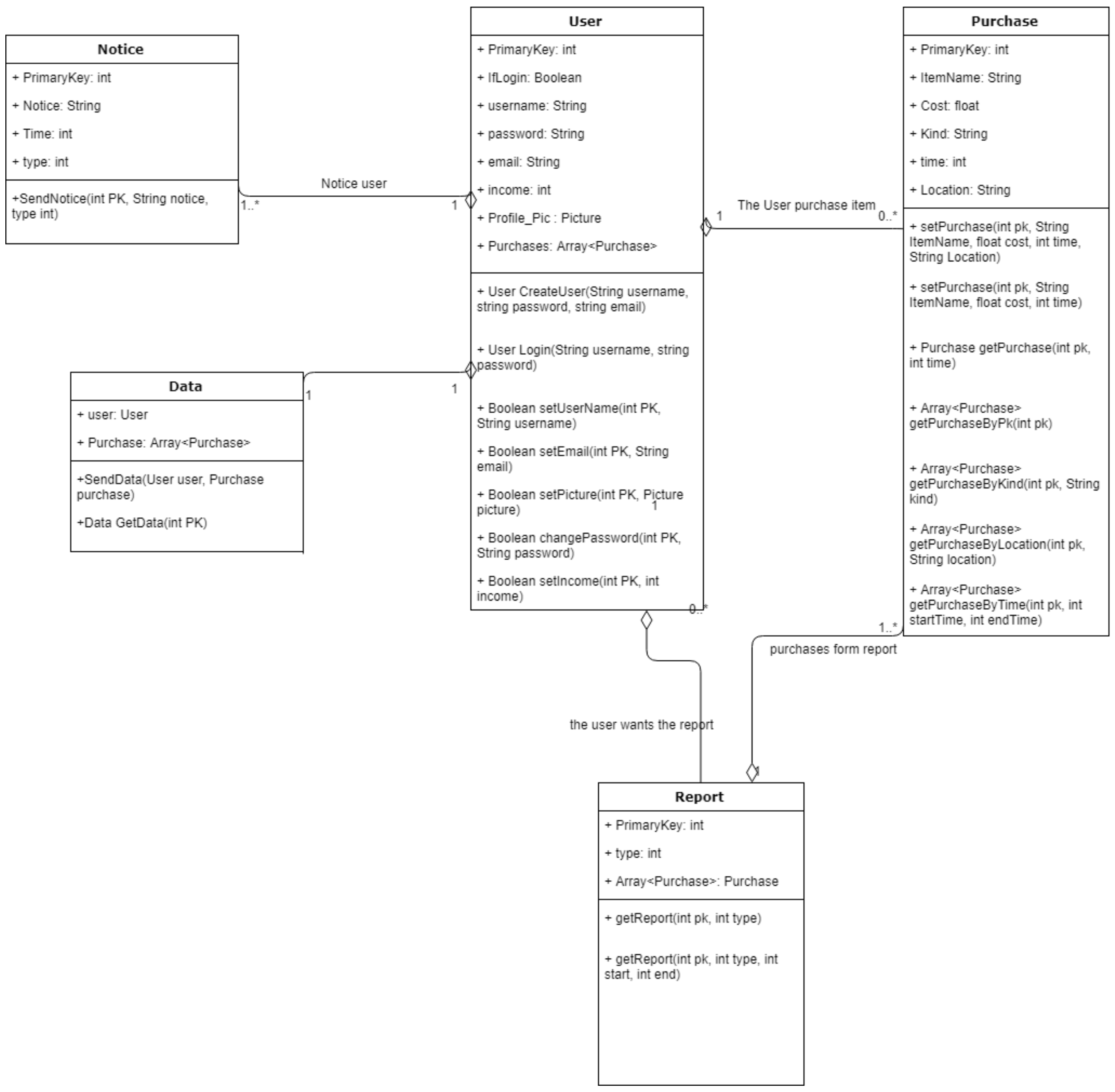
- Option 1: Android Studio
- Option 2: Eclipse
- Option 3: Visual Studio
- Option 4: Komodo

Decision: Option 1

Reasoning: Though we considered a few different options such as the ones above, we were all in agreement that Android Studio would be our best option, especially since it is Google's official IDE for Android. It is also a worthy choice since we are currently not worrying about a cross-platform app (this will only run on Android).

Design Details

Class design


Notice
+ PrimaryKey: int
+ Notice: String
+ Time: int
+ type: int
+SendNotice(int PK, String notice, type int)
Notice user
1..*
1
User
+ PrimaryKey: int
+ IfLogin: Boolean
+ username: String
+ password: String
+ email: String
+ income: int
+ Profile_Pic : Picture
+ Purchases: Array<Purchase>
+ User CreateUser(String username, string password, string email)
+ User Login(String username, string password)
+ Boolean setUserName(int PK, String username)
+ Boolean setEmail(int PK, String email)
+ Boolean setPicture(int PK, Picture picture)
+ Boolean changePassword(int PK, String password)
+ Boolean setIncome(int PK, int income)
The User purchase item
1
0..*
Purchase
+ PrimaryKey: int
+ ItemName: String
+ Cost: float
+ Kind: String
+ time: int
+ Location: String
+ setPurchase(int pk, String ItemName, float cost, int time, String Location)
+ setPurchase(int pk, String ItemName, float cost, int time)
+ Purchase getPurchase(int pk, int time)
+ Array<Purchase> getPurchaseByPk(int pk)
+ Array<Purchase> getPurchaseByKind(int pk, String kind)
+ Array<Purchase> getPurchaseByLocation(int pk, String location)
+ Array<Purchase> getPurchaseByTime(int pk, int startTime, int endTime)
Data
+ user: User
+ Purchase: Array<Purchase>
+SendData(User user, Purchase purchase)
+Data GetData(int PK)
1
1
0..*
1..*
purchases form report
the user wants the report
1
Report
+ PrimaryKey: int
+ type: int
+ Array<Purchase>: Purchase
+ getReport(int pk, int type)
+ getReport(int pk, int type, int start, int end)

Descriptions of Classes and Interaction between Classes

Our class design is based on what object we need in our domain and each of the attributes inside the class will have some method to interact with them. And the specific description of each class is as follow:

User

~ User object is created when the user log in and finish register

~ User object will be destroyed when log out or other device log in (when other device log in it will only destroy the user object in the client)

~ When register, the user will enter unique username, email and their password and this will store in our database.

~ After register each user will have a unique digit primary key.

~ Each User object will have a Picture type attributes which is the profile the users submit

~ User object has a Boolean value IfLogin to indicate if the user is already login to prevent duplicate log in.

~ Each User object will have a list of Purchases object obtained from database

~ User class provide method to change/set user's username, email, picture, password, income and login status

~ User class provide method to create user.

Purchase

~ Created when user purchase item.

~ Purchase objects will differ from each other by primary key and time since one use can only create one purchase each time.

~ One user can have multiple purchases and one purchases can only belongs to one user.

~ Each purchase will have one primary key that indicate belonging, one Item name, one cost, one kind, one time and one location.

~ Purchase will form an array(list) storing in the user object.

~ User can create Purchase by entering item name, cost. And the time and location will automatically generate and if the user don't want to add the location the program will only generate time.

~ Purchase class provide method that can get one user's all purchases using the primary key

~ Purchase class provide method that can get one kind of purchases using the primary key and kind string

~ Purchase class provide method that can get purchases in one location using the location

~ Purchase class provide method that can get purchases created between start time to the end time.

Report

~ Created when user log in or user wants to obtain report

~ One user can have multiple report and each report can only belongs to one user

~ Each report will have one primary key, one int number to indicate type, a list of purchases including in this report.

~ The type variable will indicate the report is graph report or word report or other king of report.

~ Report class provide method that can get report object by primary key and type.

~ Report class provide method that can get report including the purchase created from start time to end time.

Notice

~ Created in some situation like insufficient budget, monthly/yearly report ready etc.

~ Notice will have primary key which indicate the belonging, Notice which is a string shows the words inside the Notice, time which is a int shows when the notice is sent and an int, type, to indicate the notice type.

~ Notice class provide method to send notice to one user using the user's primary key.

Data

~ Data class is the class that interact between the front-end and the back-end (database)

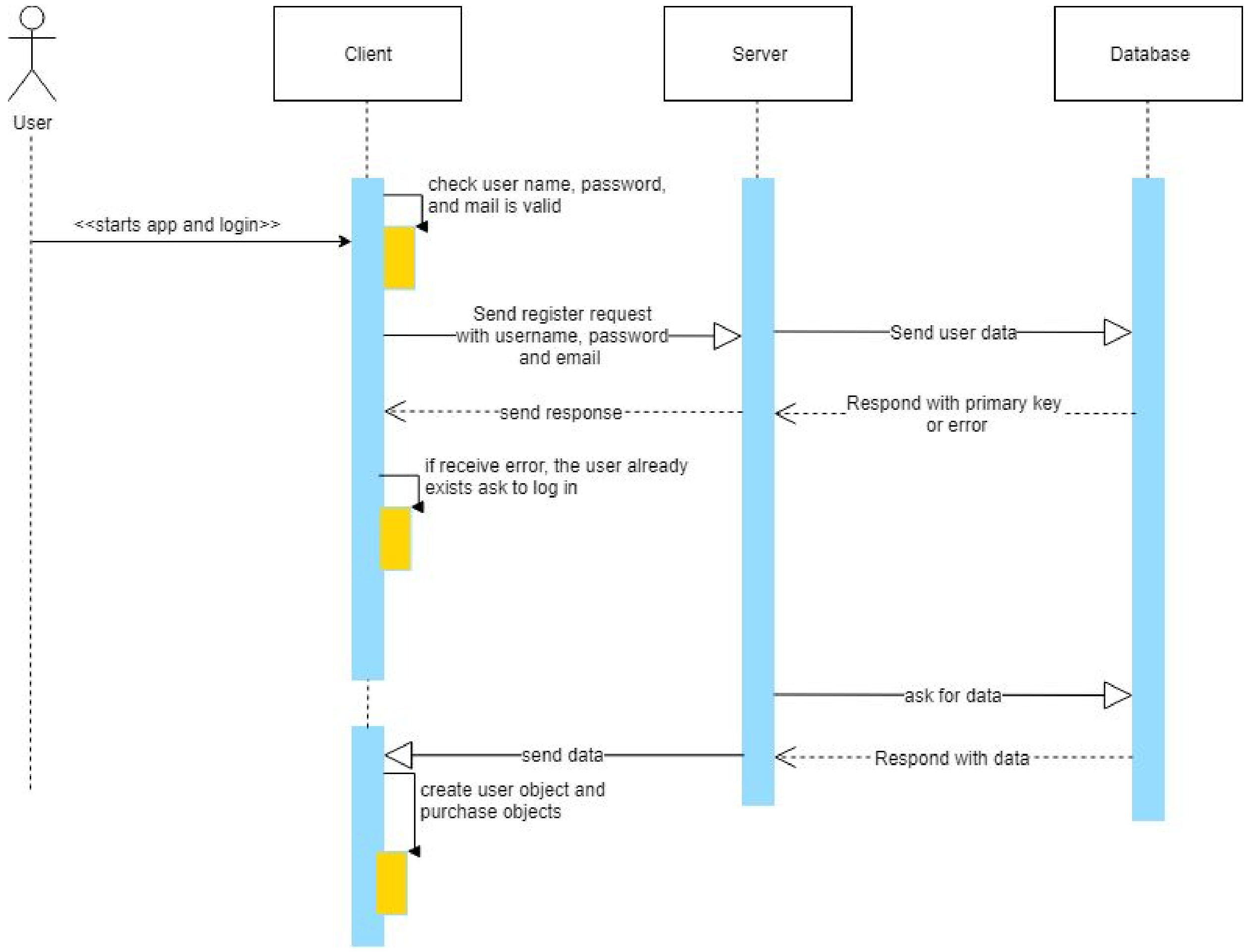
~ Data class will have User Object to hold all the user attributes and an array of purchase to hold all the purchases belong to this user.

~ Data class provide methods to send data and gain data.

Sequence Diagrams

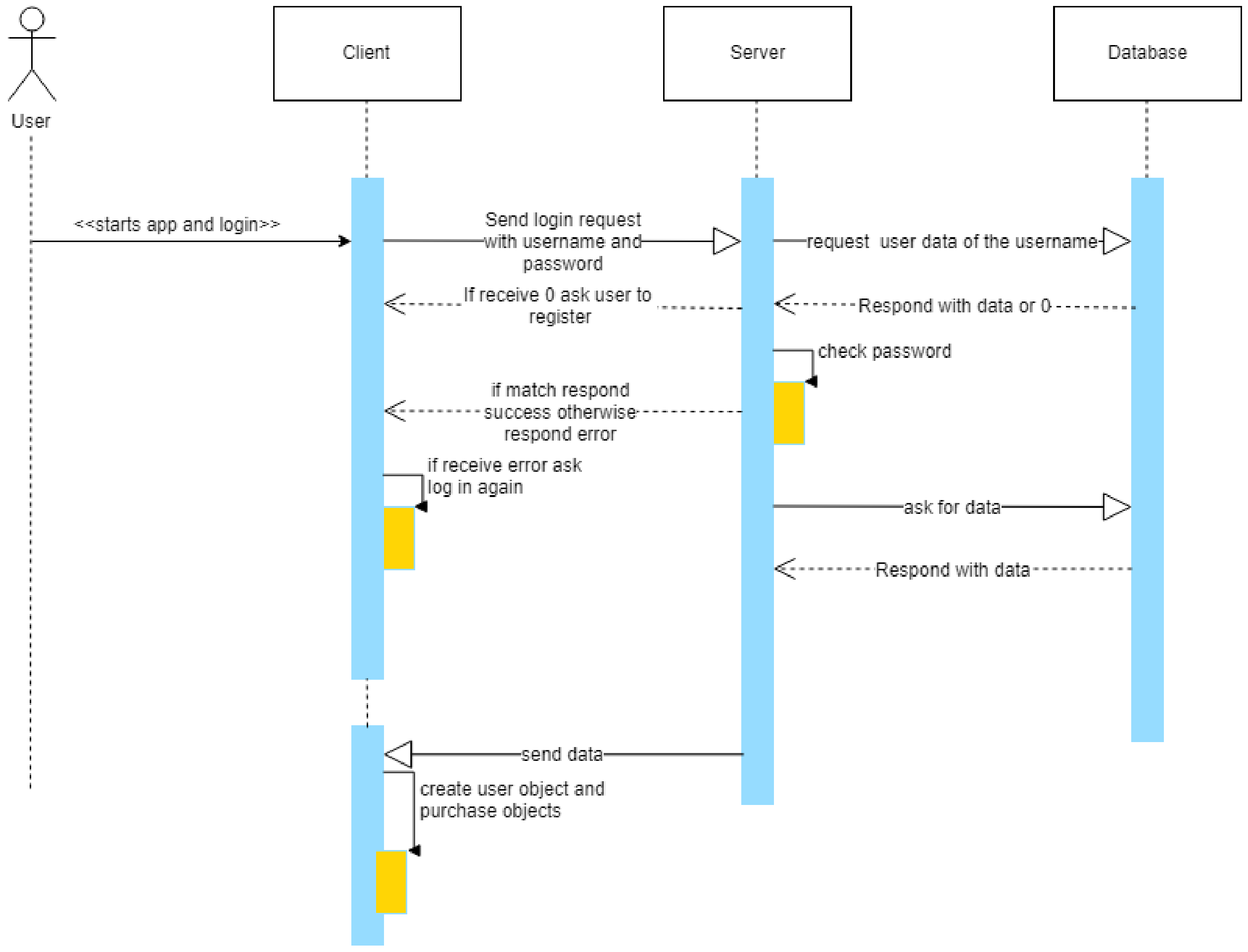
Sequence of events when the user register

When user registers, they will enter their username, password and email. Then the client will check if the username, password, and email are valid. If valid it sends the register request to the server, otherwise ask the user to enter the credentials again. When the server gets the register request it will send the user data to the database, and the database will check if the user is already use. If the user is already use send error to the server otherwise send the primary key to the server. If the Server doesn't receive error it will create an user object and then set timer to frequently update the user data. If received an error, the client will tell the user that there is an existing user with the name.


User
Client
Server
Database
check user name, password, and mail is valid
<<starts app and login>>
Send register request with username, password and email
Send user data
send response
Respond with primary key or error
if receive error, the user already exists ask to log in
ask for data
send data
Respond with data
create user object and purchase objects

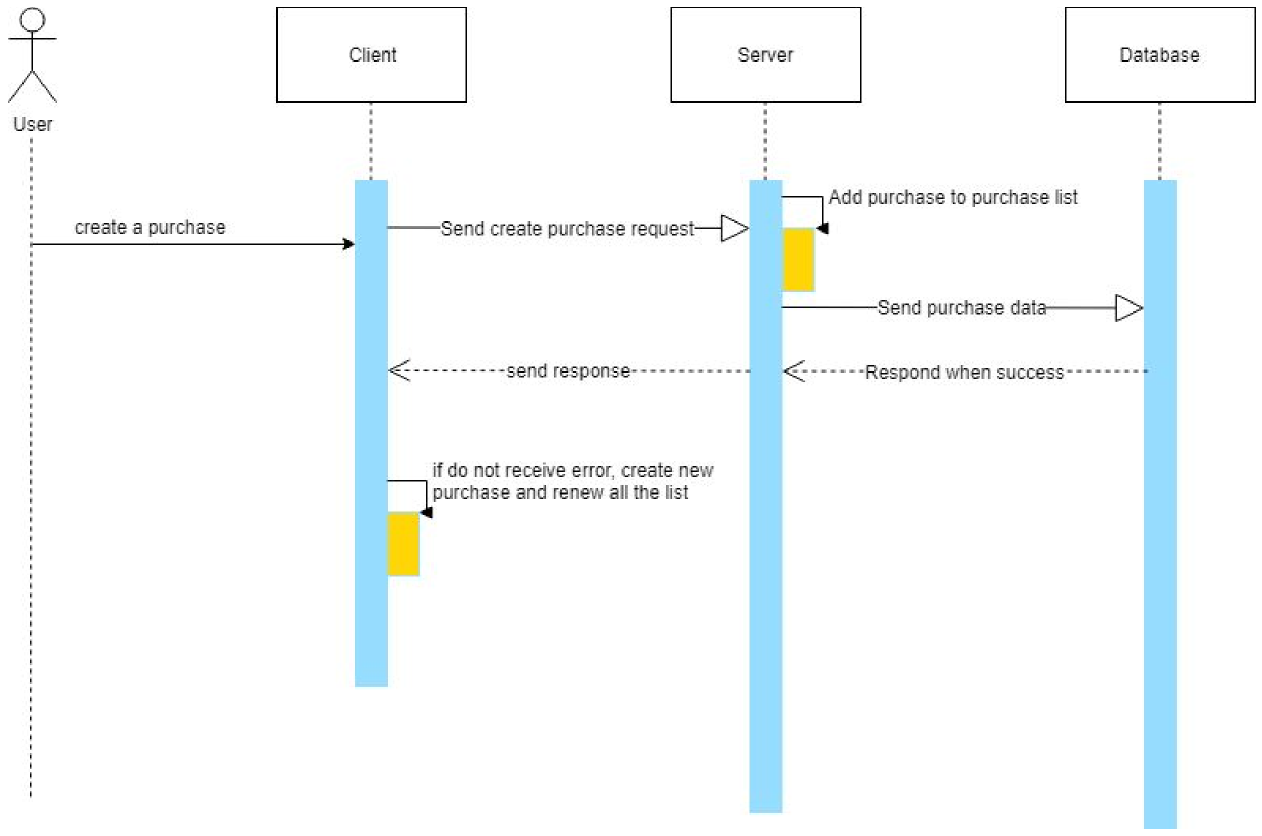
Sequence of events when users login

When the user logs in, the client will send a login request with username and password to the Server. Then the server will first check if the user is already logged in and if so, kick the other devices offline. If not, send a request to the database to gain user data. If receive 0 from database, it shows no such a user and ask the user to check their username or register. After receiving the user data check the password match if match respond success otherwise respond error to the client. Also when user login, set timer to frequently update user data.


User
Client
Server
Database
<<starts app and login>>
Send login request with username and password
request user data of the username
If receive 0 ask user to register
Respond with data or 0
check password
if match respond success otherwise respond error
if receive error ask log in again
ask for data
Respond with data
send data
create user object and purchase objects

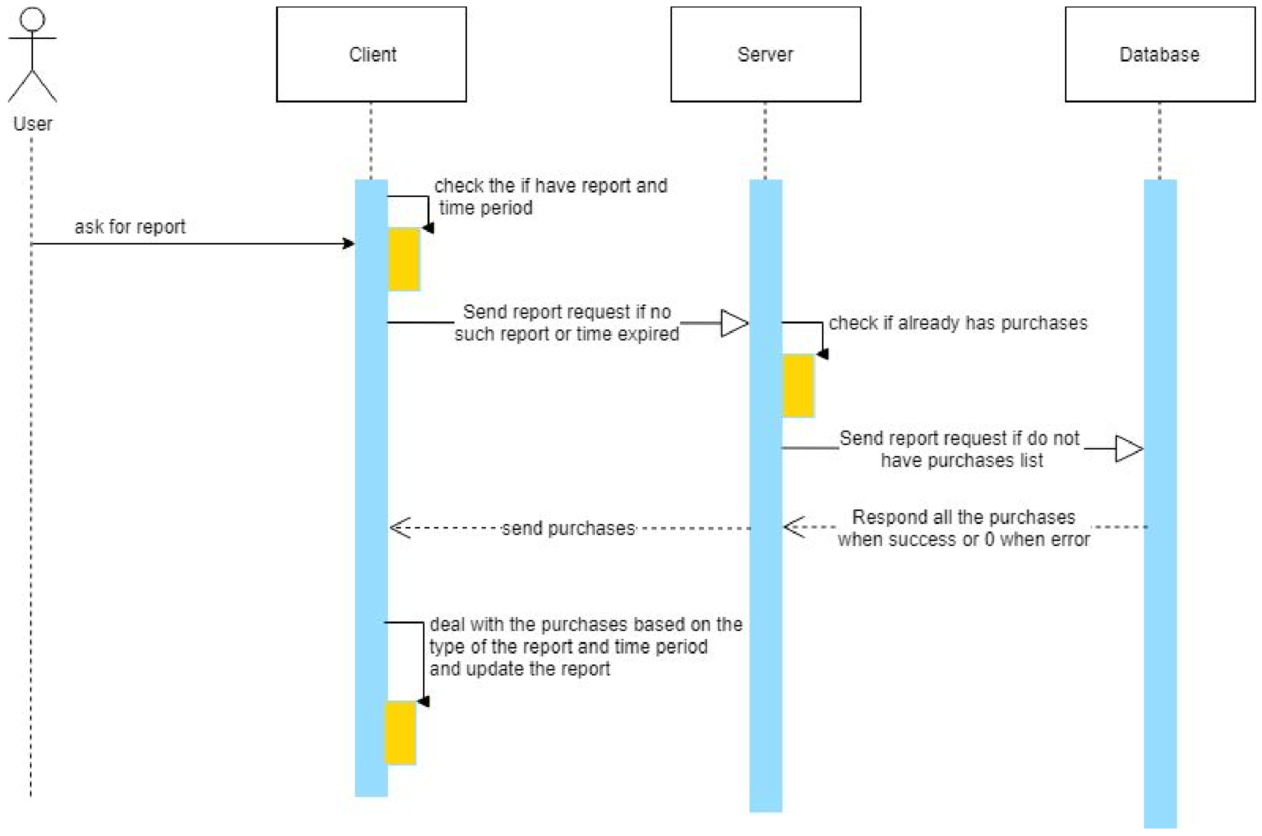
Sequence of events when users purchase item

When user purchase item, the user enter the specific information about the item then the client send the create purchase request to the server and the server will add the purchase to purchase list then send the purchase data to the database to update user's purchase list. If do not receive error, the client will create new purchase and renew all the UI relates to purchase.


User
Client
Server
Database
create a purchase
Send create purchase request
Add purchase to purchase list
Send purchase data
send response
Respond when success
if do not receive error, create new purchase and renew all the list

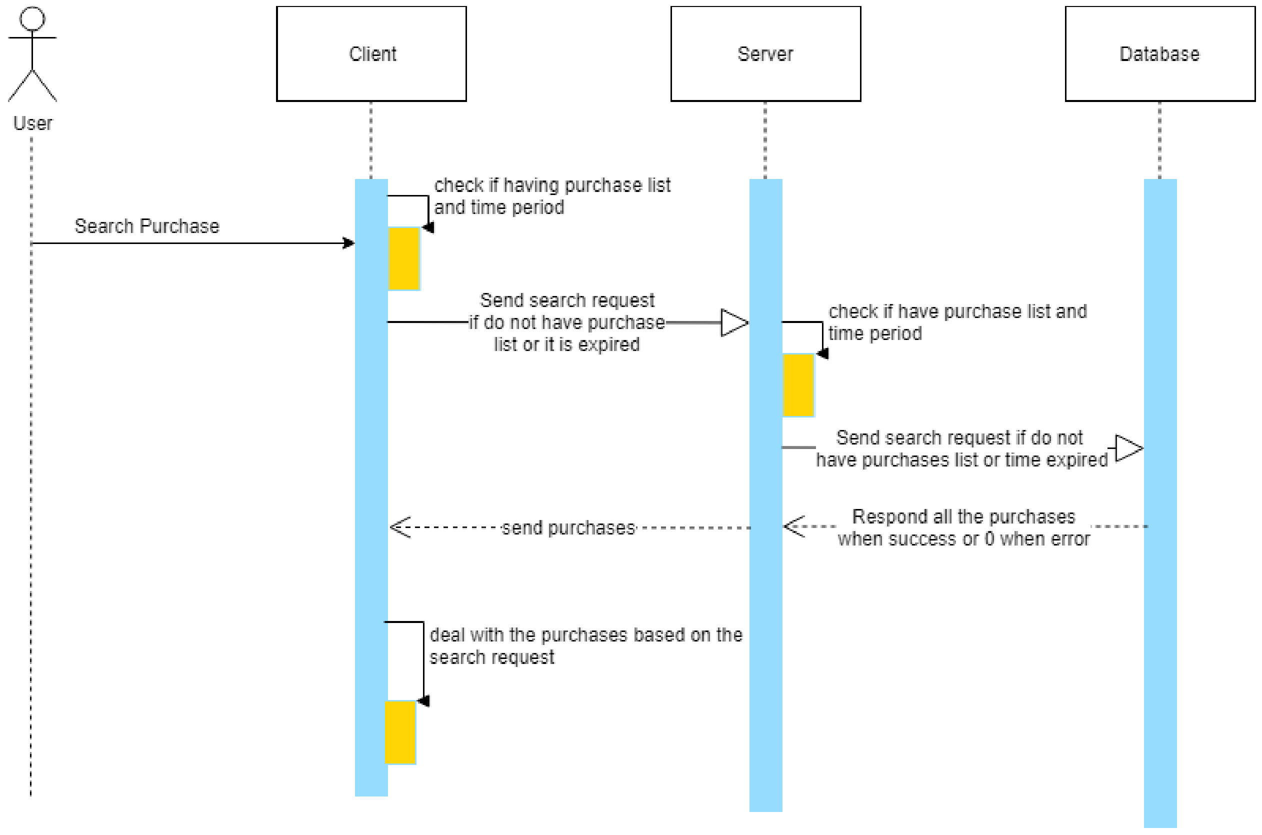
Sequence of events when users ask for report

When user asks for a report, the client will first check if the client data already has this report data and also if this report is up to date. If have the data and the data is up to date then simply use this data, and if not then send the report request to the server. The server will also first check if already has this report data and if the data is up to date. If no, then send report request to the database to gain report data and store the new report data. Then send the new report data to client to update the UI relates to this.


User
Client
Server
Database
ask for report
check the if have report and time period
Send report request if no such report or time expired
check if already has purchases
Send report request if do not have purchases list
Respond all the purchases when success or 0 when error
send purchases
deal with the purchases based on the type of the report and time period and update the report

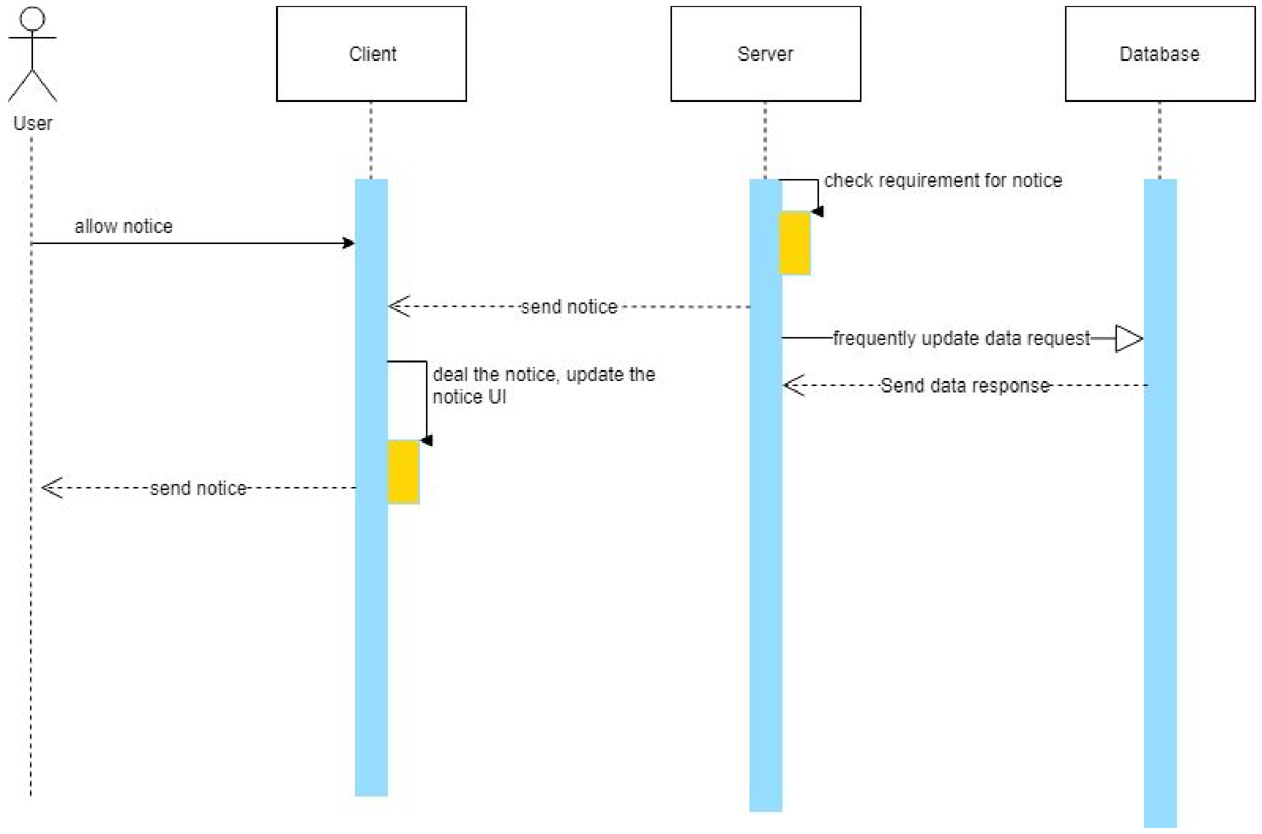
Sequence of events when users search the purchase

When user search purchase, the client will first check if already have purchase list and if the purchase list is up to date. If so, then simply use the data already has. If not send search request to the server. The server will also check if have purchase list and if the purchase list is up to date. If not send search request to the database. When server receive the data from database, send the search data to client to update the UI relates to searching.


User
Client
Server
Database
check if having purchase list and time period
Search Purchase
Send search request if do not have purchase list or it is expired
check if have purchase list and time period
Send search request if do not have purchases list or time expired
send purchases
Respond all the purchases when success or 0 when error
deal with the purchases based on the search request

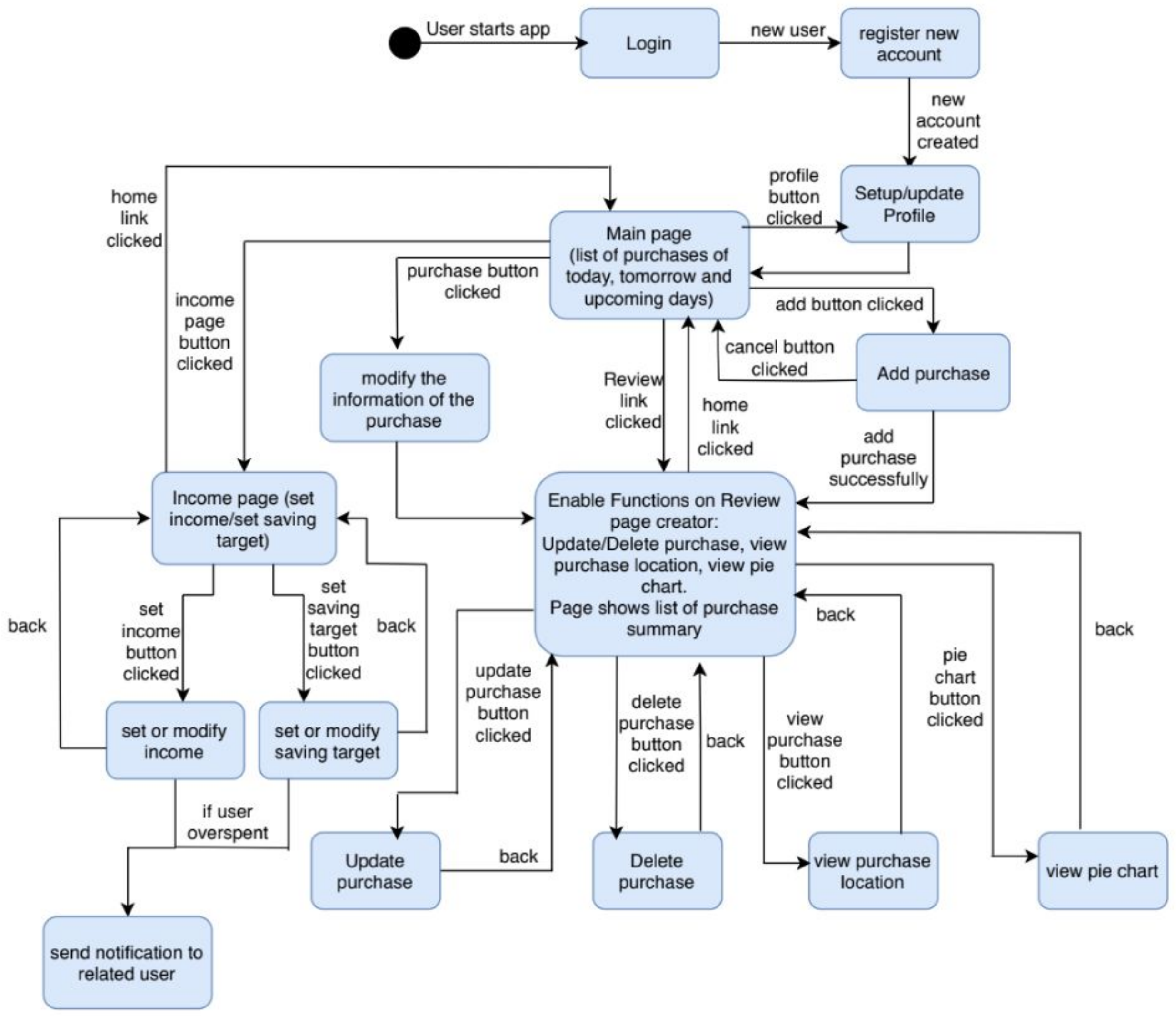
Sequence of events when send user notice

When the user allow notice in their setting, the server will frequently update data from database and send notice to client. The client will update the notice UI base on the data from server and if the user allow notice, the client will send notice to user.


User
Client
Server
Database
check requirement for notice
allow notice
send notice
frequently update data request
deal the notice, update the notice UI
Send data response
send notice

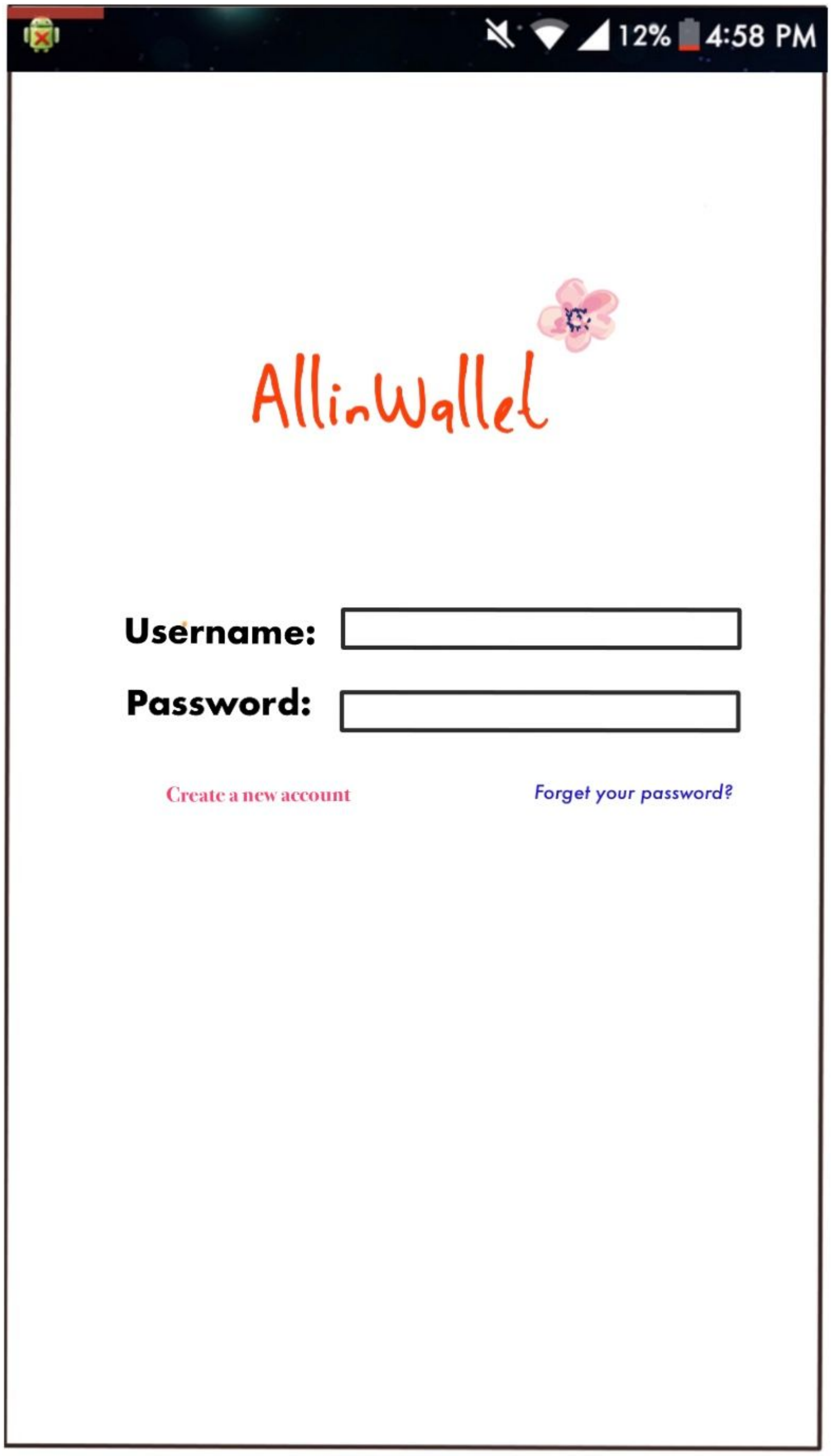
Navigation Flow Map

The design of our navigation focuses on the ease of access. We encourage our users to log into the main page with their account. If they do not have the account, they can create their own account by entering their personal information. The main page contains lists of purchases of today, tomorrow and upcoming days. Users can modify the information of certain purchases by clicking the purchase in this page. The function bar on the left side of every page contains main page, review pages, budget page, setting, contact us, etc. Users can go to any page they want by clicking the page button. The icon buttons next to the figure allows the users to update and change the figure easily.

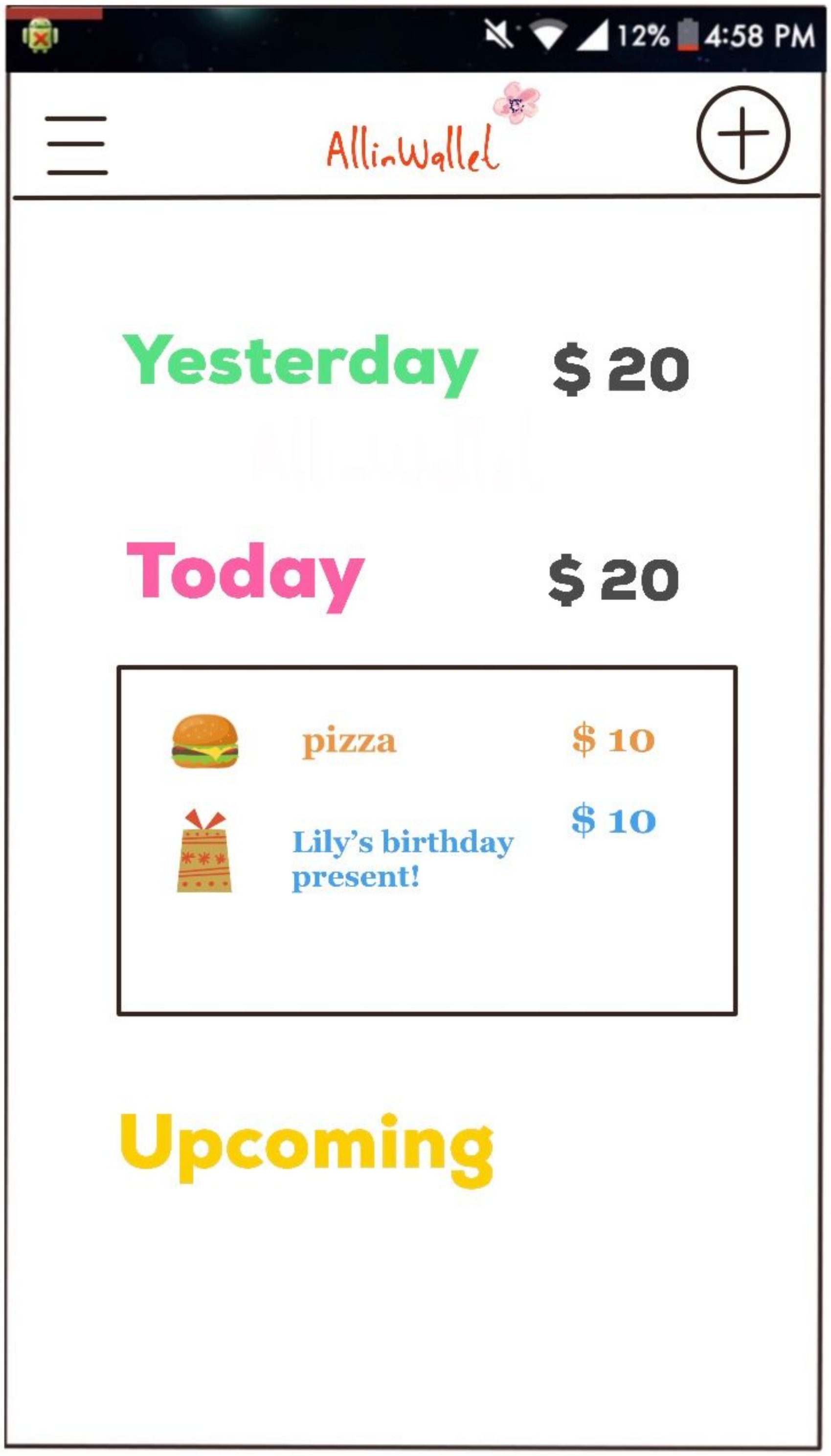

User starts app
Login
new user
register new account
new account created
Setup/update Profile
profile button clicked
home link clicked
Main page (list of purchases of today, tomorrow and upcoming days)
purchase button clicked
add button clicked
income page button clicked
modify the information of the purchase
Review link clicked
home link clicked
cancel button clicked
Add purchase
add purchase successfully
Income page (set income/set saving target)
Enable Functions on Review page creator: Update/Delete purchase, view purchase location, view pie chart. Page shows list of purchase summary
back
set income button clicked
set saving target button clicked
back
back
back
pie chart button clicked
update purchase button clicked
delete purchase button clicked
back
view purchase button clicked
set or modify income
set or modify saving target
if user overspent
Update purchase
back
Delete purchase
view purchase location
view pie chart
send notification to related user

UI Mockup

- Sample login



- Sample main page shows the spending of today, yesterday and upcoming days



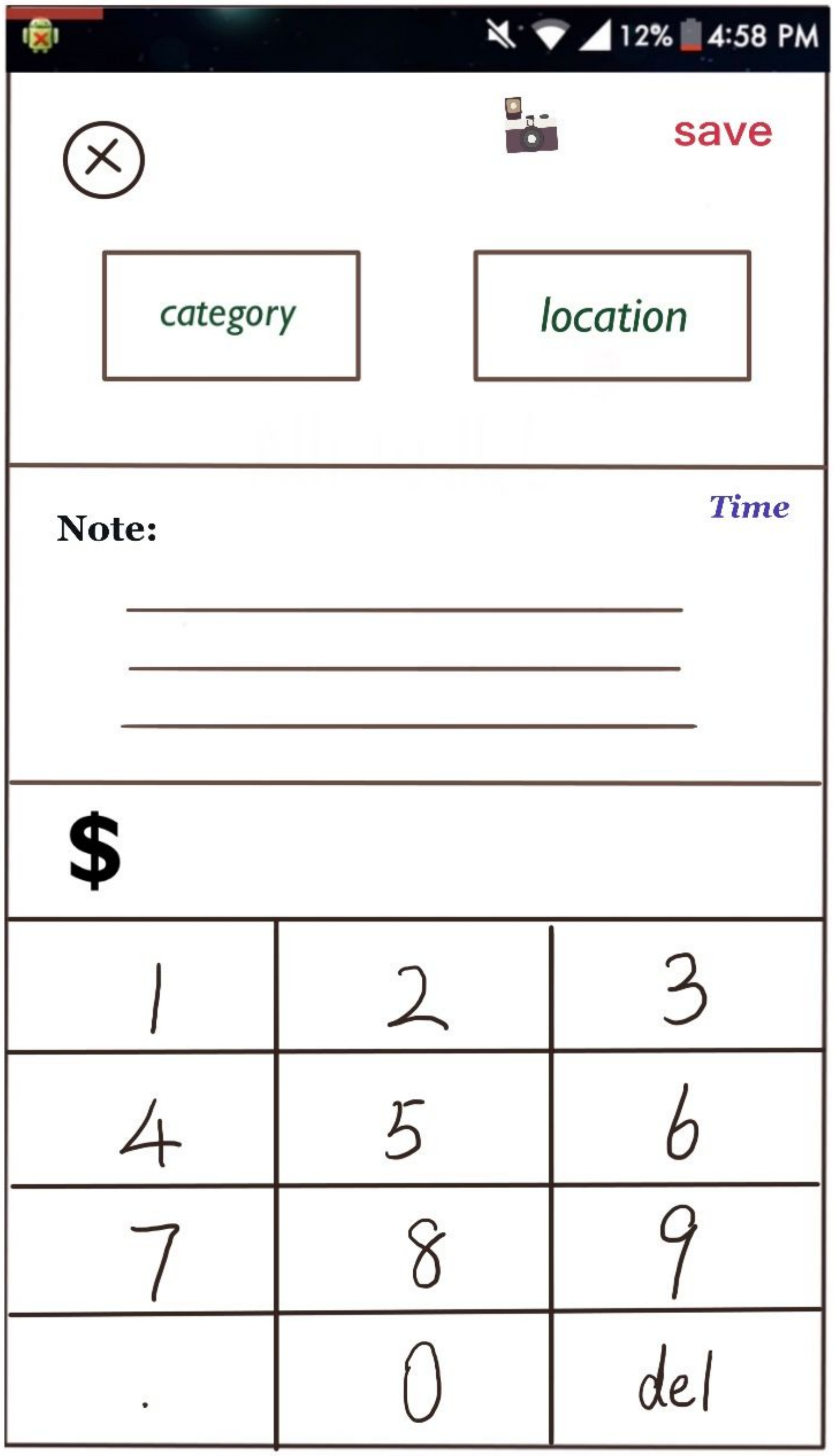
Yesterday $ 20

Today $ 20

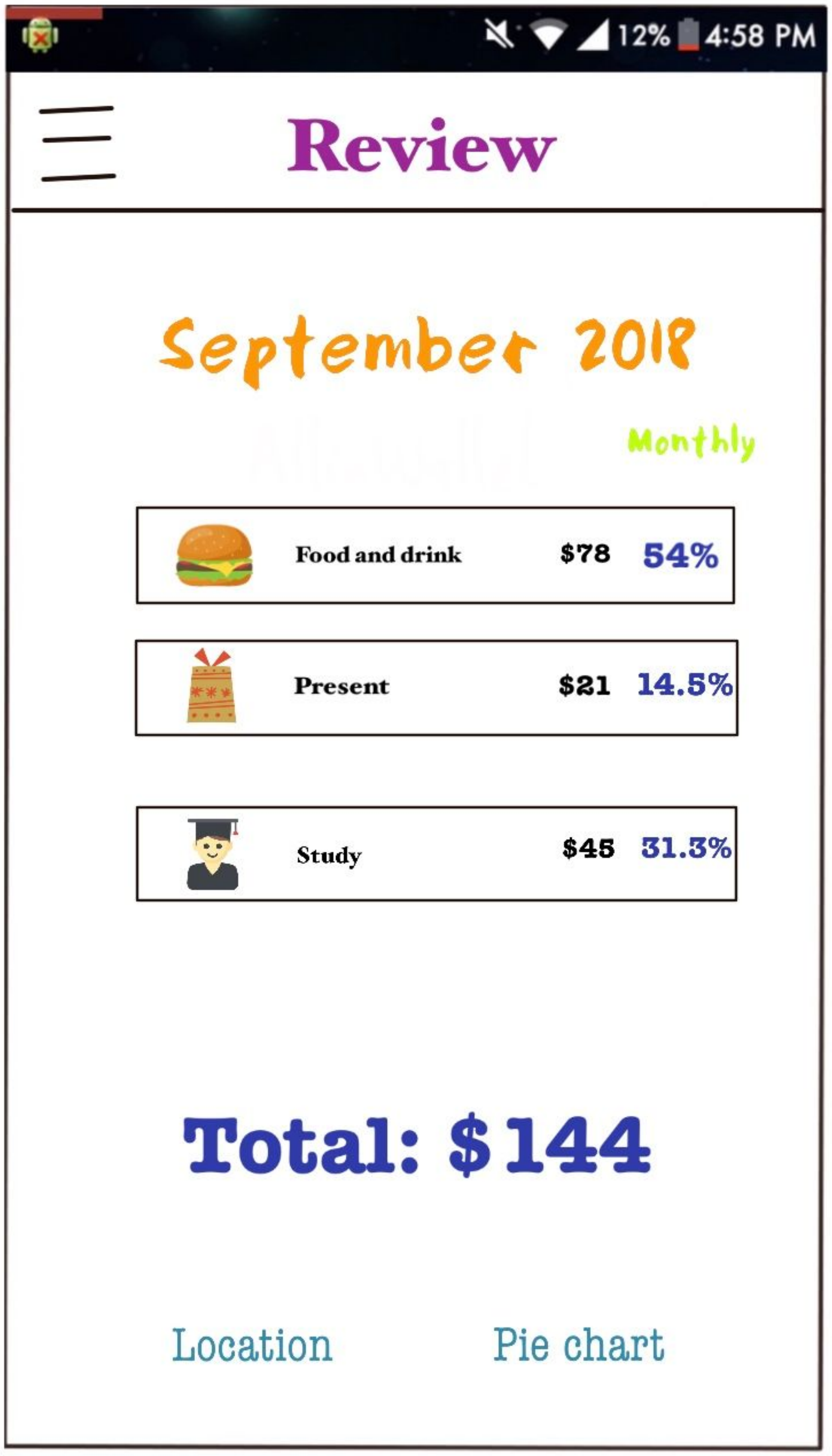
	pizza	$ 10
	Lily's birthday present!	$ 10

Upcoming

- Sample page which user can add their purchase



- Sample page which user can review their purchases of the month/week/year



Review

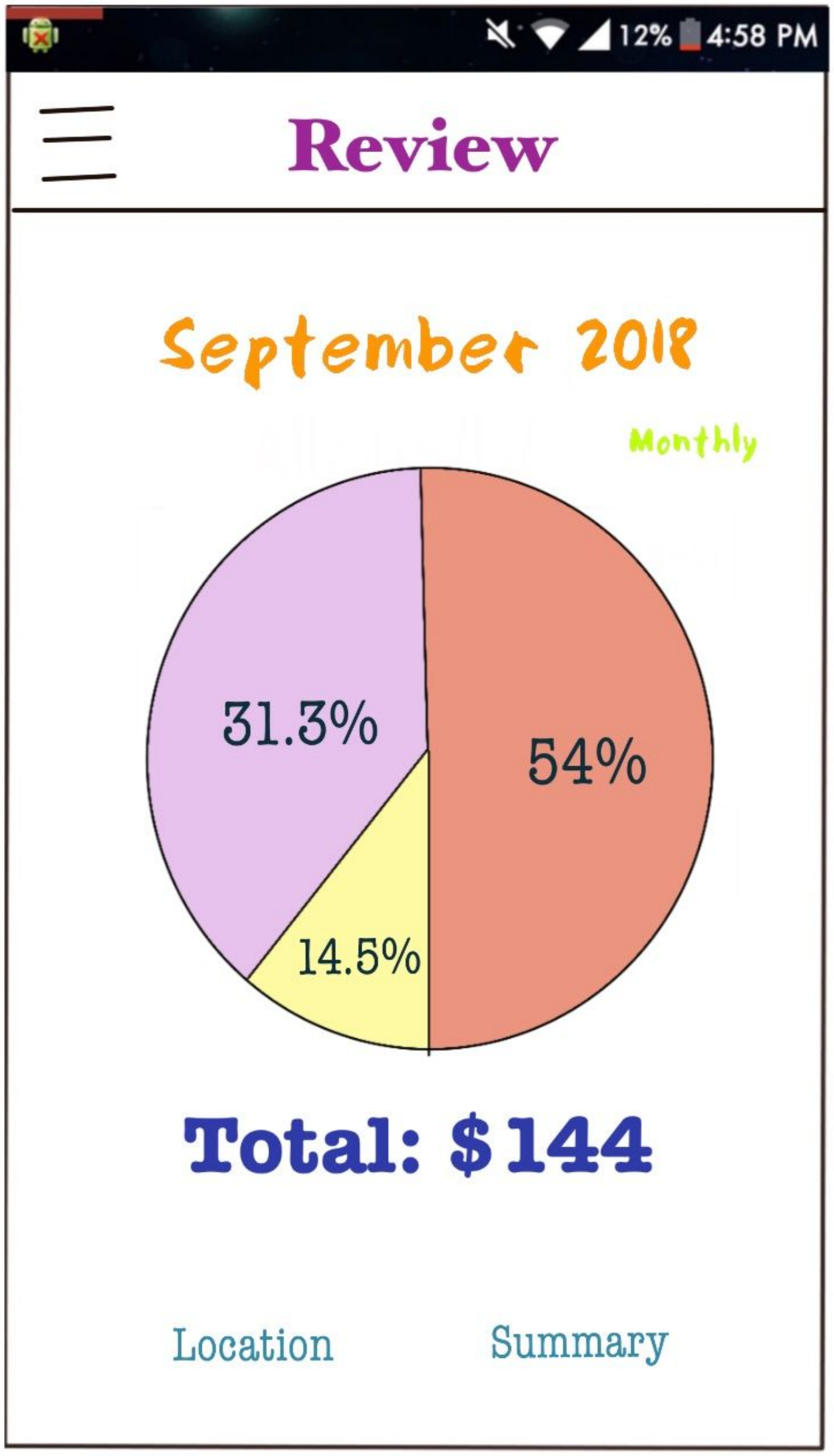
September 2018

Monthly

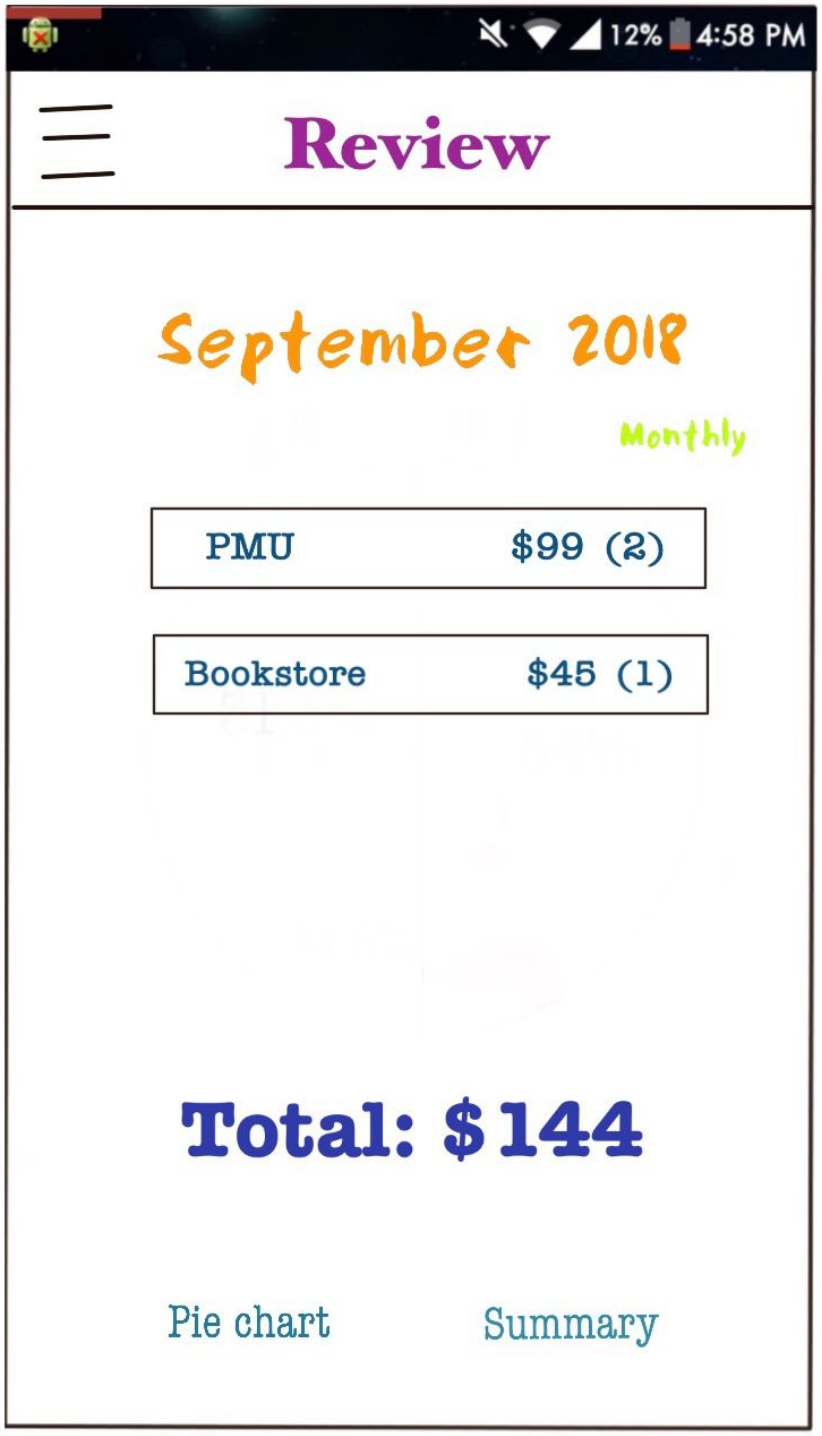
Food and drink	$78	54%
Present	$21	14.5%
Study	$45	31.3%

Total: $144

- Sample pie chart page



- Sample location page



4:58 PM

Review

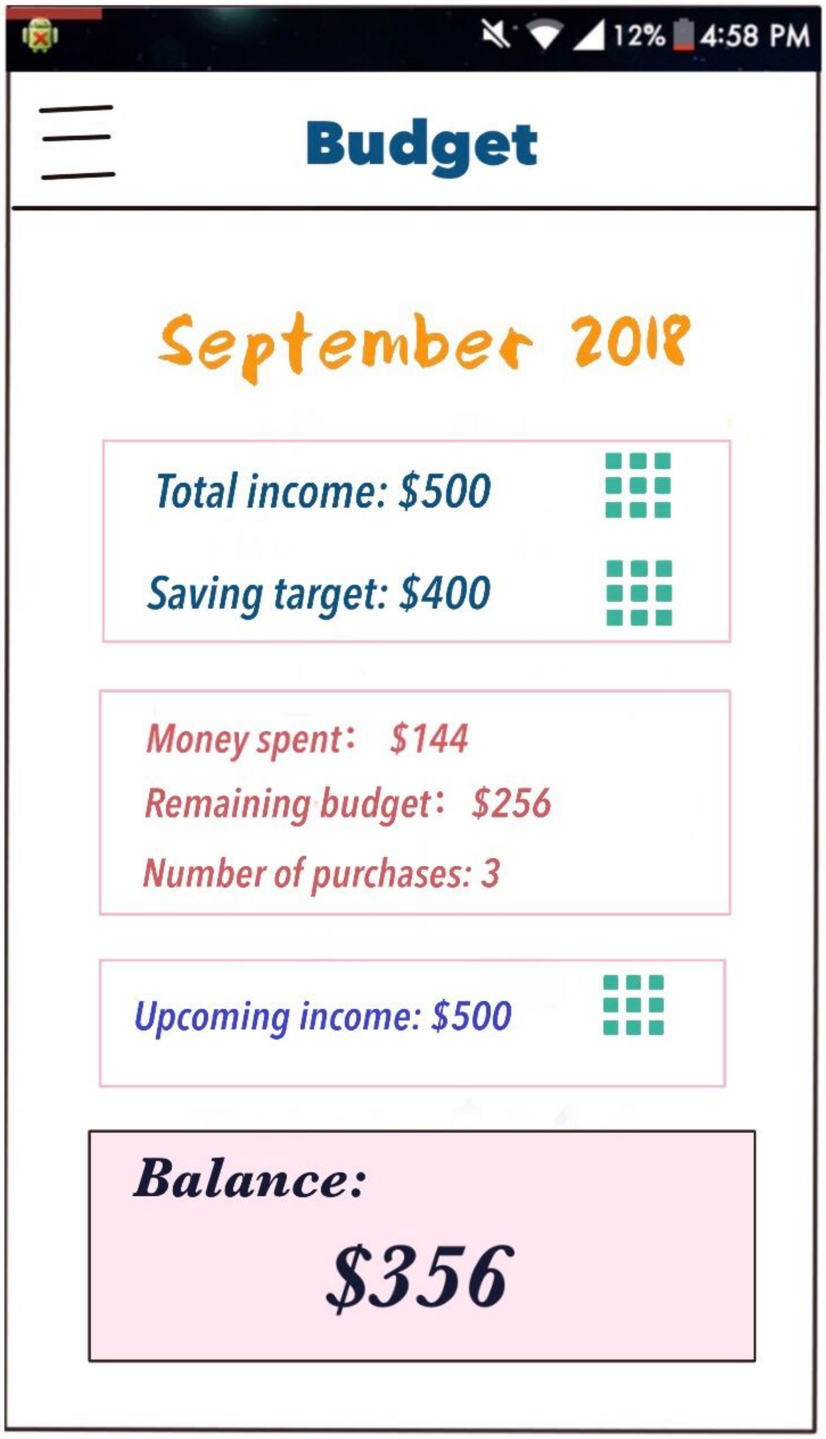
September 2018

Monthly

PMU	$99 (2)
Bookstore	$45 (1)

Total: $144

- Sample page which user can see the information about their income, budget, balance, etc



Total income: $500

Saving target: $400

Money spent: $144

Remaining budget: $256

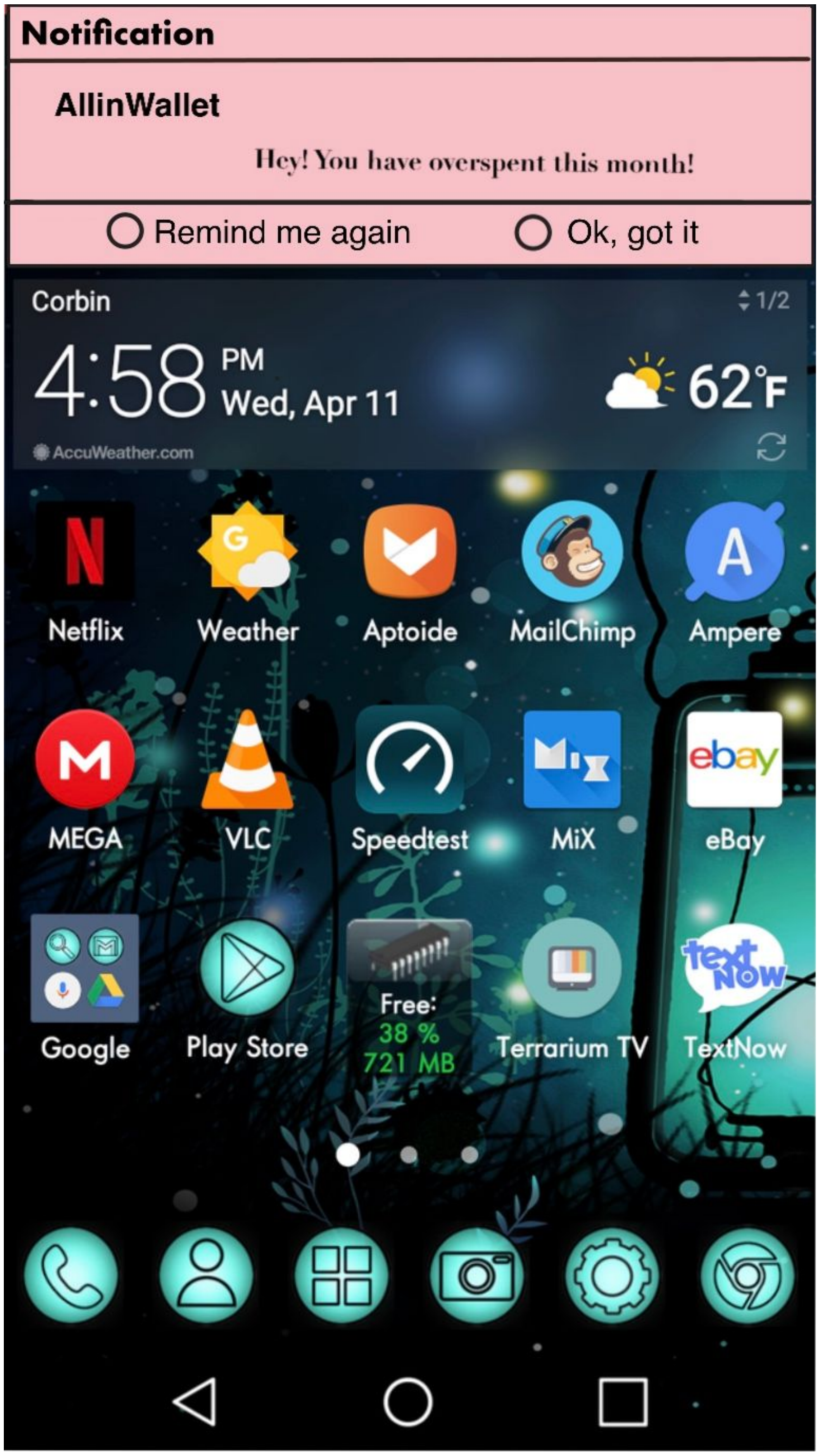
Number of purchases: 3

Upcoming income: $500

Balance:

$356

- Notification about money overspent



Notification
AllinWallet
Hey! You have overspent this month!
Remind me again
Ok, got it
Corbin
1/2
4:58 PM
Wed, Apr 11
62°F
AccuWeather.com
Netflix
Weather
Aptoide
MailChimp
Ampere
MEGA
VLC
Speedtest
MiX
eBay
Google
Play Store
Free: 38 % 721 MB
Terrarium TV
TextNow