SpendWiser

User Guide

Nathan Ng
Nathaniel Kong
Nathaniel Wright
Troi-Ryan Stoeffler

Preface **4**
README 4
Intended Audience 4

Product Overview **4**
Background 4
Description 4

Installation and Distribution **6**
Server-side Authentication Setup 6
Prerequisites 6
Firebase Project Setup 6
Authentication Setup 6
Login Services 7
Email/Password Sign In 7
Facebook Sign In 7
Google Sign In 8
Apple Sign In 10
Database Server Setup 11
Option A: Firebase Firestore 11
Prerequisites 11
Setup 11
Option B: MongoDB with Docker 12
Prerequisites 12
Firebase Admin Private Key 12
Setup 14
App Environment Setup 14
Prerequisites 14
Basic Setup and Testing 15
Distribution (App store) 16
Android 16
iOS 16

Functionalities **17**
Main Screen 17
Your Cards 18
Add a Card 19
Search for a Card 19
Add Card from Camera 20
Add Card Manually 20
Display Card 21
Spend Analyzer 22

Settings 23
Sign in as Guest 23

Troubleshooting **24**
Crashes 24
Bugs Without Crashes 24

Frequently asked Questions **24**

Contact Information **25**

Glossary **26**

Appendix: Design Document **27**
Introduction 27
Technology Survey 27
Real-World Constraints 30
Dependencies 31
System Architecture Overview 31
Server Architecture 31
Front-End 31
APIs 31
Requirements 32
User Stories (Sorted By Priority) 32
Github Repository 33
Technologies Employed 33
Social/ Legal Aspect 33
Acceptance Tests 33
Cost Analysis 34

Preface

README

This document constitutes the user manual for the SpendWiser application. It includes documentation on installing the application development environment, how the application is deployed, and a basic overview of the system architecture and application functionality. Please see each related section for further information.

Intended Audience

This manual is intended for groups or individuals who are looking to understand and further develop the SpendWiser application as well as gain a working knowledge of how the application is intended to function and be used. If further development is desired, a working knowledge of Javascript and ReactJS will be desired for application development and basic knowledge of NodeJS will be desired to extend any backend functionality. While other technologies are used within the application, they are not essential in the understanding of the application as a whole.

Product Overview

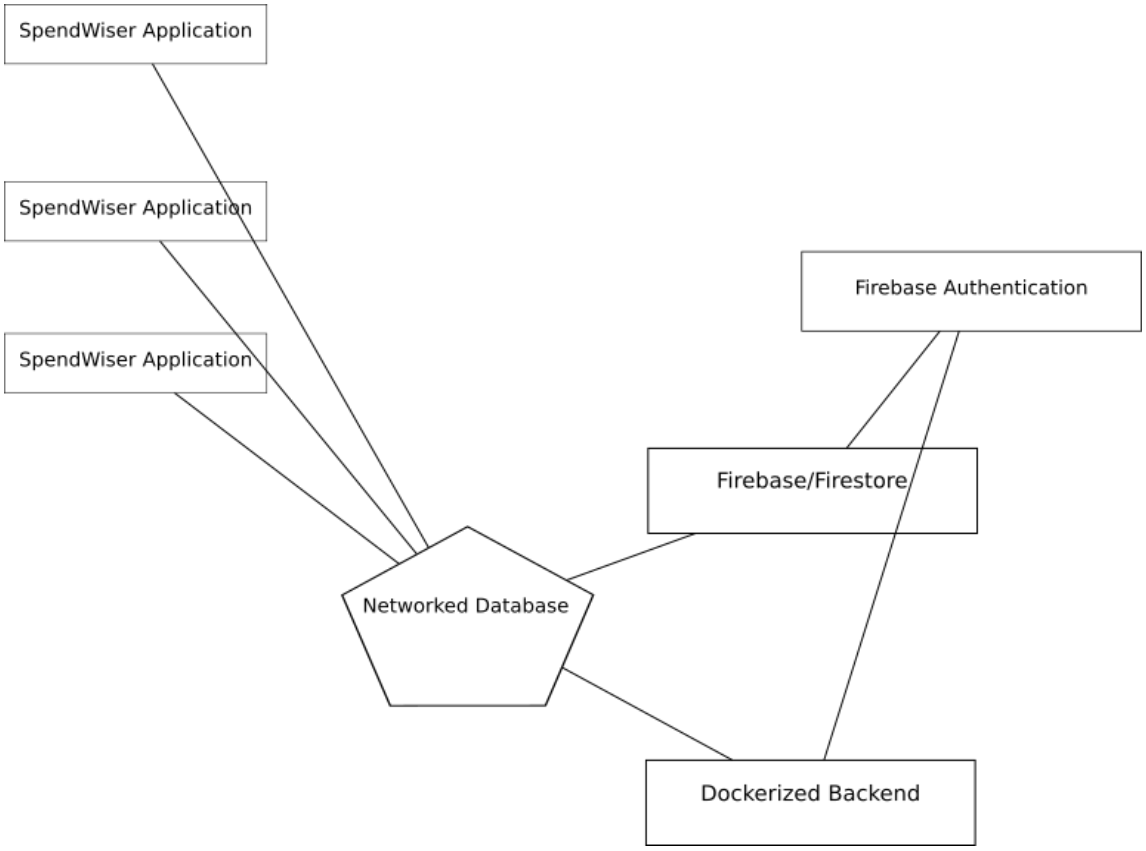
Background

The usage of credit cards has increased over the years and more businesses are accepting them. However, with the large range of credit cards, each with their own set of rewards and benefits, cardholders who own multiple of them might be confused as to which credit card he/she should use in a given store. Moreover, cardholders who possess multiple credit cards might wish to have an application that can keep track of his/her overall spendings.

Description

The SpendWiser application allows a user to manage a digital portfolio of all their credit cards. By specifying which credit cards are in their possession, the SpendWiser application will use this information along with the phone’s geolocation to suggest the best credit card to be used at whatever store the user is shopping at.

The SpendWiser product is divided into two main components. The first is the frontend application that the user interacts with on their phone. The second is the backend which constitutes a backing database and user authentication.



Here is an overview of the interactions between each component. The phone application is able to do five things: request a login, request card information, add cards to a user's account, add transactions to a user's account, and edit previously added transactions. All of these interactions will occur locally on the phone as it is backed by a local database. If an internet connection is available to the phone, any interactions specified above will be mirrored on the remote database as well.

The backend can field login attempts and is the main source of truth for what card data is associated with a user's account. It is also the only component that contains the entirety of the card information with information on over 100 different types of credit cards. When an application requests a card be added to a user's account, only that specific card information is sent back to the application. There are currently two versions of the backend that can be deployed: a dockerized NodeJS server application or firebase.

The chart shown above is the way these interactions between components are connected. Each phone application is connected to an abstract layer (the database api) that is represented as the "Network Database". In reality, this database is either one or the other: the firebase/firestore implementation or the dockerized backend implementation. The application will speak to one or the other, but not both at the same time. Either backend can then reach out to the firebase API to provide easy and safe user authentication, presenting that interface to the phone application seamlessly.

Installation and Distribution

Server-side Authentication Setup

Prerequisites

1. Firebase account (through a Google Account) for authentication and (optionally) Firestore. See the website for additional information.

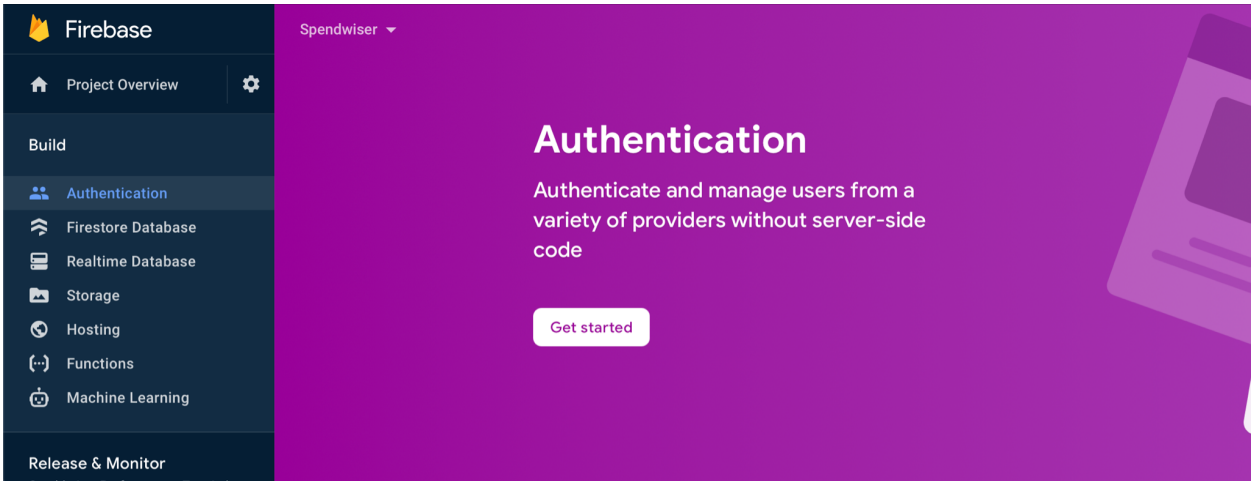
Firebase Project Setup

To continue, a Firebase project first has to be created in order to use authentication and (optionally) Firestore.

1. After logging in, click “Get Started” or “Go to console” from the main Firebase website to access your list of projects.
2. Depending on whether you have previously created a project, click on “Create a project” or “Add project” to start a new project.
3. Set a project name (we will use “Spendwiser” in this guide), read and accept the Firebase terms and conditions, and click “Continue.”
4. Check whether the projects should have Google Analytics or not (recommended).
    a. **If enabled**: Click “Continue” to configure Analytics.
        i. *If it asks for an Analytics account,* choose the “Default Account For Firebase.”
        ii. *If it does not,* set your preferred Analytics location and read and accept Google Analytics terms and conditions.
        iii. Press “Create project” to create your project.
    b. **If disabled:** Click “Create project” to create your project.
5. The setup might take a while, but when it’s ready, press “Continue” to finish the setup and access the dashboard/console.

Authentication Setup

1. From the Firebase project dashboard/console, open the “Build” tab on the left and enter the “Authentication” page.



2. Click the “Get started” button and now server-side authentication is ready to be configured. Follow the instructions in the next section to properly configure the individual sign-in methods.

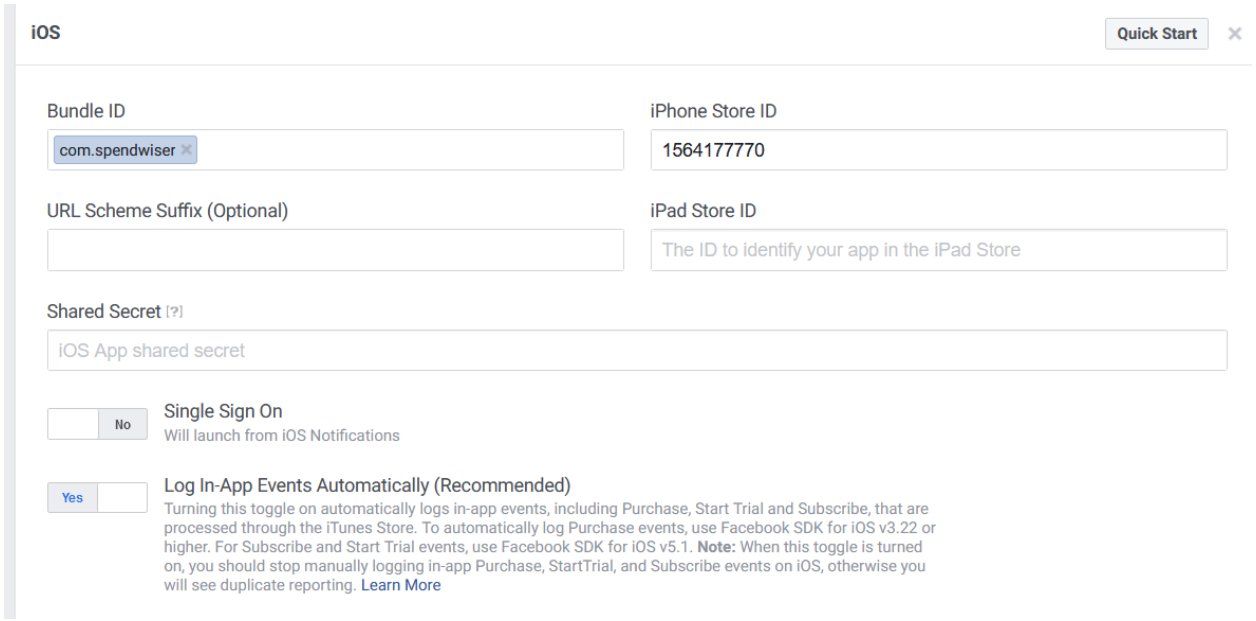
Login Services

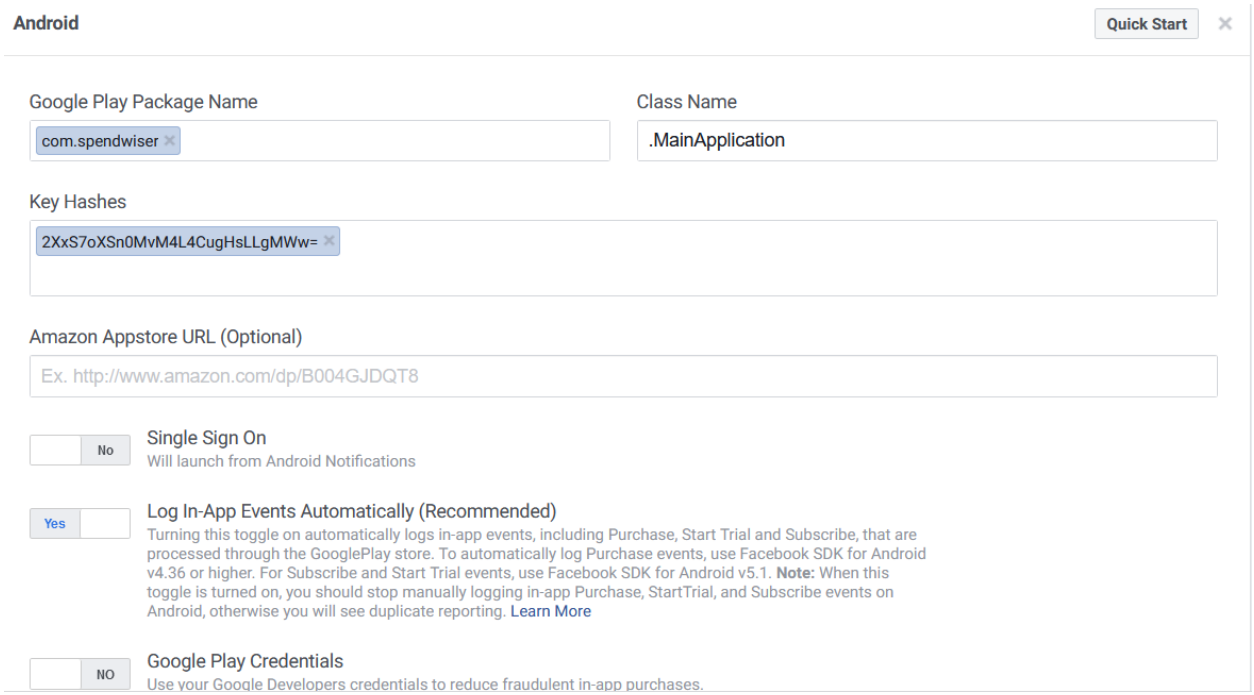
There are four different login services exposed to the SpendWiser application: Email/Password login, Google Sign In, Facebook Sign In, and Apple Sign In. Each type of login service requires some support from the Expo libraries as well as external setup on each sign in services developer websites respectively.

Email/Password Sign In

This type of sign in uses the Firebase API to provide secure email/password sign in. This is the simplest use case and requires no further external setup. To enable it, go to the “Sign-in Method” tab from the Authentication page from the Authentication Setup section. Click on “Email/Password,” check the top-most checkbox, and click “Save.”

Facebook Sign In

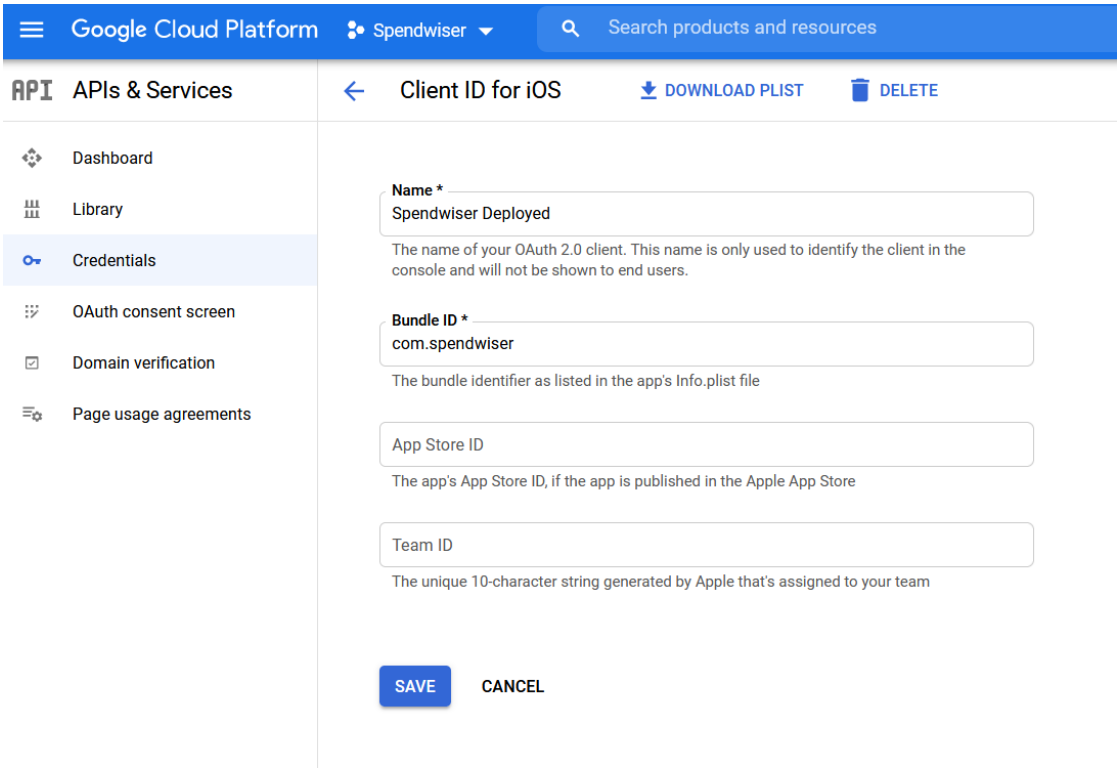


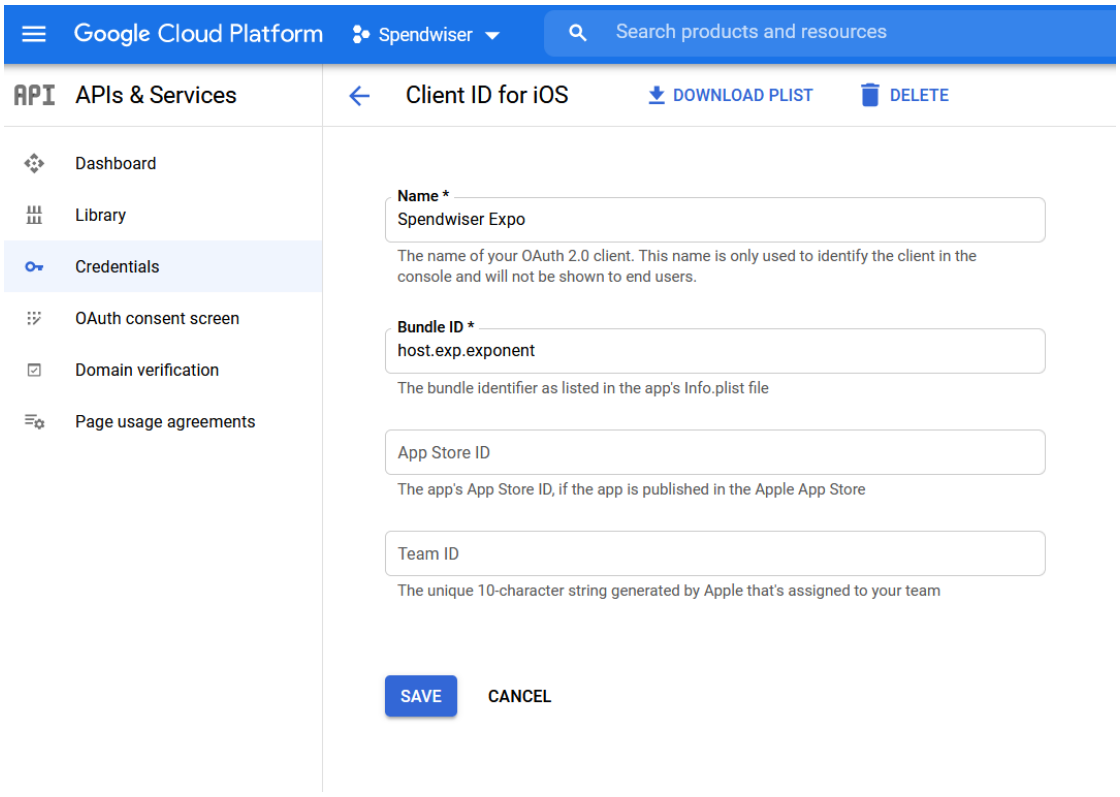


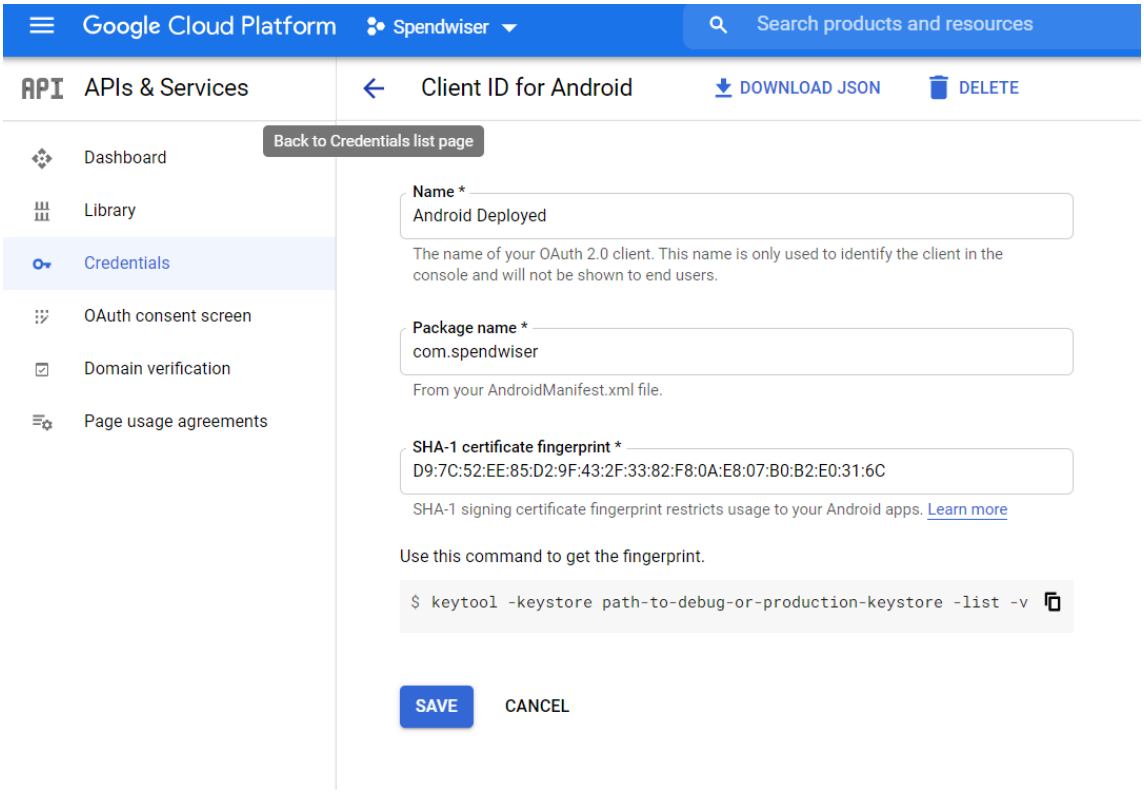
For Facebook, a developer application needs to be created and linked to the SpendWiser phone application. Both iOS and Android need specific configuration to specify the BundleID or the Package Name respectively, as well as other identifying information about the application. Please see the above examples for exact details on how each should be configured. To add it to your Firebase application, follow the steps from the Firebase documentation.

Google Sign In

For google sign in, OAuth tokens need to be created in the Google Developer Console. One is needed for the deployed application, and another for the Debug builds that will run in expo if and only if Google Sign In is needed in this environment.


Google Cloud Platform
Spendwiser
Search products and resources
APIs & Services
Client ID for iOS
DOWNLOAD PLIST
DELETE
Dashboard
Library
Credentials
OAuth consent screen
Domain verification
Page usage agreements
Name *
Spendwiser Deployed
The name of your OAuth 2.0 client. This name is only used to identify the client in the console and will not be shown to end users.
Bundle ID *
com.spendwiser
The bundle identifier as listed in the app's Info.plist file
App Store ID
The app's App Store ID, if the app is published in the Apple App Store
Team ID
The unique 10-character string generated by Apple that's assigned to your team
SAVE
CANCEL


Google Cloud Platform
Spendwiser
Search products and resources
APIs & Services
Client ID for iOS
DOWNLOAD PLIST
DELETE
Dashboard
Library
Credentials
OAuth consent screen
Domain verification
Page usage agreements
Name *
Spendwiser Expo
The name of your OAuth 2.0 client. This name is only used to identify the client in the console and will not be shown to end users.
Bundle ID *
host.exp.exponent
The bundle identifier as listed in the app's Info.plist file
App Store ID
The app's App Store ID, if the app is published in the Apple App Store
Team ID
The unique 10-character string generated by Apple that's assigned to your team
SAVE
CANCEL



Each OAuth token for IOS will have different Bundle IDs. The deployed application on the app store will have a Bundle ID of “com.spendwiser” and the expo debug environment has a Bundle ID of “host.exp.exponent”. You can see that for android, instead of the Bundle ID the SHA-1 certificate is required. To add it to your Firebase application, follow the steps from the Firebase documentation.

Apple Sign In

To use Sign in with Apple, follow the “Configure Sign In With Apple” section (steps 1-2) from the Firebase documentation to set up the required settings on the Apple Developer site. To enable it on Firebase, follow the “Enable Apple as a sign-in provider” section (step 2).

Important Note: when specifying the Service ID, if you are utilizing Expo for development, use `host.exp.Exponent` for it to work on the Expo Go app. If you are then using it on Testflight or production, use the Service ID that you created for your app.

For more help, you can use this link:
https://danielwsinger.medium.com/adding-sign-in-with-apple-to-a-managed-expo-app-using-firebase-authentication-ca331b4de05

Database Server Setup

For the database on the server-side, Firebase Firestore or MongoDB with Docker can be used. See the relevant section for setup instructions.

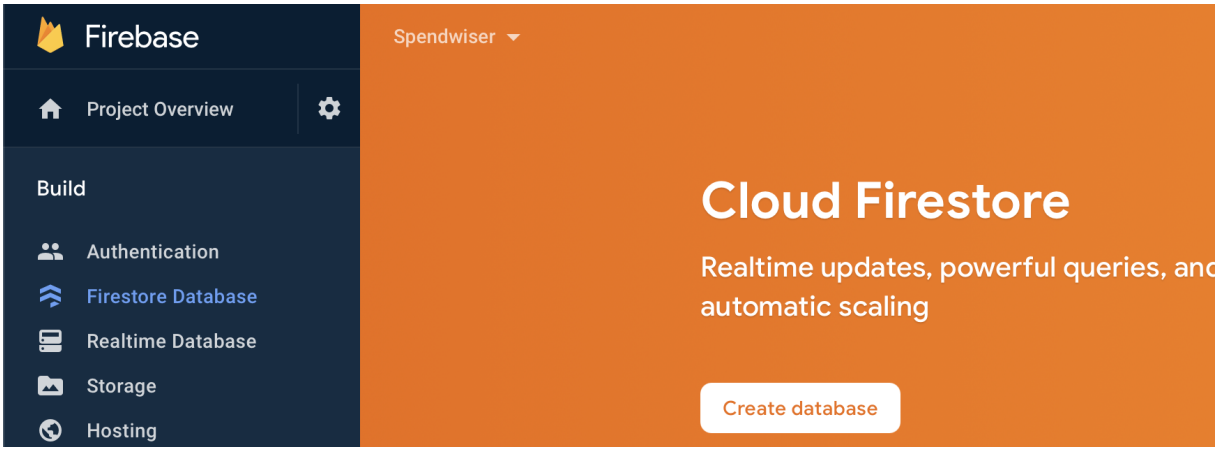
Option A: Firebase Firestore

Prerequisites

1. A Firebase account and project setup (see Firebase Project Setup in the Server-side Authentication Setup section for more details).

Setup

1. From the Firebase project dashboard/console, open the “Build” tab on the left and enter the “Firestore Database.” Click on “Create database.”



2. For the “Secure rules for Firestore” select “Start in production mode” and click next.
3. Select your desired “Cloud Firestore location” (recommended to leave at “us-central”). Click “Enable” when done.
4. Enter the “Rules” tab to set up database read/write rules. Copy and paste the database rules found from the provided source files from the `firebase.rules` file. Click “Publish” to publish the new rules.
5. Go back to the “Data” tab and start a new collection by clicking on “Start collection.” Give it the identifier “cards” (without the quotes) and click “Next.” Now we will go over the document structure of cards since the creation of the first document is required.
   Note: If needed, there is a script in the “load_card” folder that reads a text file and loads a card into our database. Further documentation for its usage is also provided there.

a. Firstly, enable Auto-ID for the document. For each card document, they are setup as follows:
    i. **cardId** (type: string) - The reference id for the card. Copy and paste the generated “Document ID” here.
    ii. **name** (type: string) - The actual name of the card that will show in the app
    iii. **image** (type: string) - The web URL to an image of the card itself
    iv. **url** (type: string) - The web URL to the website with the card’s info
    v. **dateAdded** (type: timestamp) - The time of creation this document was created
    vi. **benefits** (type: array, element-type: string) - The descriptions of the benefits of the card
    vii. **rewardType** (type: string) - The type of reward this card has that is stored in “rewards” (Cashback, Points, Other, etc.).
    viii. **rewards** (type: map, element-type: number) - The rewards the card has specifically, converted to a universal type using conversion.
        1. E.g. “grocery” with 1 (for 1% cashback using a conversion of 1)
    ix. **conversion** (type: number) - The conversion for what is stored in the database in “rewards.”
b. When done, click “Save” to enter the document into the collection.

Option B: MongoDB with Docker

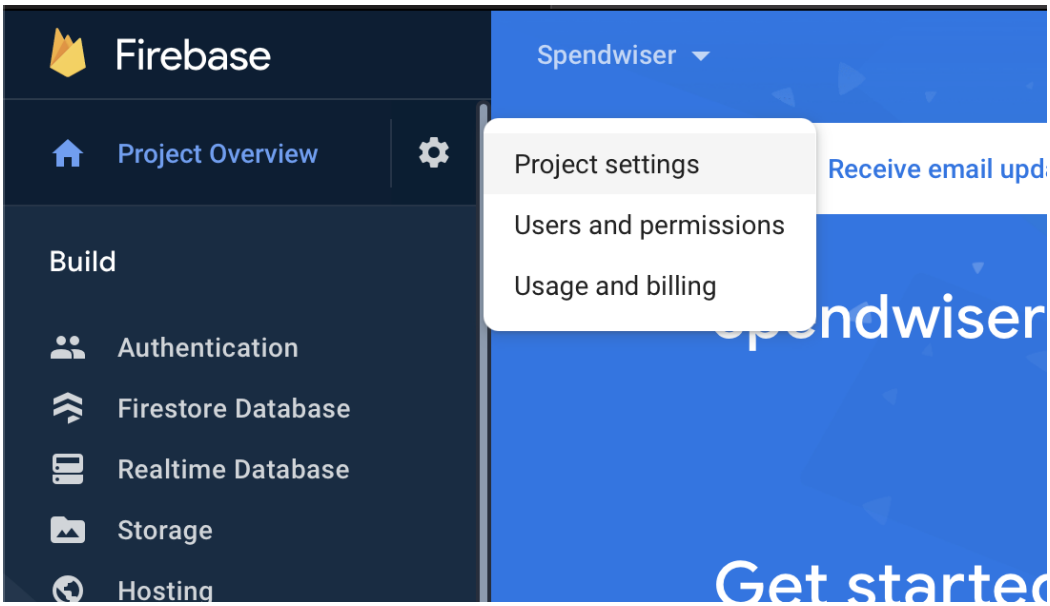
Prerequisites

1. A machine capable of running Docker. If not installed and configured already, please see the Docker documentation for instructions. A server that is always running is recommended.
2. An SSL certificate and key. We recommend using Let’s Encrypt and Certbot since it is free and has automatic renewal. For further instructions, see the Let’s Encrypt getting started page.
3. A Firebase admin private key for authentication. This will allow the server to communicate with Firebase for user authentication. To get this key, please follow the following section

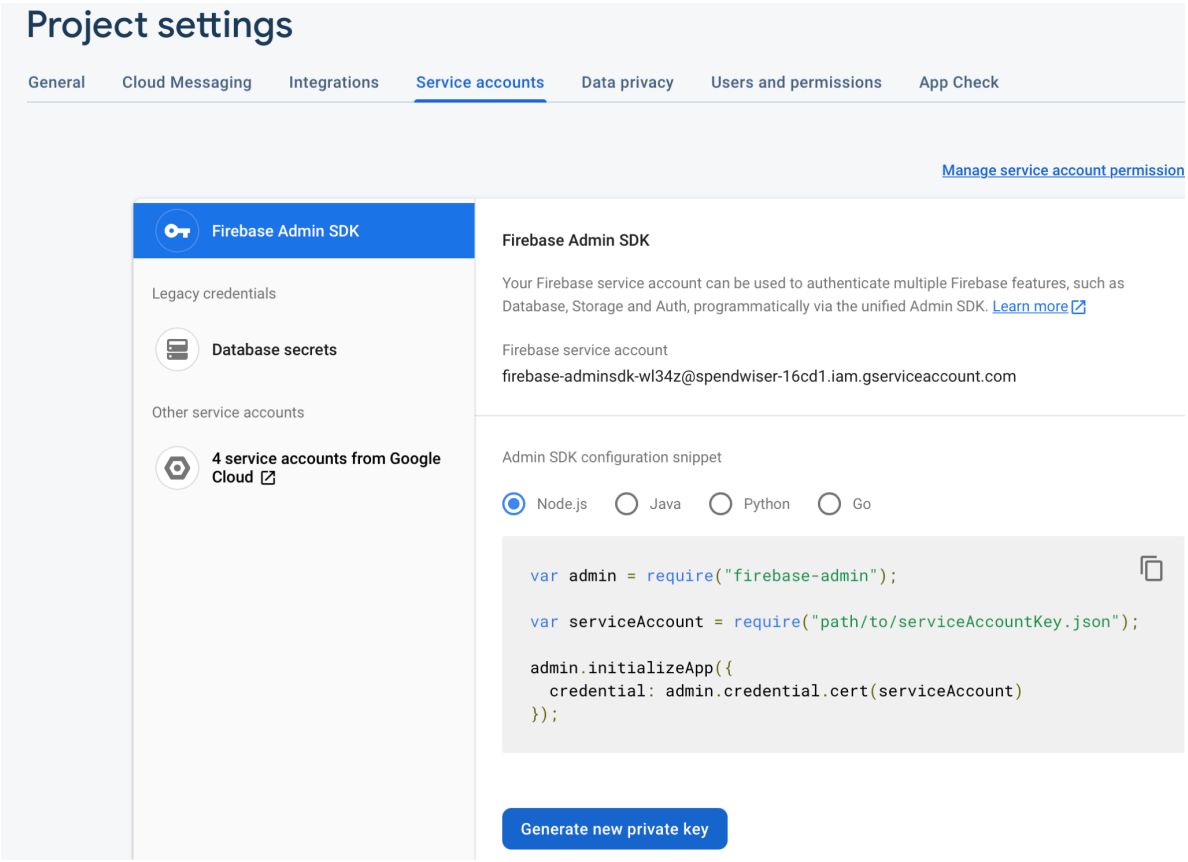
Firebase Admin Private Key

In this section, we will be generating and downloading a private key to use with the Docker server.

1. From the Firebase project dashboard/console, on the left-hand sidebar, click the gear to the right of “Project Overview” and click “Project settings” to enter the settings page



2. From the settings page, enter the “Service accounts” page and click “Generate new private key”



3. Press “Generate key” to confirm and save the file in a secure location for later use.

Setup

1. Create a directory for the source code files. In this guide, we will use `~/Spendwiser/` as an example. Navigate into this directory using `cd` and extract the supplied source code into the folder (whether it be from a `git clone` from the repository or an extraction from a zip file downloaded from Github).
2. Navigate into the `server` directory so that the current working directory becomes `~/Spendwiser/server`.
3. **Configuration**: Copy the `.env-template` file from `~/Spendwiser/server/config` to `~/Spendwiser/server`, modify the configuration to what you desire and finally save as `.env`
    a. You can copy your SSL certificate and key to the `~/Spendwiser/server/cert` folder. NOTE: Make sure the configuration variable properly point to the right files
    b. You can copy your Firebase admin key generated from the Firebase Admin Private Key section to `~/Spendwiser/server` so that you can set the `ADMIN_JSON` configuration variable to point to the file for authentication. NOTE: Since it's in the same directory as the .env configuration file, the variable is just the file name (with extension).
    c. If you have card data that should be loaded into the database, first move the JSON file to `~/Spendwiser/server` and change the `LOAD_FILE` configuration variable to point to that file. Run the server once with this configuration after everything loads up. Change the `LOAD_FILE` variable back to false when finished.
4. **Running**:
    a. When you are ready to run, first build the project by using `docker compose build`
    b. Now, run the server using `docker compose up` or `docker-compose up` depending on your docker version
    c. The server should be up and running now. In order to connect to it, obtain a domain or server IP from your hosting provider.

App Environment Setup

Prerequisites

1. A machine capable of running Node.js and npm. If not installed and configured already, please see the npm documentation for instructions.
2. Expo command line tools installed on the machine. The Expo Go app must be installed if testing on a mobile device. Please see the Expo documentation for further information and installation instructions.
    a. If an emulator is desired for testing, please see the documentation for the installation of the iOS Simulator (macOS only) and the Android emulator.

Basic Setup and Testing

1. Create a directory for the source code files. In this guide, we will use `~/Spendwiser/` as an example. Navigate into this directory using `cd` and extract the supplied source code into the folder (whether it be from a `git clone` from the repository or an extraction from the supplied zip file).
2. Navigate into the `app` directory so that the current working directory becomes `~/Spendwiser/app`. First setup the Expo environment by using `npm install`. In order to make sure every module is installed, run `expo install`.
3. **Configuration**: Copy the `.env-template` file from `~/Spendwiser/app/config` to `~/Spendwiser/app` and name the file as `.env`
4. Modify the `.env` file to suit your configuration
    a. For the Firebase configuration variables
        i. We first have to create a new app in the Firebase project. To do this, first go the the same Project Settings page accessed in the Firebase Admin Private Key section
        ii. Under the "General" tab, scroll down to the "Your apps" section. Select the "</>" icon to create a new app.
        iii. Give the app a name (in this guide, we will use "Spendwiser"), leave the "Firebase Hosting" unchecked, and click "Register app"
        iv. It might take a little bit, but when it gives you a "firebaseConfig" JSON, copy and paste the respective values into the `.env` configuration file. When you are done, click "Continue to console."
    b. To get the Google Place Search API Key, please refer to the documentation for the places API.
    c. (Optional) If using a MongoDB docker server, provide the https server URL
5. **Testing**:
    a. **On Device**: Run `expo start` and a QR code will appear after a couple of seconds. Scan on your device while on the same WiFi network as the machine it's running on and the app should run.
    b. **iOS Simulator**: Run `expo start` and after the QR code appears, press `i` to run the simulator. For the limitations of the simulator, please see the Expo documentation.
    c. **Android emulator**: Run `expo start` and after the QR code appears, press `a` to run the emulator.

Distribution (App store)

For uploading builds and publishing to both the Google Play Store and the Apple App Store, follow the following steps.

1. Set up the app store listing - we set up these store listings through the senior design accounts, it's probably a good idea to set these up through a private account.
2. Create the build file - use the command "expo build:ios" or "expo build:android"
3. Upload the build file

Android

The android store listing was relatively simple and straightforward. One can follow the steps listed here: https://support.google.com/googleplay/android-developer/answer/9859152?hl=en

iOS

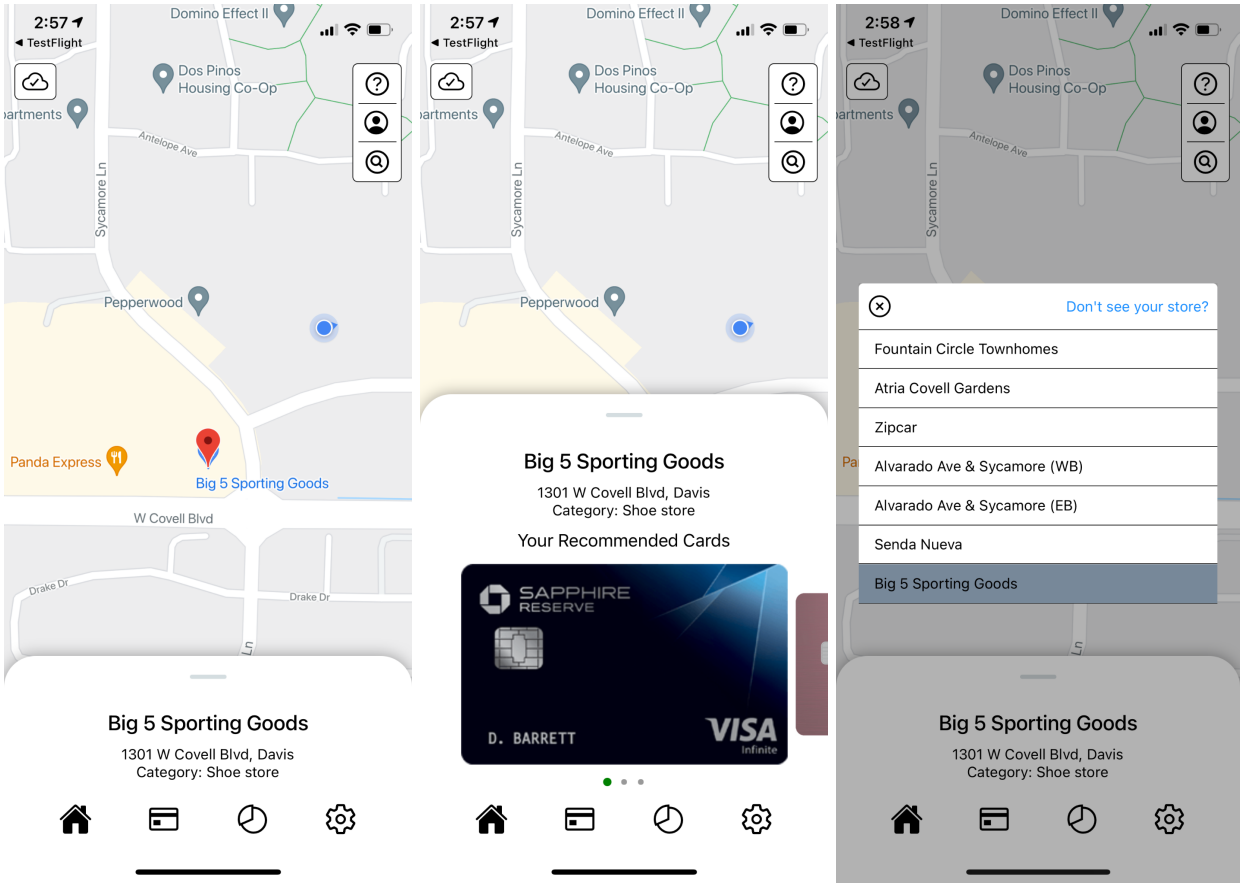
There are a number of steps you need to do to set up the actual app in the store. Since we used Expo, we followed the basic steps here: https://docs.expo.io/distribution/app-stores/

Once the app itself is set up, we followed the following steps to upload a build (.ipa) file to App Store Connect.

1. Build the .ipa binary from source code. Use the terminal command "expo build:ios" from the ./app directory. You will be prompted for apple developer credentials.
2. There are a number of ways to upload the binary to App Store Connect.
    a. For mac, you can use altool command line tools:
        i. "xcrun altool --validate-app --file <bundlepath> --username <username> --password <app specific password>"
        ii. "xcrun altool --upload-app --file <bundlepath> --username <username> --password <app specific password>"
    b. You can also automate flow using "expo upload:ios"
    c. Or, use the Transporter application on mac.

Functionalities

Main Screen



This is the home screen where the user is taken on a successful login. The page consists of a map, and an information area in the bottom sheet. The information area will show the selected store and if a user swipes the bottom sheet up, his recommended cards will be displayed in a carousel, with the first card being the best recommendation.

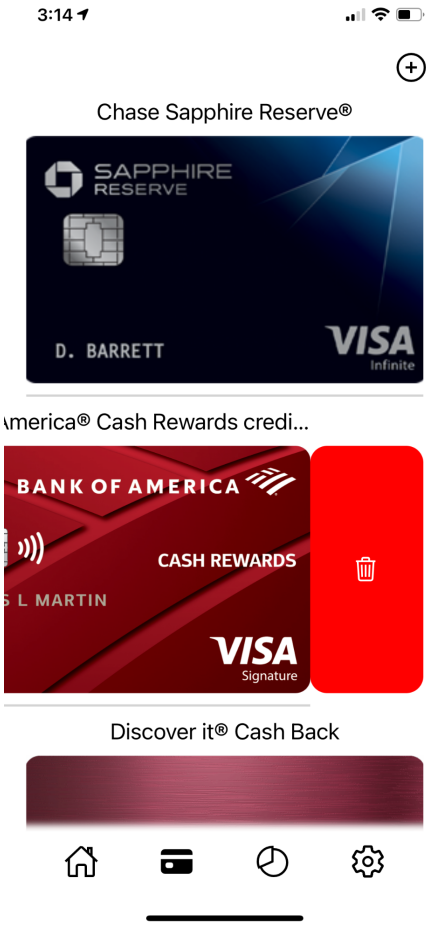
There are three buttons at the top right corner of the screen. They are the help button, re-center user button, and the store list button respectively. On the top left corner of the screen, there is a cloud icon that shows the status of your internet connection.

The main screen has certain features that allow the user to choose a store. Users can click the search button to select a store from the list, which is filled with stores in a 100m radius around the user. Users can also click stores (POI) in Google Maps to select a store. There is also an option to manually add a store to your store list by clicking the search button and “Don’t see your store?”.

All the previously mentioned features are accessible with location permissions enabled and having an internet connection. Without either of these, some features will not be available. The table below shows which features are available in the different use cases:

Feature	With location and internet	Without location, with internet	With location, without internet	Without location and internet
Re-center location	✓	×	✓	×
Initialize store list with surrounding stores	✓	×	×	×
Select store from Google Maps	✓	✓	×	×
Manual store input	✓	✓	✓	✓

Your Cards



As seen in the image on the right, this page contains a scrollable list that shows users the cards they have saved to their profile. The card images shown use stock images that are stored in our database, minimizing user's privacy concerns.

Users can click the plus icon on the top right of the screen to add a new card to his/her profile. More information on this feature can be found in the Add a Card section

A user also has the option to delete a card from his account by swiping to the left on a card. A user can also "disable" a card by tapping a card and choosing the "disable" option. This will cause the card not to show in the recommended card section and make the card have a light-gray overlay.

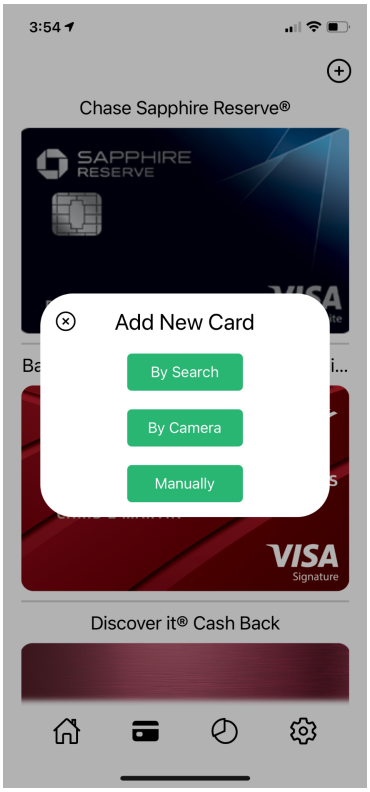
A user can click on a card to see the card's details, including it's rewards and user's transactions. More information can be seen in the Display Card section.

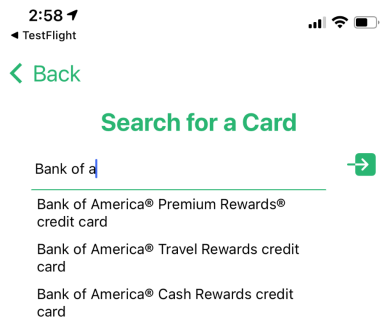
Add a Card

A user can add a card to their account using three methods:
1. Adding a card by searching through our database using a search bar.
2. Adding a card by taking a picture of their credit card, which then searches our database for similar card images.
3. Adding a card manually.

A user is only permitted to add a credit card to their account through our database and by search or by camera only if they are logged into an account. This means that if a user is not logged in, he/she only has access to adding a card manually.

One can open the *Add New Card* modal from the Your Cards page by clicking on the plus icon in the upper right.



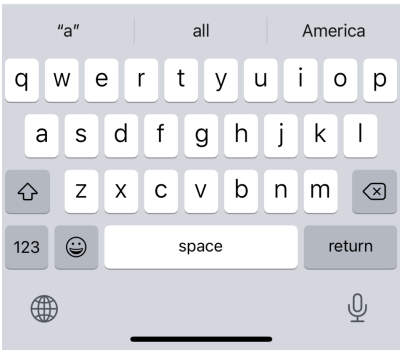


Search for a Card

With a stable internet connection, the user can input a credit card name and cards with similar names will be retrieved from the database.

Some cards might not exist in the database and users will then have to manually add a card.

Select a card by clicking the chosen card name and clicking the arrow button on the top right of the page. This will add the card to your profile and redirect you to the Your Cards page

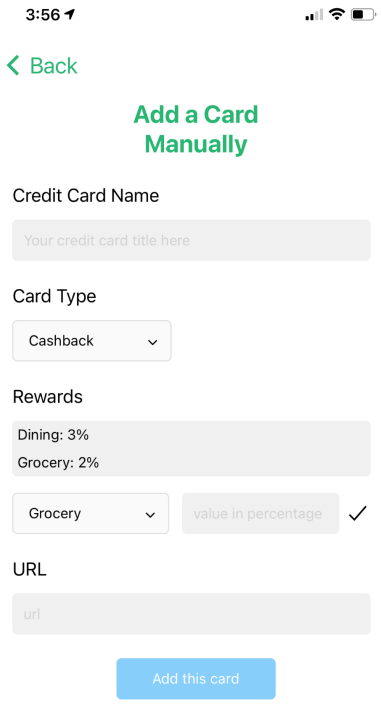


Add Card from Camera

Users can opt to add a card with a picture taken from his/her photo library or camera. Once a picture has been chosen, the user will be brought to the page as shown on the right.

Although our application does not save any information related to the picture, users with privacy concerns have the option to add manipulatable boxes to cover up sensitive information before the image is sent to Google Vision's API to be compared with images in our database.

If the user requires help, a help modal is available for users to get a better understanding of the page's features.



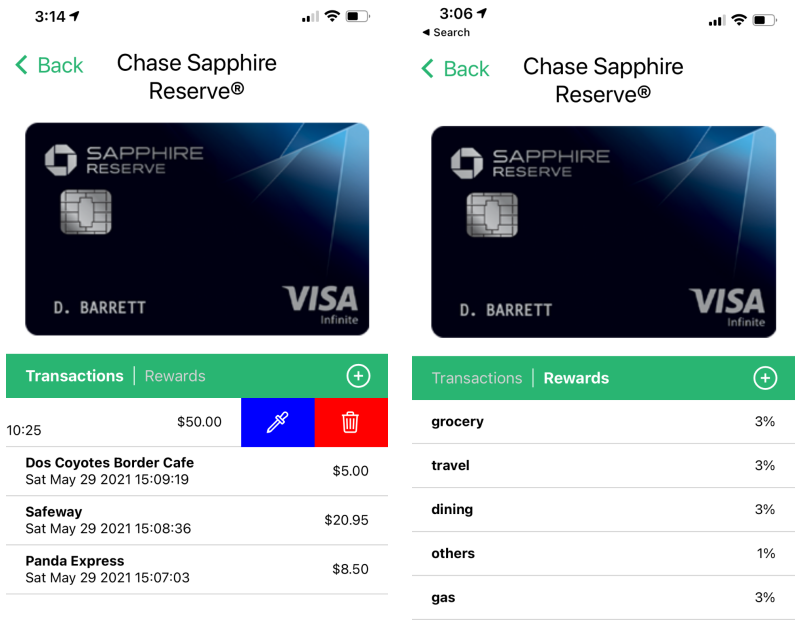
Add Card Manually

If a user is not logged in or is unable to find his credit card in the database, he/she can opt to add a card manually through this page.

A credit card name and at least one reward is required for the card to be added. If not, an error message will appear to notify users of the requirements.

Cards that are manually added will display a card image with a randomly generated color.

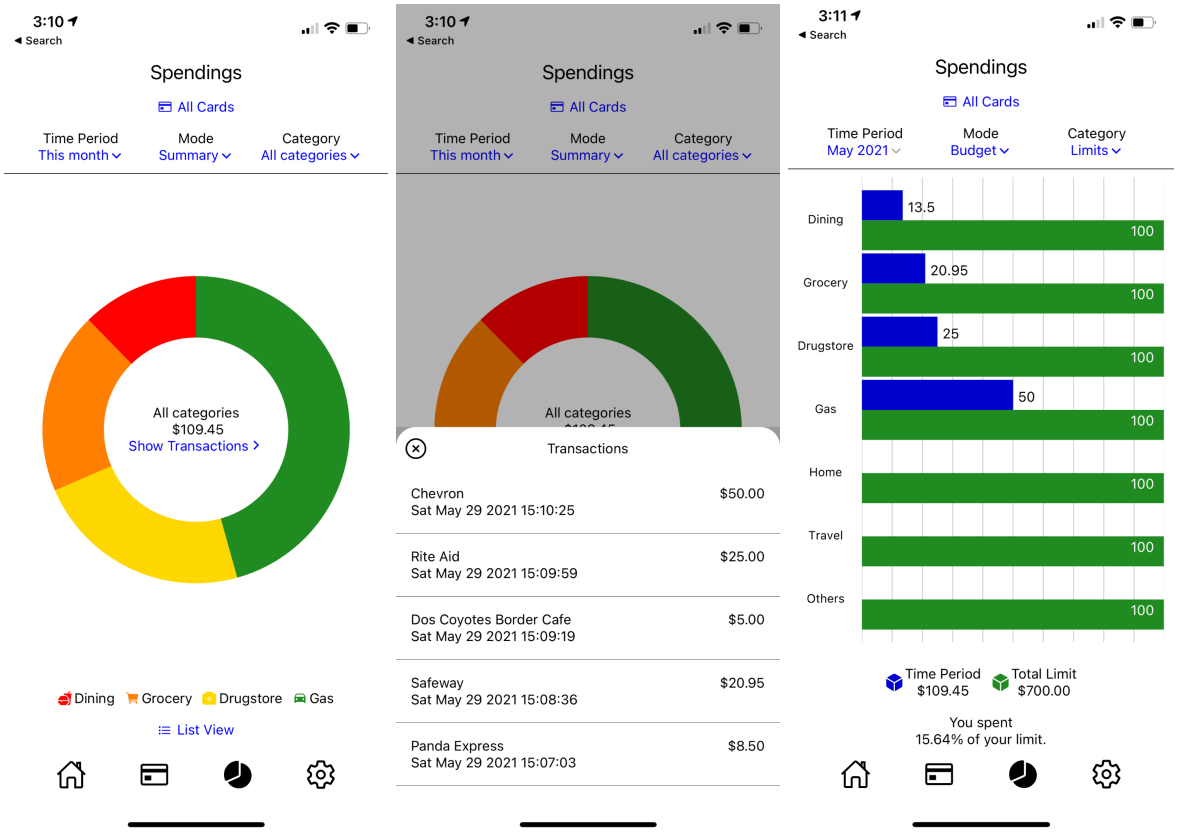
Display Card



The display card page shows a user's card information, including the card's rewards and the user's transaction history with the card. The selected card can also be deleted in this page.

To add a transaction, a user can click the plus icon under the card image. The user will be prompted to input how much he/she spent at their current selected store. Transactions that are added are then seen in the current page and in the Spend Analyzer page. A user can swipe left on a transaction to edit or delete it.

Spend Analyzer



The Spend Analyzer has three modes: Summary, Compare, Budget.

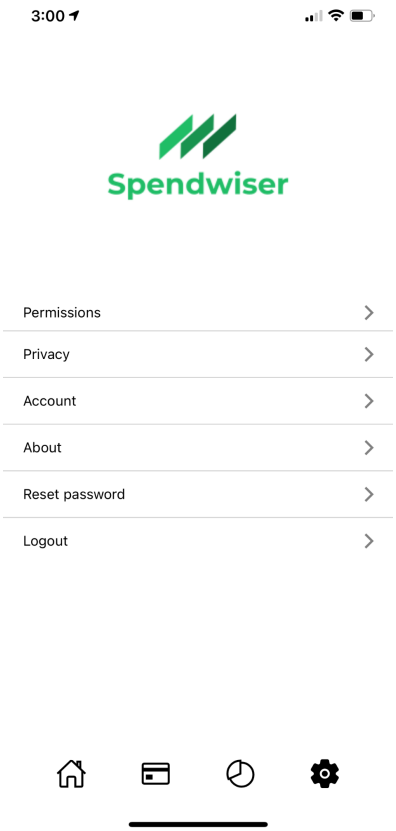
In the summary mode (left picture), which is the application's default mode, users can view their spendings for the current month, previous month, or the last three month in a pie chart or in a category list. Users can click "Show Transactions" to view their transactions for specific categories and for specific cards (center picture). Clicking the card text under *Spendings* will allow the user to filter which card he/she wants to view.

The compare mode allows users to compare their spendings by selecting two different time periods, while the budget mode (right picture) allows users to set category limits and compare a selected time period with said limits.

Settings

The settings page allows a user to edit items related to their account through a number of sub-pages.

From this page, a user can: edit the permissions of the application, view the privacy policy, view information about their account, view information about the developers, get a link to their email that resets their password, and log out of their account.



Sign in as Guest

Users who do not wish to create an account with us can opt to continue without logging in. The option to do so is located in the login screen. However, the user will have limited access to the application's functionality. Moreover, the user's profile will only be stored locally and not added to our database. This means that the profile cannot be transferred over to another device.

Guest users will only be able to add cards manually as they will not have access to our card database.

If a user decides to create an account after signing in as a guest, he/she will have to manually transfer all her data over to the new account.

Troubleshooting

There are a number of ways to debug why the application might be failing. The developer code documentation has comprehensive descriptions of all functions and components. We've separated troubleshooting scenarios to *crashes* and *bugs that aren't causing crashes.*

Crashes

Crashes for the application can be logged using the tool Sentry, that detects crashes within the application. It logs the application and the stack trace on the sentry website. View these to see the exact stack traces and errors, and you should be able to get an idea of why the application is crashing. In addition, the Google Play store and Apple store also have some crash logs. If the application crashes users are allowed to provide feedback for Apple.

Bugs Without Crashes

Because mobile applications have no debuggers, one option is to run the application locally using "expo start" and having console logs to view information. If the problem is related to the database, you can view the firebase website to see the entirety of the database.

Frequently asked Questions

1. What information is being collected? What is it utilized for?
    a. We DO NOT collect any personal information that might cause security issues. For example, we do not take pictures of your credit card and we don't store sensitive information like the credit card number. The only information we do collect is information you're willing to store within the app, that is securely stored in our database and protected by your password. This information, such as what cards you have and the transactions are only used to provide suggestions based on the spend analyzer and the recommended card.
2. How does the app rank the different cards if they have different types of rewards like Cash Back vs Points?
    a. Each card has a cash translation in our system, so we account for cashback versus points when we recommend you a card. If you still aren't convinced with our recommendation, please leave us feedback through the app store.
3. What happens if the store I am at is not listed?
    a. If the store you're at isn't listed, you can manually input a store name with the relevant category
4. What happens when I don't provide location access?
    a. No problem! You can manually input a store or click a store's Point of Interest on Google Maps.
5. Can I delete my account?
    a. You can delete your account any time. The option is in the settings.

6. Why does the card scanning give me a list of cards?
    a. Our image-to-text recognition using Google Cloud's Vision API isn't perfect, so the best we can do is give you suggestions based on what your card might be.
7. Do I need to use an account?
    a. No! You can actually use an offline account and have all your information unrelated to your name or email. However, if you do this you will have to manually switch all your information to your account later if you do decide to use a regular account.
8. Why is one card suggested over another at my store?
    a. A card is suggested over others at particular stores based on the benefits and rewards associated with that card. If a particular credit card provides better benefits for shopping at grocery stores it will be rated higher than others when you are at stores like Safeway, Food For Less, etc.
9. Why are my credit cards' rewards different from those listed in the app?
    a. Some credit cards have rotating or alternating reward structures that create problems for displaying accurate rewards within the application.
10. Is my password safe in your database?
    a. Your password is securely encrypted in our Firebase database through Google's hashing method. This will prevent unauthorized users from stealing your password from the database.

Contact Information

- Nathan Ng
    - wxng@ucdavis.edu
- Nathaniel Wright
    - njwright@ucdavis.edu
- Nathaniel Kong
    - nskong@ucdavis.edu
- Troi-Ryan R. Stoeffler
    - trstoeffler@ucdavis.edu

Glossary

- **Credit Card:** a card issued by a bank in the United States that provides a line of credit, benefits, and rewards for its usage
- **Docker:** a platform and program that uses virtualization to deliver software in packages called containers
- **Expo:** an open-source platform for making universal native apps for Android, iOS, and the web with JavaScript and React.
- **Firebase:** a platform developed by Google for creating mobile and web applications. In the context of this user guide we are referring to the storage and authentication functionalities of firebase.
- **Google Cloud Platform:** a large suite of services provided by Google to provide cloud computing/storage services
- **Google Places API:** a service that returns information about places using HTTP requests
- **IP Address:** An identifier for a device using Internet Protocols
- **JavaScript:** a programming language primarily developed for web applications, but can also be used within the React Native framework for GUI and application programming
- **MongoDB:** a document-oriented database that is classified as a NoSQL program
- **NPM:** a package manager for the JavaScript programming language
- **React:** a JavaScript library for building user interfaces or UI components using a mixture of HTML and JavaScript syntax.
- **React Component:** a “class”/”object” within the React framework used for displaying different application GUI screens
- **React Native:** a JavaScript library that provides mobile application framework for IOS and Android development using the React library
- **SSL:** The “Secure Socket Layer” used for internet communication which provides a means of encrypted communication between endpoints
- **Spend Analyzer:** Our page that analyzes and give you a breakdown of your transactions
- **Transaction:** an exchange of money between the user and a place of business

Appendix: Design Document

Introduction

Modern transactions involve the heavy usage of credit card technology. Due to the large number of banking companies in the United States there are numerous credit cards, each with their own unique advantages. For example, Discover Card has 5% Cash Back Calendar, versus Chase Sapphire Card, that can waive the insurance cost when renting a car. SpendWiser seeks to optimize the use of different credit cards for each user by choosing the credit card that will give the user the most benefit for each of their use cases and locations.

Technology Survey

- Front-End
    - React Native
        - Pros
            - Based on JavaScript (provides access to the large NPM package ecosystem)
            - React for UI (Team has experience using React for frontend web development)
            - Open source (there is some community involvement)
            - Uses native UI components
        - Cons
            - Does not have a 1.0 release, may be compatibility issues
            - Specific native platform features may not be accessible
            - Startup delay (may become less of an issue as React Native develops)
    - Flutter
        - Pros
            - Compiled for each platform (possible performance gains versus React Native)
            - Easy to get a minimum viable product up and running
            - Channels (custom platform specific code) to access IOS or Android specific features
        - Cons
            - Even newer than React Native (relatively immature framework)
            - Dart is still a new programming language and doesn't have access to as many libraries as JavaScript
            - Does not use native UI elements
    - Native iOS and Android
        - Pros

                - Access to all native libraries
                - Team has experience with Java from coursework (assumes the use of Android Java SDK)
            - Cons
                - No previous experience using these either Swift or Kotlin for Native development
                - Would need two versions of the application (difficult for the amount of time allotted for this course)
    - **Conclusion: We will use React Native as our team is comfortable with the technology and it gives cross platform access to native UI elements**
- Back-End
    - Django
        - Pros
            - Very mature due to age
            - Uses python, fast development speed
            - More secure
        - Cons
            - Can be bloated
            - Won't seamlessly integrate with a Javascript Full Stack
    - Node
        - Pros
            - A lot of support due to popularity
            - Works well with React frontend and Firebase
            - Lightweight and easy to get up and running
        - Cons
            - Not as performant
            - Not as secure
    - **Conclusion: We will use Node since it would integrate well with a React Native front end with the Javascript Stack**
- Server Architecture/Database System
    - AWS (Databases on AWS/Build Mobile & Web Apps Fast | AWS Amplify | Amazon Web Services)
        - Pros
            - Allows for easy app deployment with many options available
            - Many database options for different use cases
            - Integrates with AWS
        - Cons
            - Steeper learning curve if using other AWS features
            - More options available leads to more complexity
    - Firebase (Firebase by Google)
        - Pros
            - Built for app deployment, interfaces with NodeJS already
            - Real Time applications
            - Ease of use

        - Cons
            - Not relational, similar to MongoDB/NoSQL
            - Not great for larger datasets/apps (performance and cost)
            - Simplicity can hinder flexibility
    - **Conclusion: Leaning toward Firebase due to the ease of use and quickness of deployment**
- Credit Card Information
    - CCstack.io API
        - Pros
            - Has information about the rewards for 2,883 credit cards
            - Has other information as well, including earnings, bank
        - Cons
            - Costs $9/month or $30/month, depending on usage
    - Other Options
        - Manually gathering information or ask user for rewards
            - Pros
                - Freedom for gathering information
            - Cons
                - Time consuming
                - May require scraping
                - Requires another database
    - **Conclusion: We will likely use the CCstack.io API because gathering information for thousands of sources is very time consuming.**
- Geolocation Information
    - Google Places API
        - Pros
            - Based on the user's geolocation, provides access to the store's category.
            - Search for stores in a specified radius with the user's geolocation as it's centroid.
        - Cons
            - Store category might not accurately reflect actual merchant category
            - Some stores might not be registered in Google Maps.
        - https://developers.google.com/places/web-service/overview
    - Visa Merchant Locator API
        - Pros
            - Merchant category accurate since it is retrieved from Visa
            - Gives a list of stores around the given geolocation, enabling user to choose other stores if defaulted store is wrong
        - Cons
            - Store might not take Visa, which results in the store not being found
        - https://developer.visa.com/capabilities/merchant_locator

    - **Conclusion: We decided to use Google Places API as it has the ability to search for stores in a radius around the specified geolocation, and most stores where the users would search for will be found in google maps.**
- Image Recognition
    - Image Similarity API
        - This API matches two images given their urls. The API uses a machine learning model to match the two pictures.
        - Pros
            - This API is free to use.
        - Cons
            - The API would probably require an upload of the image to some url because the API would require it
            - There would be many steps to query the internet for some credit card image, and then match the image to the picture the user takes.
        - https://deepai.org/machine-learning-model/image-similarity
    - Google Cloud Vision API
        - This API we could use to recognize text from the credit card. We could use the text recognition to get the brand of credit card, and give a list of suggested cards from the given brand.
        - Pros
            - Non invasive - If we use this API to only get card brand, no credit card information needs to leave the client's phone or be saved anywhere.
            - If we give a list of suggested cards, we don't have to worry about an accuracy issue with the API
        - Cons
            - The Google Cloud API has some free credits, but requires payment after
        - https://cloud.google.com/vision
    - **Conclusion: The Google Cloud Vision API seems like a likely solution, because of its ease of use and safety for privacy.**

Real-World Constraints

- If the application constantly monitors the user's geolocation to recommend a credit card based on frequently visited stores, the user's phone battery will drain relatively quickly. Hence, a challenge is to maximize the application usage of the user's geolocation.
- Geolocation might not be accurate if the user is in a small store, hence the store that the application found might not be the one that the user is in. Hence, we have to provide the user with an option to choose the store he is in from a list of surrounding stores.
- There are roughly a thousand credit cards in the US and developing a web scraper to scrape each credit card's benefits is impractical. Hence, an alternative

we found is to use an API that provides each credit card's general information and benefits. The API would require a $8.95 monthly payment that gives us 5000 requests per month.
    - Since there are many types of credit cards in the US, it might be impractical to find each credit card's sample images and to compare them with the image that the user captured. Thus, a real-world challenge is to find a way to efficiently compare the user's image with the list of credit cards available.

Dependencies

Below are the major dependencies the application uses. Smaller dependencies are not included. Please refer to our package.json file with our project for a full list of dependencies.

- React Native
- Expo Authentication
- Firebase

System Architecture Overview

Server Architecture

The backend architecture is a modular system that allows for easy implementation of different server backends that will serve the database and user authentication system. This creates an abstraction that allows clients to make requests for user authentication or other key services without the need for interaction with proprietary APIs. NodeJS will serve as the main server architecture that will be containerized so that it can be deployed on any server platform.

The Firebase platform currently serves as our database and server platform. Its platform services allow for easy implementation and utilization of a **Database System** (for storing global credit card information as well as user credit card metadata and transaction history) and **User Authentication**. This allows for simple implementation of data synchronization between the client and Firebase since it's handled through their platform.

Front-End

The front end phone app will be implemented through a react native cross platform application. It will provide a simple user interface to input a user's current credit card types (Chase Freedom Unlimited, Citi Double Cash Card, etc.). The application will also provide helpful suggestions while the user is within the app for which credit card to use at a particular store and help keep track of how much is spent with each transaction and where.

APIs

Various APIs are necessary for the application to function properly. The user's phone will make requests to APIs to receive location and image recognition information to make appropriate decisions for the user in terms of selecting a particular credit card for a purchase. A Geo-location is retrieved from the phone, which is used in Google's Place Search API. The API queries are periodically made to inform the application of a user's proximity to different stores, gas stations, etc. An image recognition API will also be queried by the application running on the phone to determine what type of credit card a picture resembles. This, along with the credit card information query-able from our server architecture, informs the application of the user's different credit cards which it will use to provide suggestions to the user.

Requirements

User Stories (Sorted By Priority)

1.1. As a user, I can use SpendWiser to find the best card to use based on my geolocation, so that I can maximize my rewards
1.2. As a user who is interested in getting a new card, I can use the application to recommend the best credit card for me to get.
1.3. As a user who does not have an accurate geolocation, I want to choose the store I am currently in from a list of nearby stores.
1.4. As a user, I can create an account to securely store my list of credit cards so that I can transfer my information effortlessly to another phone.
1.5. As a user who wants to monitor my spendings, I can review my transaction history through the application.
1.6. As a user who is security and privacy conscious, I can delete any data the application has collected from me.
1.7. As a user who is concerned about privacy, I want to have the option to edit geolocation settings, either disable, always allowed, or only while using the app (so that I can ensure my location is not always monitored).
1.8. As a user, I can use an account so that my credit card information is secure.
1.9. As a user, I want the option of selecting my card type from a list in the case that my photo of the card cannot be successfully matched or if I feel insecure to take a photo.
1.10. As a user, I want to be able to see all my credit card benefits so that I save time by not having to manually find the information.
1.11. As a user, I can take a picture of my credit card and let the app determine which type of card it is, so that I do not have to manually find it in the list.
1.12. As a user who is privacy conscious, I want the app to let me know that my credit card data is not being tracked after being taken a picture of
1.13. As a user who has disabled geolocation, I want the option of manually inputting a store so that the app can still determine the best card to use.

1.14. As a user who is interested in having less credit cards, I can use the application to find which cards have the most benefit and remove my redundant or overlapping cards.
1.15. As a developer who wants to expand the application, I want the application to be containerized so that it is readily portable and scalable.
1.16. As a user who is interested in learning more about credit cards, I can use the application to browse credit cards and their rewards.
1.17. As a user who is interested in keeping track of their card usage, I can use the application to keep track of my most frequently used card.

Github Repository

https://github.com/ngwx1996/Spendwiser

Technologies Employed

- **Back-End**
  - NodeJS
  - Firebase
- **Front-End**
  - HTML/CSS/JavaScript
  - NodeJS
  - React Native
- **APIs**
  - Google Places API
  - Google Cloud Vision API

Social/ Legal Aspect

- A potential concern of our application is the storage of the types of credit cards each user has. Storing this information is a privacy concern for the user and requires more research into its feasibility or a “Privacy Agreement”.
- Another concern relates to the application finding a credit card based on the user’s provided image. When the user takes a picture, some sensitive information might be captured. Although the application will not use this information, it might feel unsafe for the user. Thus, our team has to potentially create a privacy policy and design an approach to make the application seem more secure.

Acceptance Tests

Since our application is primarily user interface driven, our acceptance criteria will be dictated by the usability of the application. Other than interfacing with our server architecture, each feature needs to be manually tested on a developer’s phone to ensure that our product meets the requirements defined in our user stories. The hope is to get our application in the App Store and

Google Play store relatively quickly this quarter to help crowdsource this acceptance testing to more users by getting feedback on its usability. Additionally, acceptance criteria hinges on our client's understanding of our progress. During weekly meetings a demonstration of our applications progress and usability will be given and any feedback or suggestions will be taken into account for following meetings/demos.

Cost Analysis

The potential costs for the application are listed in the table below:

Cost Item	**Description**	**Cost (USD)**
App Store	An Individual Developer Account is required for distribution via the app store.	$99
Backend System (Firebase)	The system that will run the user authentication and database system in the cloud. The database system is free up until 600,000 writes, 1,500,000 reads, and 600,000 deletes. User authentication is free unless Phone Authentication is utilized (free for 10k a month).	$0.18/100K writes $0.06/100K reads $0.02/100K deletes $0.01/phone verification
Google Cloud Vision API	This API would be used to match a credit card picture to a credit card in our database. The API cost is based upon usage, which is outlined here: https://cloud.google.com/pricing/list?skip_cache=true. The first 1000 requests per month are free.	Depends on usage, $1.50 after first 1,000 requests
Google Places API	The API requires a fee after using the $200 credit (70,000 calls) given by Google.	Depends on usage, $2.83 per 1,000 calls
User Subscriptions	This system would allow users to pay us for some services that would reduce overall cost. Doing so could possibly pay for the costs above.	TBD