Team 16 - Design Document

Jaineek Parikh, Zhenmu Gong, Canyu Yang, Wynn Smith, Lucas Hall, Aakash Patel

Table of Contents

Purpose

It is difficult to manage one's finances with so many methods of payment these days. A person can have multiple bank accounts, gif cards, and even have transactions in cash. Juggling all of these to keep track of how much a person has spent can get confusing and time consuming.

Expensely is a financial management app, intended to allow a user to manage all their finances and view/store their purchases in a monthly budgeting plan. The app will give feedback in the form of alerts and graphs to help a person stay on a certain budget while also being a place to keep track of all their expenses and receipts.

Functional Requirements

1) As a User, I would like to be able to create an account on the app.
2) As a User, I would like to be able to login with my Google account.
3) As a prospective User, I would like the ability to use the app anonymously and later transfer my data to an account.
4) As a User, I would like to be able to login and access my information.
5) As a User, I would like to be able to access my personalized app every time I log in.
6) As a User, I would like to be able to manage my account (changing information, deleting).
7) As a User, I would like to be able to manage my account password.
8) As a User, I would like to be able to recover my password if I forget it.
9) As a User, I would like to manually input new expenses and information about them.
10) As a User, I would like to store history for expenses so that change over time is measurable.
11) As a User, I would like to upload photos so that receipts can be stored for future access.
12) As a User, I would like to be able to view my uploaded photos.
13) As a User, I would like to be able to delete my uploaded photos when I do not want them to be on the app anymore.
14) As a User, I would like my receipt photos to add total expenses automatically through OCR so that I don't have to type in everything manually.
15) As a User, I would like to be able to see my data in graphical form so that I can easily monitor my spending.
16) As a User, I would like to be able to switch between viewing my information in multiple graph designs so that I may use whichever suits my preferences.
17) As a User, I would like to save my graphical preference so that the app remembers my choices when I log in. (If time allows)
18) As a User, I would like to be able to add scheduled additional splurges to my budget plan so that I can plan for vacations or events.
19) As a User, I would like to see quick statistics on my daily, weekly, and monthly spending so that I can track my spending history quickly and accurately.
20) As a User, I would like to be able to modify my data so that I can fix mistakes and update values.

21) As a User, I would like to be able to rate budget plans so that I can provide feedback to the community.
22) As a User, I would like to set alerts so that I can get notifications of specific payments.
23) As a User, I would like to set alerts so that I can get notifications when I am over budget for the month.
24) As a User, I would like to download template budget plans so that I may track my own expenses alongside them.
25) As a User, I would like to have access to data/budget plans specific to me so that I know they are accurate for my situation.
26) As a User, I would like to be able to store my salary range so that I can get a good estimate of my financial situation in mind of how much I make.
27) As a User, I would like to place items into categories so that I can customize my graphs.
28) As a User, I would like to create custom categories so that all my purchases may be classified regardless of size.
29) As a User, I would like to be able to have efficient customer support so that when technical difficulties arise, I know who to contact.
30) As a financially savvy user, I would like to upload my budget templates online so that others may benefit from them. (If time allows)
31) As a User, I would like to be able to customize notification frequency.
32) As a User, I would like to have the option to enable email notifications.
33) As a User, I would like to be able to receive SMS notifications.
34) As a User, I would like to have the option to receive monthly account statements.
35) As a User, I would like to be able to sort my historical data
36) As a User, I would like to search my historical data.
37) As a User, I would like to add tags to my uploaded photos.
38) As a User, I would like to be able to set the amount of on-board storage allotted for imagery.
39) As a User, I would like upload and download progress bar on my images.
40) As a User, I would like to be able to use the app at all times, even when I'm uploading or downloading something from the server (aka background processes).

Non-Functional Requirements

Performance

Our aim for this project is to produce a functioning, responsive app first and foremost, that is then scalable in terms of storage space. Our app should first display no noticeable difference between 1 user and 10 concurrent users and should have enough space to hold all their data.

At no point in our app, even for large operations, should a user have to spend time waiting for more than 12 seconds.

We will compress images before uploading them from the user's device to our server. We will put a limit on upload size and quantity of uploads to prevent a user from hogging bandwidth and abusing our service.

Usability

The UI should be intuitive and easy to use. The user should be able to access all functions using a reasonable number of taps. We will be supporting this app in English to start, but we may add more language options if we have time. We will make this app for the earliest version of Android OS that fulfills our needs for this project to provide access for more users, since Android OS versions are backwards compatible.

Security

Any sensitive information, such as salary, user name, etc. is to be encrypted both locally and in the cloud. It is only to be decrypted at the client side helping mitigate information loss in the event of a data breach. Any non-sensitive information should not be encrypted as it would impact performance with no tangible gain in security. Most of the data in the app will ultimately end up being encrypted using a secure 256-bit algorithm.

Hosting

We intend to make use of the strong inter-functionality between Android Studio and Google Cloud to serve as our back/front end connection. We will be using Firebase for a real time database and authentication system working in tandem with our android app.

Design Outline

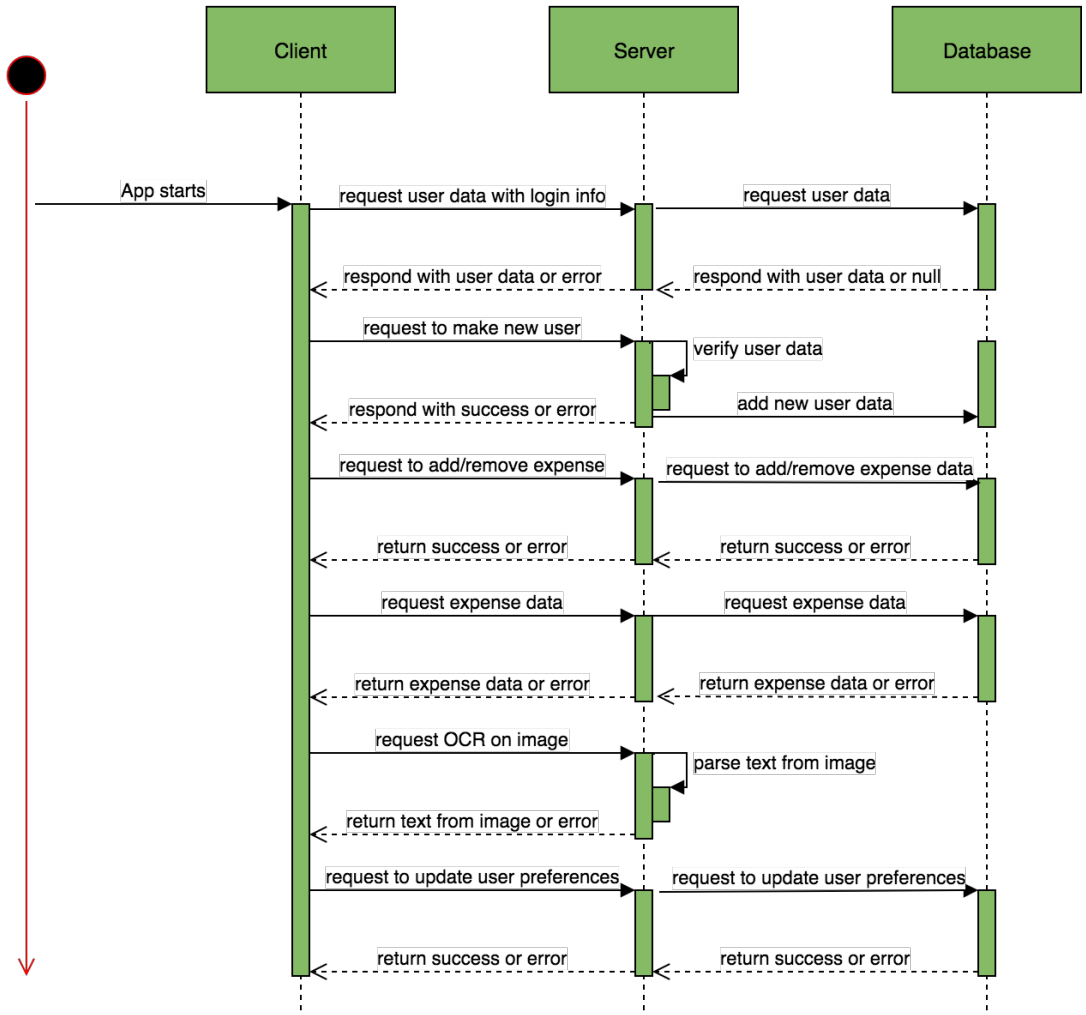
High Level Overview

This project will be an Android application that allows users to track and manage their daily expenses. This will be in the form of a client-server model where one or more users send requests to the server through Firebase API calls. The server will accept requests, manipulate data in the appropriate database, and then relay the updated information back to the client.

1) Client
    a. The client is the application that allows a user to interact with the server
    b. The client sends requests to the server through API calls.
    c. The client then receives responses from the server and displays the changes in the UI accordingly.
2) Server
    a. The server receives requests from the clients.
    b. The server verifies the requests and then sends queries to the database for manipulation of data.
    c. The server receives responses from the database, and then appropriately sends a response back to clients.
3) Database
    a. A real-time database to store any information pertaining to a user such as login information, preferences, images, and much more.
    b. The database receives queries from the server and appropriately adds, modifies, or retrieves data.
    c. The database responds back to the server with the gathered data.

Sequence of Events Overview

The sequence of events diagram shows what a typical interaction between the server, client, and database would look like. It begins when a user starts the application, or client. The client would then attempt to login. If successful, it would retrieve the user data and the client would use the received data to set the visual preferences for the application. After logged in, the client sends whatever the user does, whether that may be creating new expenses or editing the settings. These changes are relayed to the server, and then relayed to the database, which sends back a success message or an error. The sever relays this information back to the client which then updates the UI to reflect the new changes.



Design Issues

Functional Issues

1) What should we use to create new user accounts?
    - Option 1: Username and password
    - Option 2: Username and password or a Google account
    - Option 3: Username and password or a Google account or an anonymous login

    Choice: Option 3

    Justification: Giving users these extra options will make it easier to get potential new users to sign up for our application. A username and password are the baseline method of verification that allows us to send notifications to users via email. But this method is often hard to get people to sign up for as they are wary of spam if they ultimately choose not to use the application. Signing in with a Google account is a one tap method that doesn't require remembering passwords and users can control permissions via the Google security settings. This makes it a far more attractive choice and allows us to use Google's robust security to our advantage. The final choice of anonymous login is simply there to help people try out the app without any worry of making new accounts. After a certain period of time, the user will be prompted to migrate to an email account or a Google account to continue use.

2) How should we divide up timely reports?
    - Option 1: Weekly
    - Option 2: Weekly and Monthly
    - Option 3 Weekly, Monthly, and Yearly
    - Option 4: Weekly, Monthly, Yearly, and user specified time frame

    Choice: Option 3

    Justification: Weekly, monthly, and yearly reports gives enough variety to see how people are doing on their budgeting track on the short term, but also how they are doing in the long run. Any less and it will feel too restrictive. Adding a user specified time frame, such as quarterly, would introduce far too much variability and perhaps stray too far from the apps intended design. In addition, this adds needless complexity and may introduce many bugs over the course of the development cycle.

3) What tags are allowed for expenses?
    - Option 1: Predefined tags
    - Option 2: Predefined tags, and user specified tags

- Option 3: Predefined tags, user specified tags, and AI determined tags

Choice: Option 2

Justification: Predefined tags are simply not enough to encompass every sort of expense there may be. In addition, users may hold different premiums on tags and would maybe like to add a tag to their very specific and nice expense. Giving the user this ability would allow for them to organize based on their specific use case without conforming to our predefined set. Adding AI to determine tags could go very wrong as it could completely misclassify what something is categorized as. In addition, this would be time consuming within the time constraints we have and adds needless complexity.

4) How many images can be attacked per expense?
    - Option 1: Just one image
    - Option 2: One to three images
    - Option 3: An unlimited amount of images

    Choice: Option 2

    Justification: Just one image initially seems like enough, but with certain edge cases, such as an exceptionally long receipt, one image is not sufficient. One to three images gives enough range to accommodate long receipts or even multiple receipts. An unlimited number of attached images is a bad choice because it can be abused by the user. As such, one to three images is perfect for our intended use case.

5) What information should be encrypted?
    - Option 1: All data is encrypted
    - Option 2: Only user login information is encrypted
    - Option 3: Data relating to the user such as login, expenses, etc. are encrypted

    Choice: Option 3

    Justification: Although encrypting everything would be the most optimal solution, our app is going to run on mobile devices with limited CPU power and battery life. As such, constantly encrypting and decrypting all the data we're going to store is going to have a performance impact and drain the batter of the device quicker. This is suboptimal, and as such, we will only encrypt data that is sensitive to the user. That way, all sensitive data is secure whilst non-sensitive data can be stored as is for better performance

Non-Functional Issues

1) What backend service will we use?
    - Option 1: AWS

- Option 2: Google Cloud/Firebase
- Option 3: Azure

Choice: Option 2

Justification: As we only have a limited amount of time to build our app, we want to spend that time baking functionality into our app and less time worrying about the actual backend part. Google Cloud and Firebase abstracts a lot of the difficulty in building a backend and allows us to focus on our application alone. In addition, it integrates perfectly with Android Studio.

2) What language should we use?
    - Option 1: Java
    - Option 2: Kotlin

    Choice: Option 1

    Justification: As a few of us in our group are inexperienced Android developers, we want to maintain the lowest barrier to entry possible. All members of the group are experienced with Java which would lead us easily into learning Android development. Learning something new like Kotlin, even if it is an interesting language, would delay our development and potentially cause problems with inexperience in the future.

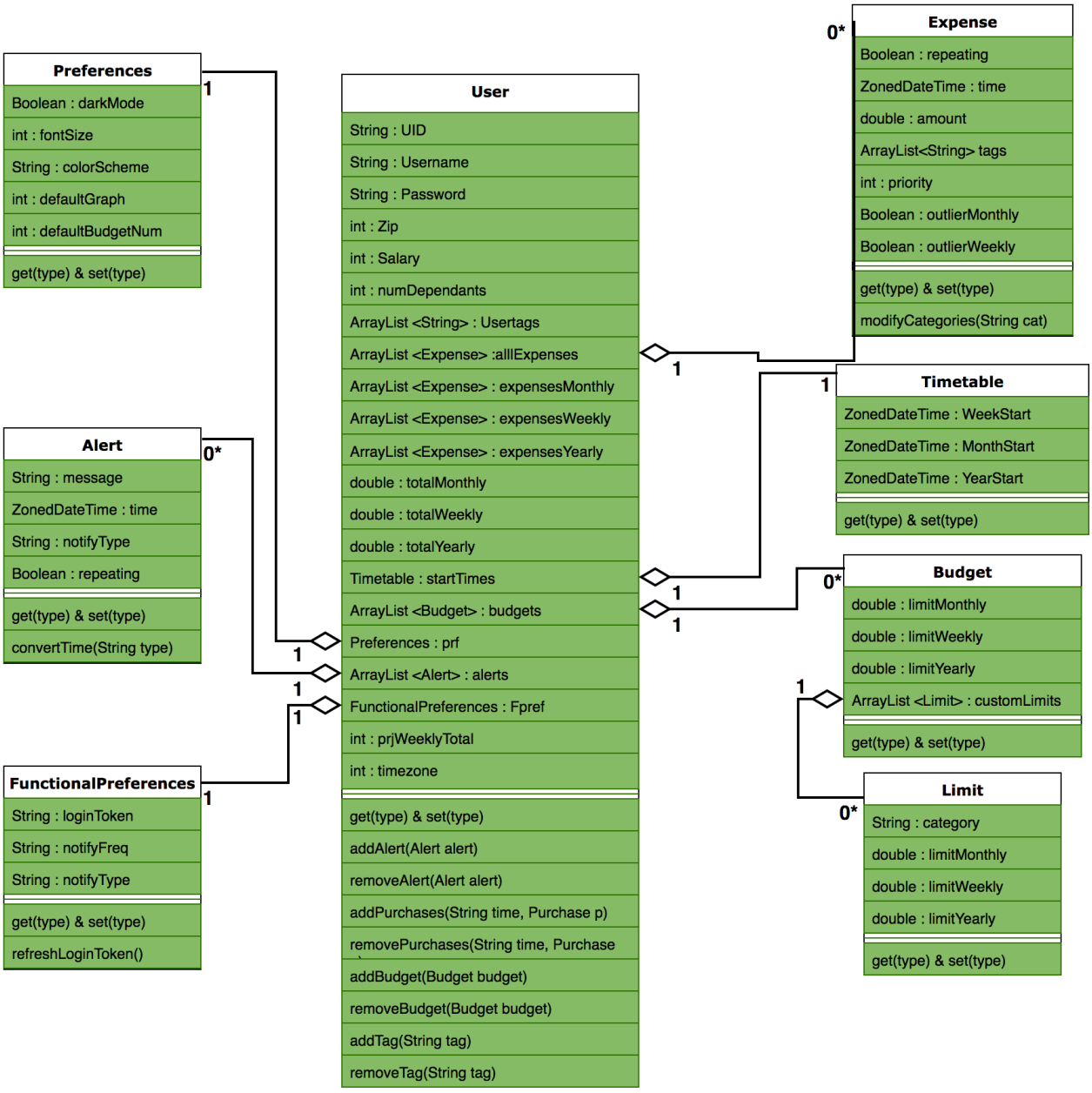
3) What design should our UI follow?
    - Option 1: Material and minimalist design
    - Option 2: Design with a lot of options and information

    Choice: Option 1

    Justification: Although it would be nice to offer the user a lot of options and information, this can be sensory overload. If we keep our design simply while following the material design guidelines, we can create an aesthetically pleasing app that conveys everything we need it to. In addition, it is easier to take into account things like color blindness when there are far fewer design elements.

Design Details

Class Design



Description of Data Classes and Their Interactions

These classes are designed based on what objects we will need for our app to function. Each class has attributes needed for app functionality.

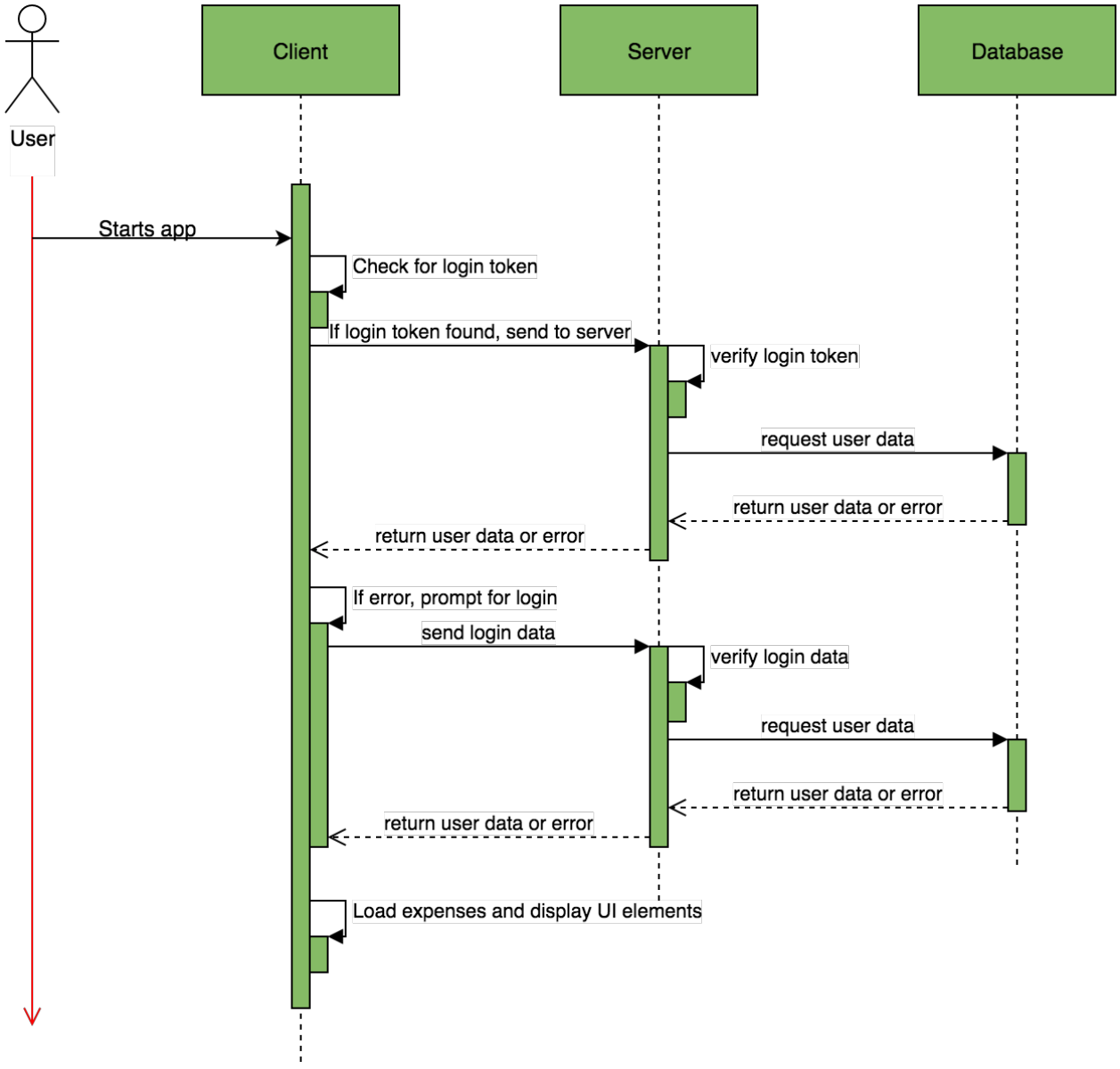
- **User**
    - A user object is created when someone first signs up.
    - Each user has a unique UID used for identification.
    - Each user has an email, password, and a login token used for logging in.
    - Each user has a zip code, salary, and number of dependents used to determine financial bracket.
    - Each user will have a list of budgets that are used to determine the constraints of the graph.
    - Each user has a list of tags where all default and user defined tags will be stored.
    - Each user has a number of lists of expenses where their expense objects are stored. These are displayed in a menu in descending chronological order.
    - Each user has integers that keep track of how much money they've spent thus far. These will be displayed in various graphs.
    - Each user has a timetable used to keep track of how much time has passed for alerts.
    - Each user has a non-functional Preferences object used to keep track of visual preferences of the app.
    - Each user has a functional Preferences object used to keep track of preferences that affect functionality.
    - Each user also has a list of alerts used to keep track of what alerts are currently active.
- **Expense**
    - An expense object is created when the user clicks the 'New Expense' button.
    - Each expense contains a boolean that keeps track of whether or not the object is repeating.
    - Each expense contains a time object at which the expense was created.
    - Each expense has a list of tags which keeps track of the applicable tags.
    - Each expense has an integer which represents the expenses priority.
    - Each expense has two booleans that determine if the expense is an outlier and should be ignored in graphs and budgeting advice.
- **Timetable**
    - Each timetable object contains a start time of each week, month, and year. This
    - is used to determine when a time frame ends.
    - The timetable is updated once a time frame ends and a new one begins.

- **Budget**
    - Each budget object has limits for weekly, monthly, and yearly spending.
    - Each budget object has a list of custom user-specified limits for each tag
    - These are used for constraints within the graph and are used when alerts need to be sent for overspending.
- **Limit**
    - Each limit object has a string that identifies which tag is to be limited.
    - Each limit has weekly, monthly, and yearly limits used to define constraints.
- **Preferences**
    - Each preference has a boolean that determines whether or not the user wants dark mode.
    - Each preference has an integer that determines their preferred font size.
    - Each preference has an attribute that determines the accent color of the app.
    - Each preference has attributes that determine which graph and budget are to be displayed by default.
- **Alert**
    - Each alert has a message that is meant to be conveyed to the user.
    - Each alert has a time at which the alert is meant to go off.
    - Each alert has an attribute that determines whether the alert is going to be emailed and/or texted to the user.
    - Each alert has a boolean that determines whether or not it is going to be repeated in the future.
- **Functional Preferences**
    - Each functional preference object has a string that denoted the login token. This token is used to login automatically and expires after a certain amount of time.
    - Each functional preference object has strings that denote user preference for how often they wish to be notified about alerts.
    - Each functional preference object has strings that denote whether or not they want to be emailed and/or texted alerts.

Sequence Diagrams

These diagrams show the events that the application will follow when the user initiates a certain request. It depicts how the user interacts with the UI, and which request will be sent to the server. The server validates and then processes the request by interacting with the database. The answered query is relayed back to the app where the UI is updated to display the changes.

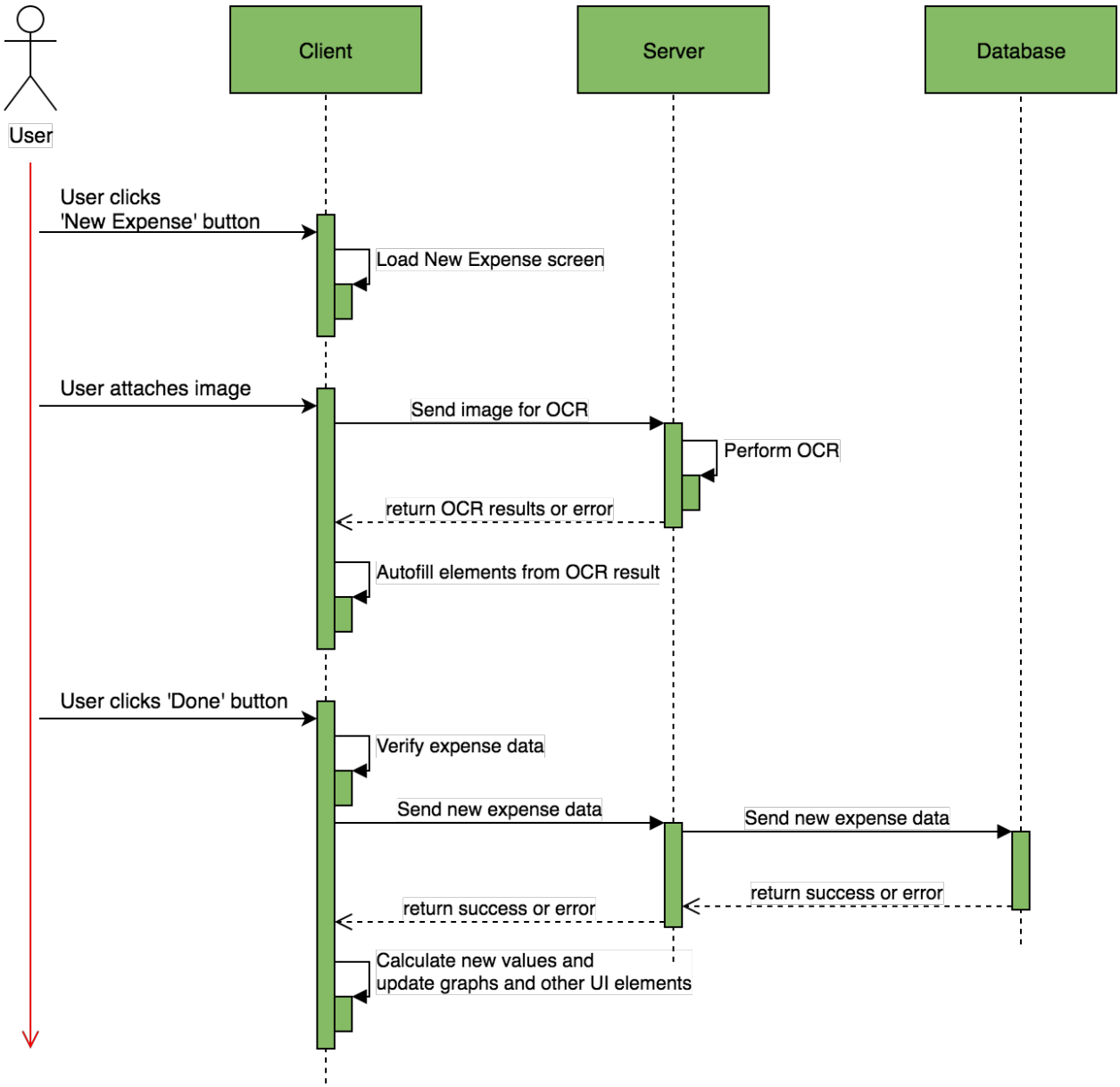
1) Sequence of events when a user requests to login



When a user opens the app, the client first checks if there is a stored login token. If there is, then it sends that to the server to login. The server verifies the validity of the

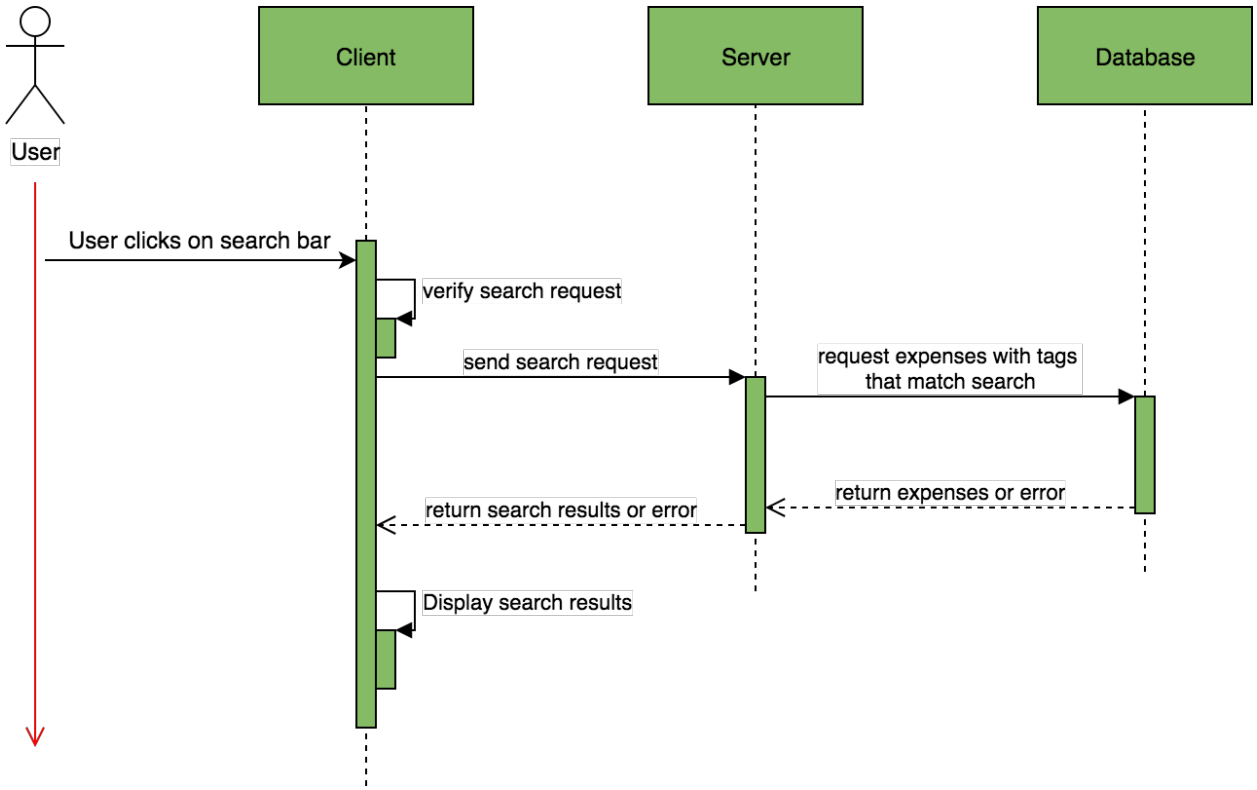
token, and either sends a query for the user data or sends an error back to the client. If there was an error with the login token, the client updates the UI to prompt the user for their username and password. The client then sends this information to the server to login which then checks the validity of the login information. If correct, it sends a query to the database for the user information, otherwise it sends an error back to the client. At the end, the client loads all of the expense data and displays it on the UI.

2) Sequence of events when a user requests a new expense



When a user clicks the ‘New Expense’ button, the New Expense activity screen is loaded and displayed. If they choose to attach an image, we perform OCR on that image and autofill the elements with what text has been extracted. The user can then add more information or delete what has been auto filled until they are satisfied. When the user clicks ‘Done’, the information they’ve inputted is verified for format. If the formatting is correct, then the expense data is sent to the server to be stored. The server then queries the database to store the new expense. The database either succeeds or errors and the server relays this information back to the client. New values and graphs are calculated and then displayed on the UI.

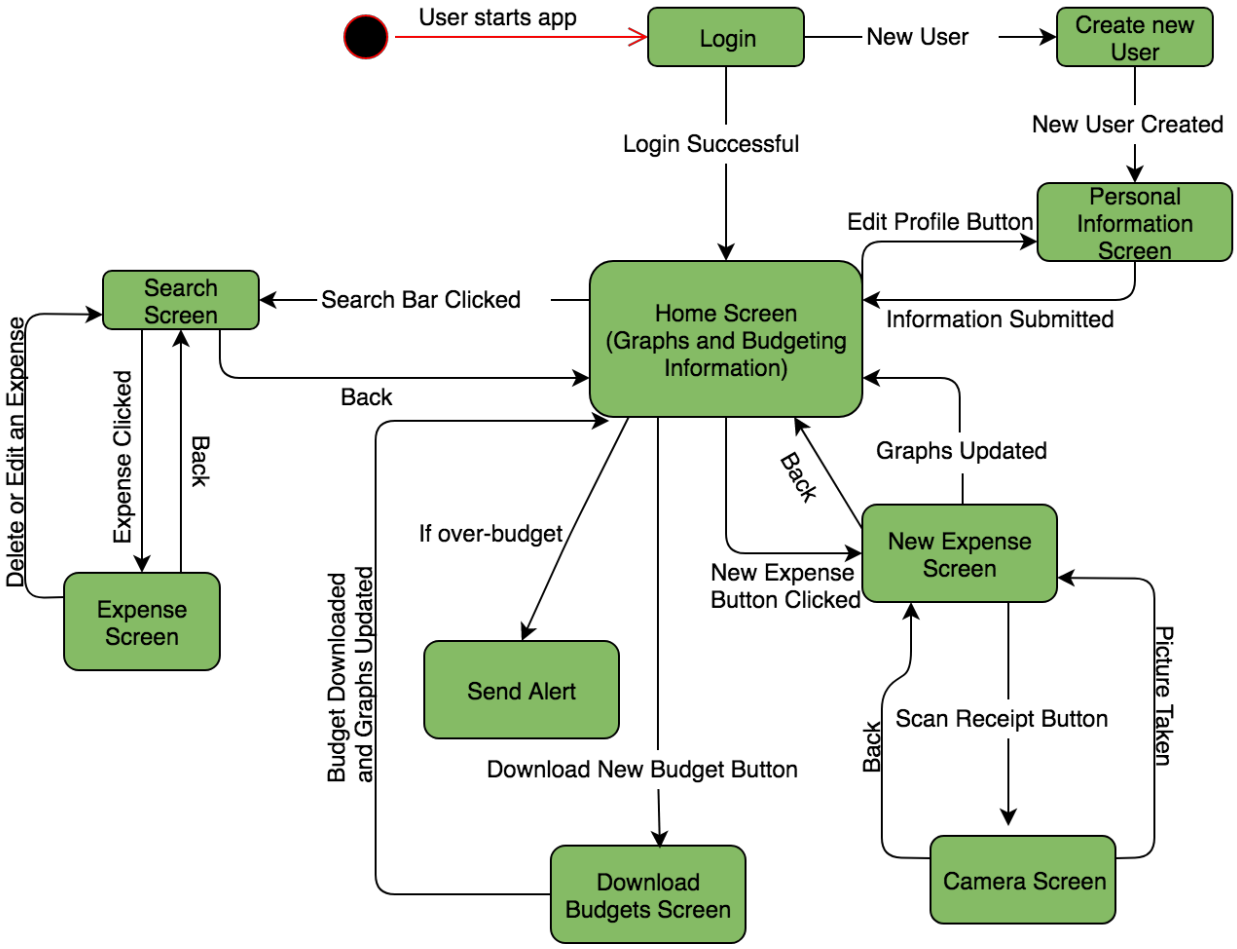
3) Sequence of events when a user requests a search



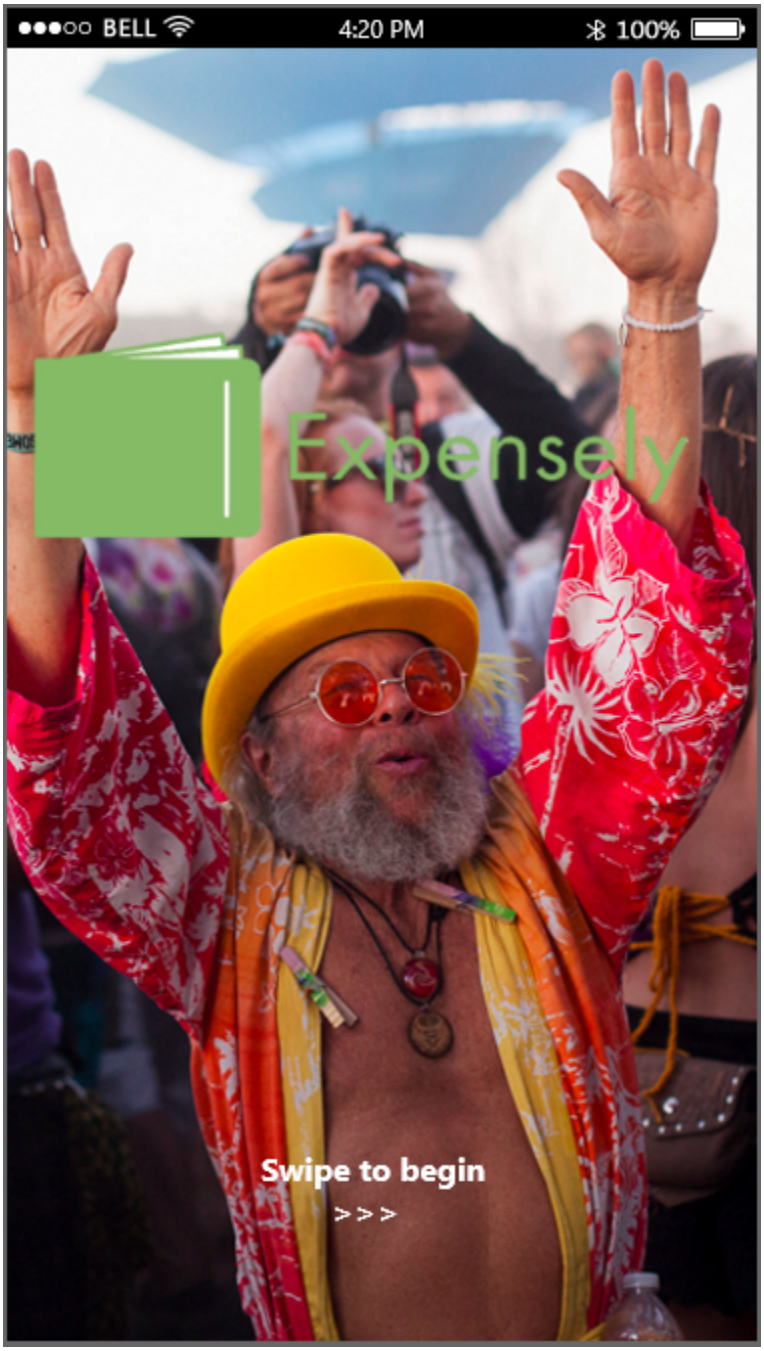
When a user clicks on the search bar, they input their search request. This request is verified to make sure that it will not break the sorting and searching algorithms. It is then sent to the server. The server then requests expenses with tags that match the search. The database sends back those expenses which the server then relays back to the client. The client updates the search page with the recently received query.

Navigation Flow Map

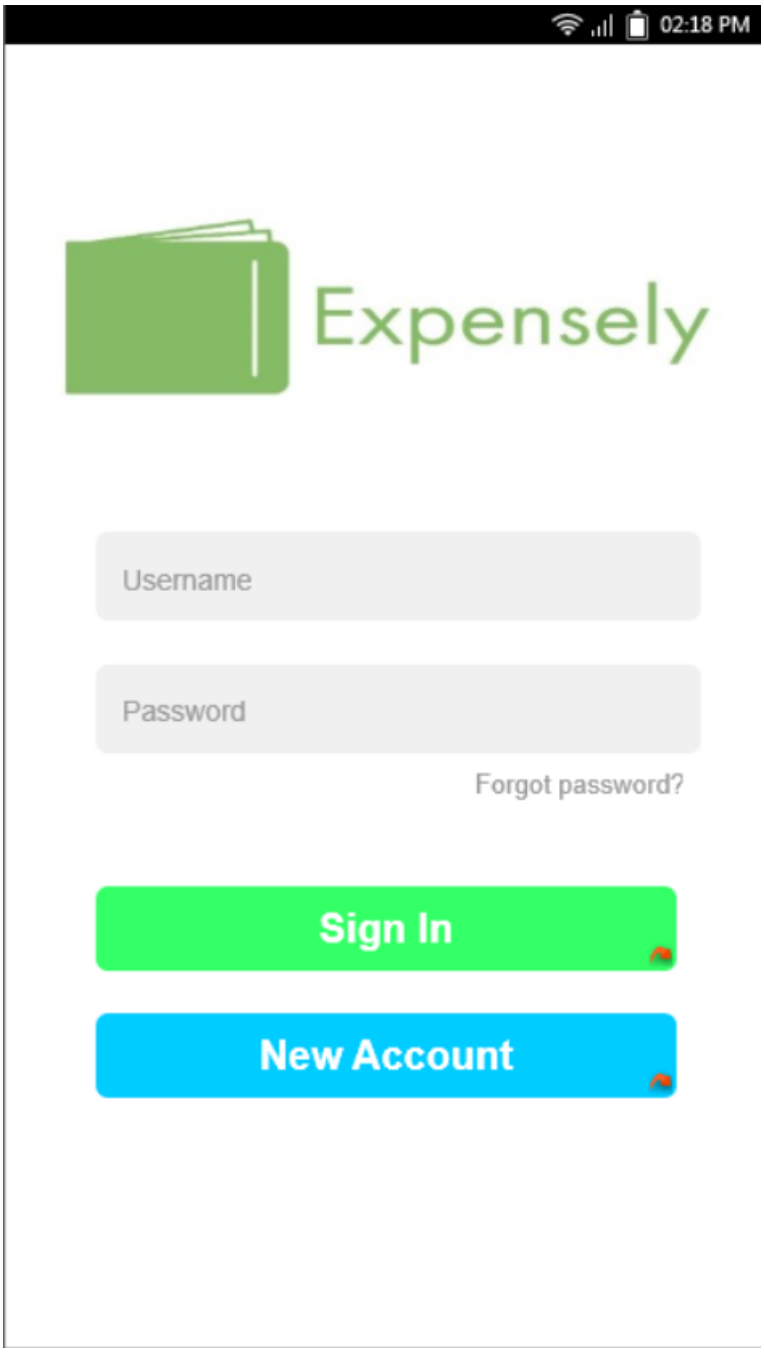
Our design places an emphasis on the materialist design and using the least amount of states to get complete an action. When a user first starts up, they either login or create a new account. If it is the latter, then they also input pertinent financial information about themselves. Then they are sent to the home screen where most of the relevant information is shown for consumption. From the home screen, there are three places you can go: the new expense screen, the download budgets screen, or the search screen. And from those three screens, you can do everything our app allows.



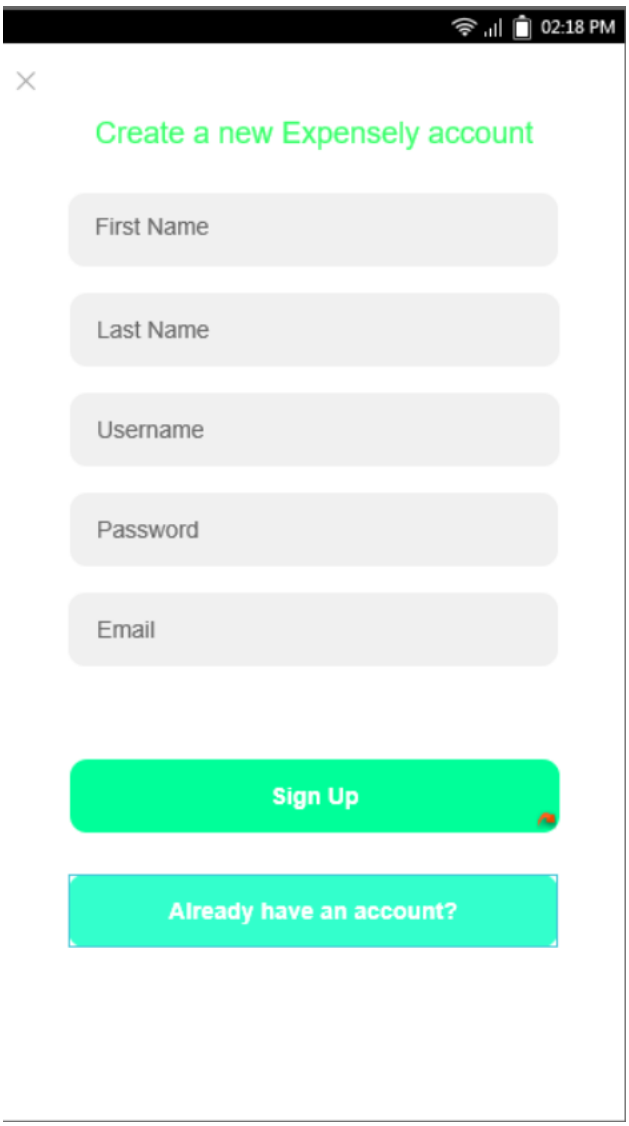
UI Mockup



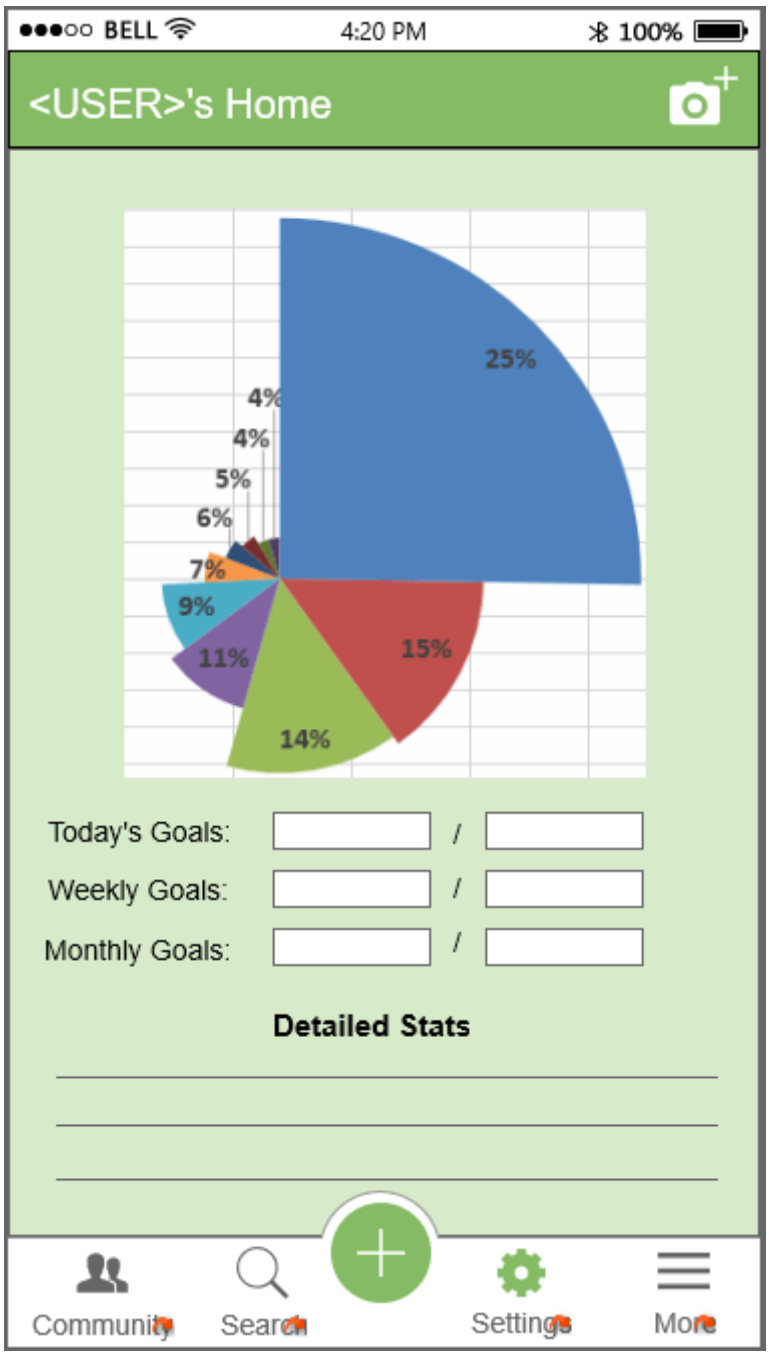
This is the screen that greets the user. It Is simple and has an upbeat tone to it.



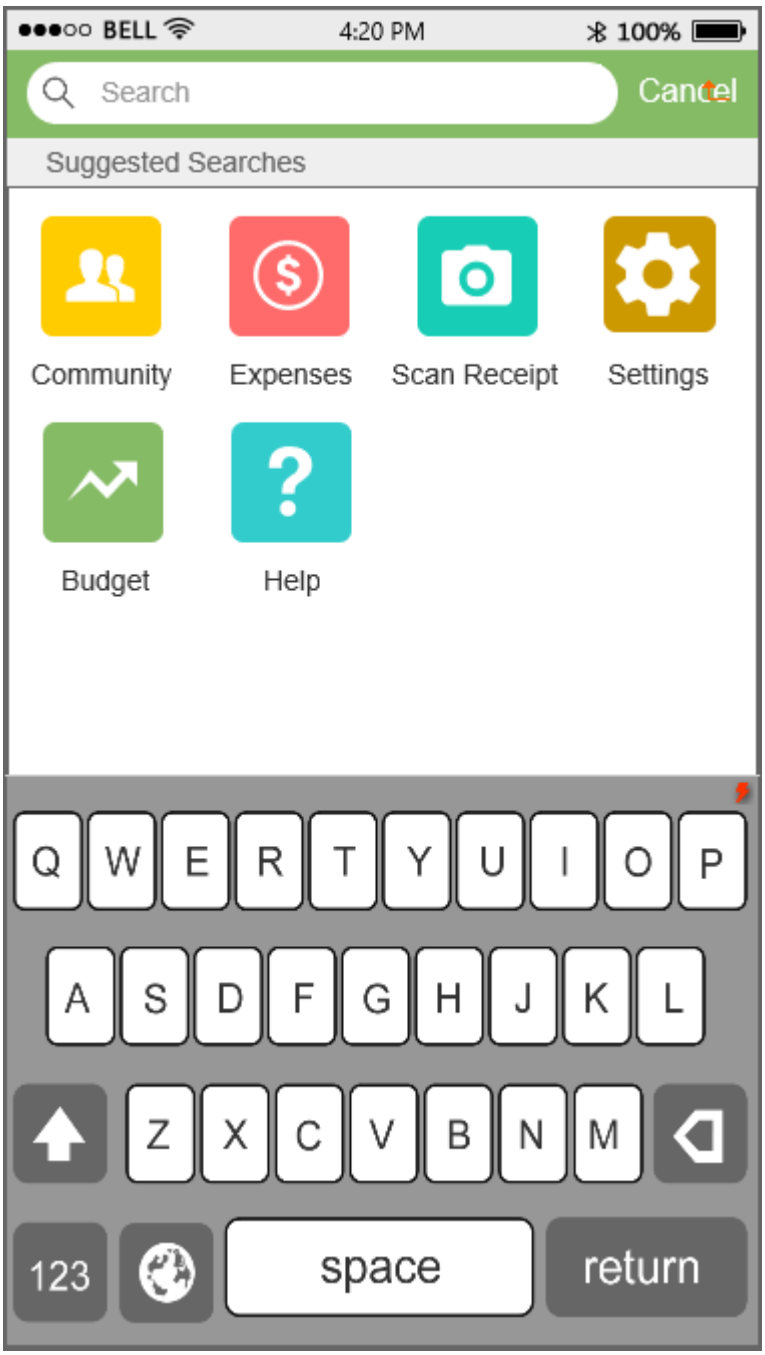
This is the login screen. It shows our logo, and the requirements for signing in. In addition, it has a ‘New Account’ button to let new users sign up. It is simple and conveys our intentions perfectly.



This is the create new account screen. The ‘New Account’ button would lead here. It has information pertinent for a new user to create an account. It has a button for singing up and then a button to go back to the login screen.



This is the what home screen once a user has logged in looks like. It has a graph that conveys the current spending, as well as budgeting information for the user. It also has detailed statistics about a user's spending and projected spending.



This is what the search screen looks like. It has a search bar to look through expenses. The search functionality will be extremely intensive, being able to look through not only expenses, but any data and functionality within the app. There are buttons for expenses, the community, settings, scanning a receipt, and etc.

This is what the camera screen from the ‘Scan Receipt’ button looks like. It is simply a page that lets you take a picture. In addition, there are quick buttons at the bottom for the convenience of the user.