CS-4850 Spring 2024
Indy 12 - Student Budget App (FrugalU)
Joshua Schoen, Chase Mo, Logan Getzfred
Professor Sharon Perry
April 1st, 2024

Website : INDY-12 College Student Budget App (indy12-college-resource-app.github.io

Github: INDY12-College-Resource-App/FrugalUApp: FrugalU Mobile Application (github.com)

Lines of Code In Project Currently: Approximately 3,200

Number of Components In Project: Expo, React Native, VS code, Firebase, Github.

Table of Contents

1.0 Introduction

Frugal U is a budgeting app for the common college student. Over or almost half of college students don't know how to budget (depending on the sources you look at). The goal of this app is to be a logging system for common expenses of the college student, things like: Food, School supplies, Tuition, transportation, among other things are predetermined in the app already. All a student has to do is download the app and create an account by entering their name, college, email, password, some financial information like their salary and payment schedule and begin their budgeting journey.

There are four main tabs in the application, "Homepage", "Budgeting", "Resources" and "Profile".

Homepage has all your main information on it, including things like your top expenses, recent expenses, and a graphic showing expense breakdown.

Budgeting is where you'll have the ability to log your expenses in any of the predetermined categories, they should cover all basic college student expenses. Simply click on the category and the "Add Expense" to add an expense to said category.

Resources is where you'll go to learn more information about finance or budgeting. It's connected to a database which currently has 16 entries, but can be dynamically updated any time. Simply go to resources, click the financial advice or budgeting advice buttons to see all the entries laid out in the database for either section. Then, simply click on an entry and you'll be redirected right to it. If you want to learn something new maybe you haven't before on your downtime you'll have the ability to gain some financial/budgeting knowledge anytime.

Finally, the profile tab has all your information you've set once the account was created. Things like your name, email, college, and unique user ID. There are other features that are currently being worked on too, things like budget limits, which are a hard cap on the amount of money you can put into an expense are an example of a new feature which will be available when the final app is deployed to the app store.

2.0 Requirements

The app has many requirements, the following is a rehash of all the requirements like functional and non-functional attributes and other characteristics. The app had a few main points: Be able to log expenses, be able to see the breakdown of expenses, and be able to track patterns of logging. This is the main functionality of the app. The side functionality were things like budget limits, resources, seeing your profile and creating and logging into an account.

2.1 Requirements

1.1 Overview
Frugal U is a college student budget app for young adults. It's an app that won't replace a bank account but plainly an app where users can create an easy budget for the main college expenses.

It will help students budget and learn where their money is going; it will have budget limits created by students and a Pie Chart breaking down each expense as a percentage of their income.

Its UI is going to be so easy to understand that even your grandma could use it, and a database will hold all the unique information the student inputs.

The signup system will use a student's college email address, with full authentication to avoid hackers. However, if someone were to crack into a student's account, they wouldn't be able to access anything incriminating as no financial information like credit or debit card information would be present there. This app is going to be a logging system for expenses, and a bank account will not be connected.

Each expense is going to be broken down into specific college and young adult-specific categories. Some expenses, such as housing and food, is going to be generic, but this is more aimed at college students with categories such as tuition and class-specific expenses like books and Pearson fees.

2.2 Project Goals

The goal of this application is that each student

- Is going to be able to know exactly where their money is going in one easy-to-use app.
- Able to learn and understand budgeting and what makes a great budget.

- Will see a visual representation of their expenses as a percentage of their budget via a pie chart.
- Is going to be able to access financial tips they weren’t privy to without spending time googling.
- Relieve stress and wind down by understanding how their money is spent.

2.3 Definitions and Acronyms

- UI (User Interface): The means by which the user and a computer system interact, specifically the use of input devices and software.

- UX (User Experience): The overall experience of a person using a product such as a website or a computer application, especially in terms of how easy or pleasing it is to use.

- API (Application Programming Interface): A set of protocols, routines, and tools for building software applications, specifying how software components should interact.

- GDPR (General Data Protection Regulation): A regulation in EU law on data protection and privacy in the European Union and the European Economic Area.

- Database: A private entity that is connected to the main application is used for storing information.

2.4 Design Constraints

2.5 Environment

- The app is designed for iOS and Android mobile platforms.
- Requires internet connectivity for syncing data and retrieving updates.
- App is going to be made in Expo.dev

2.6 User Characteristics

- Users are going to be college students who are young adults aged 18-28. They can have varying degrees of financial literacy, they should at least understand what a budget is and what an expense is.

- Users need not necessarily have experience creating a budget, as the app will have built-in tutorials on how the process works.
- Users should have at least one source of income, either an allowance or some kind of job that pays a solid wage. However, even minimum wage jobs are going to be compatible with this app.

2.7 System Requirements

- The system should be able to support access to a large user base.
- Scalable database that can hold hundreds if not thousands of specific user information such as usernames, passwords, and individual budget profiles.
- The System should be able to handle any Android and IOS version, such as Android 10,11, etc.

2.8 Functional Requirements and Actions

- Login and Password Authentication: Secure login process with password recovery options. Users will sign in with their email and a unique password.
- Sign up: Users can enter their college email, first name, and password with college email confirmation.
- Profile Button: This will display unique user information such as first and last name, email address, and the ability to reset passwords or change colleges.
- Display Home Page: Overview of the user's expenses as a percentage of their income stream.
- Expense Tracking: Users can input and categorize expenses.
- Budget Setting: Users can set budget goals for different categories based on their income stream.
- Financial Tips and Education: Access to articles and tips on managing finances tailored to college and financial aid specific to that college.
- Alerts and notifications-to-unseizable notifications for budget limits and financial tips to navigate income easily from their paycheck via income stream. (Weekly, bi-weekly, semi-monthly, monthly).
- Having a button to view custom-set budget limits for user-specific categories.

2.8.2 Nonfunctional Requirements

2.8.2.1 Security

- Authentication and Authorization: Use secure authentication mechanisms (e.g., OAuth 2.0) to manage user access and ensure users have appropriate permissions for different levels of data access.
- Regular Security Audits: Conduct regular security assessments and updates to protect against vulnerabilities, including SQL injection attacks, which are pertinent to MySQL databases.
- Data Privacy Compliance: Ensure compliance with relevant data protection regulations (e.g., GDPR, CCPA) to protect user privacy and data.

2.8.2.2 Capacity

- The app should be able to handle a user base of hundreds or even thousands of active users.
- The app's database should handle hundreds or even thousands of daily queries for active users.
- The app should plan for adequate server resources (CPU, memory, storage) to handle the expected load, with monitoring in place to scale or adjust as needed.

2.8.2.3 Usability

- The app should have an easy-to-use UI interface that a college student should be able to easily navigate.
- The app should have a feedback system such that students can leave feedback for the improvement of the UI
- The app should have accessibility systems in place like text-to-speech, microphone settings, and scalable font sizes for all different types of eye-sights.

2.8.2.4 Scalability

- The app should be able to scale up in the database as numbers climb above the tens of thousands.
- The app should be able to optimize the database such that cloud services could be used for an excess of information.

2.9 Miscellaneous Requirements (External)

2.9.1 Hardware Interface Requirements

- Compatible with smartphones running iOS 11 or above and Android 8 or above.

2.9.2 Software Interface Requirements

- The application will need to be able to connect and communicate (Query information) from a database.
- The application will need to get specific information from websites (information from university websites or other general financial websites).

2.9.3 Communication Interface Requirements

- Utilizes HTTPS for secure communication with the server.

3.0 Analysis

The project had addressed challenges as well as general assumptions that were made in regards to development, all of which are discussed below.

3.1 Challenges

The application should prevent account creation in the event that certain criteria are not met when entering account information. This includes emails that do not match, passwords that are not long enough, and passwords that do not match. The screen should give an error and be unable to progress to another screen automatically.

Similarly, the application should prevent a user being able to log in with incorrect credentials. The screen should give an error and be unable to progress to another screen automatically.

The application should alert the user when they are approaching a spending limit within a category, if limits are set. It should also alert the user when they go over their limit, if applicable.

3.2 Assumptions

The application should be able to run on iOS and Android devices.

Students or general users of the application should have a basic understanding of the functionality of mobile apps. With that being said, the application should be clear and intuitive in its functionality.

The application will require connection to the internet to create or log in to an account and to store data such as expenses inside the database.

The database used for the app will employ firebase for security measures, and must be able to securely store email and password information in correlation with that user's account ID and expenses.

3.3 Risk Assessment

For risk assessment there were many factors:

- Un-stable communication between teammates
- Guidelines that weren't properly communicated clear enough leading to an completely different final product

4.0 Design

The application was to be designed to meet the criteria of a simplistic design and a nice flow of functionality through the different screens. To accomplish this, each member first created a general mockup for the app using Figma. After looking at all three mockups, it was agreed that we would use Chase's mockup as a framework for the app's design.

Regarding simplistic design, the development and design of the app avoided any implementation of over-complicated elements. For example, overloading the home page with every expense purchase rather than separating them out into categories.

When it comes to the flow of functionality,the development process made sure that each button option had a very clear and concise description of what it would do, or what screen it would push to.

The app was to be structured in such a way that makes sense to the user and is very functional for every element. For example, selecting an expense type on the home screen to move to that expense specific screen to see individual purchases as well as add additional purchases that fall under that expense type.

4.1 Design of the Application's Screens

The first page the user is greeted by is the opening splash screen. On this screen, there are two options, create an account and log in to an existing account. If the user clicks create an account, they are presented multiple fields they must fill out to create an account, such as email, username, and income amount. If the user clicks log in, they only need to enter their email and password to move on to their home screen.

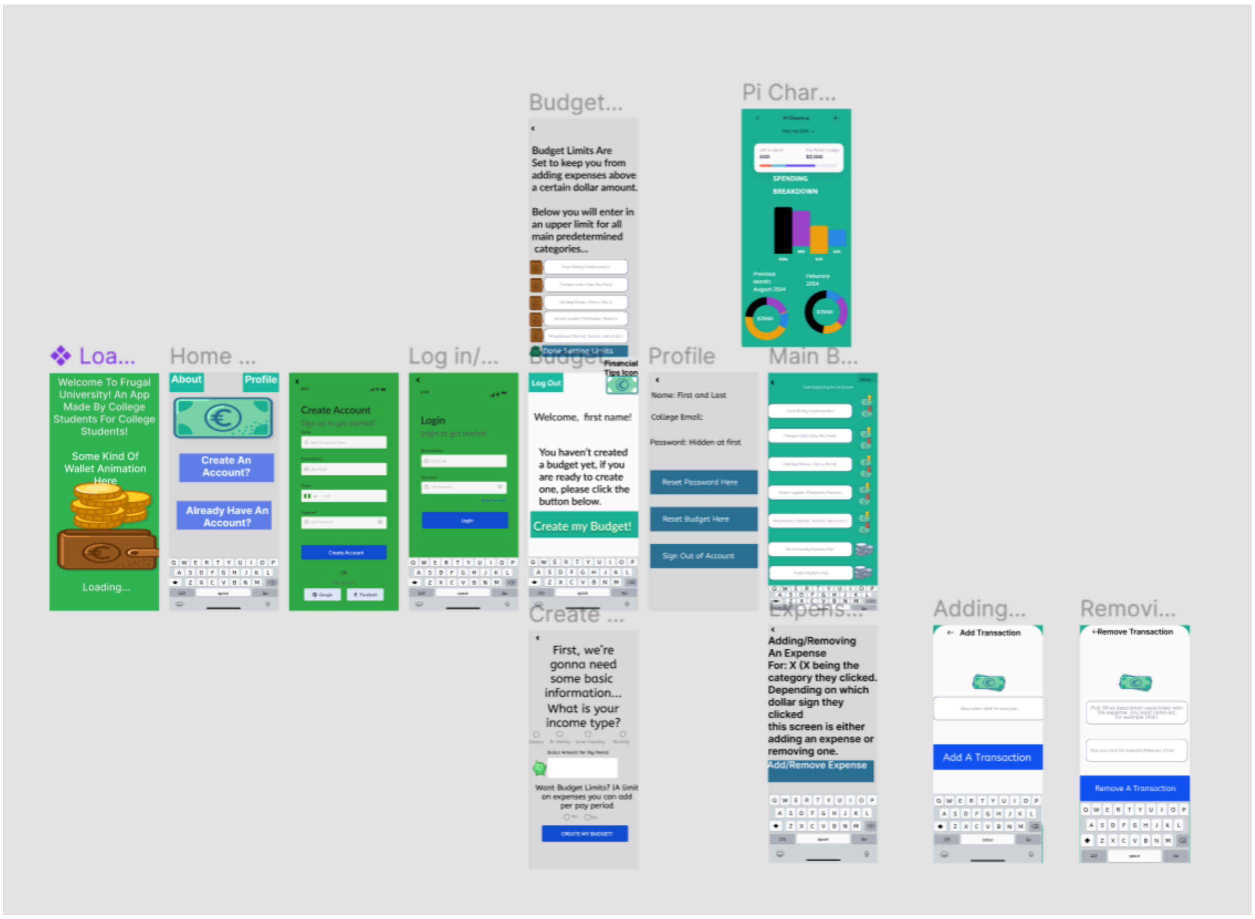
The home page shows the different types of expenses, their limits, and their current amounts for that pay cycle. It also shows a pie chart of the spending for each category. Upon clicking an expense type, the user is presented with a screen that allows them to enter a purchase under that expense.

The resources page, which can be reached at the bottom of the screen, presents the user with multiple different types of available resources for help such as financial aid or budgeting tips.

The account/settings page, which is also reached from the bottom of the screen, displays details of the user's account. This includes, but is not limited to, email, password, account ID, and income amount.

4.2 Mockup and React Native for our Mobile App

This is Joshua's Mockup:


Welcome To Frugal University! An App Made By College Students For College Students!

Some Kind Of Wallet Animation Here

Loading...

Budget Limits Are Set to keep you from adding expenses above a certain dollar amount.

Below you will enter in an upper limit for all main predetermined categories...

Welcome, first name!

You haven't created a budget yet, if you are ready to create one, please click the button below.

Name: First and Last

College Email:

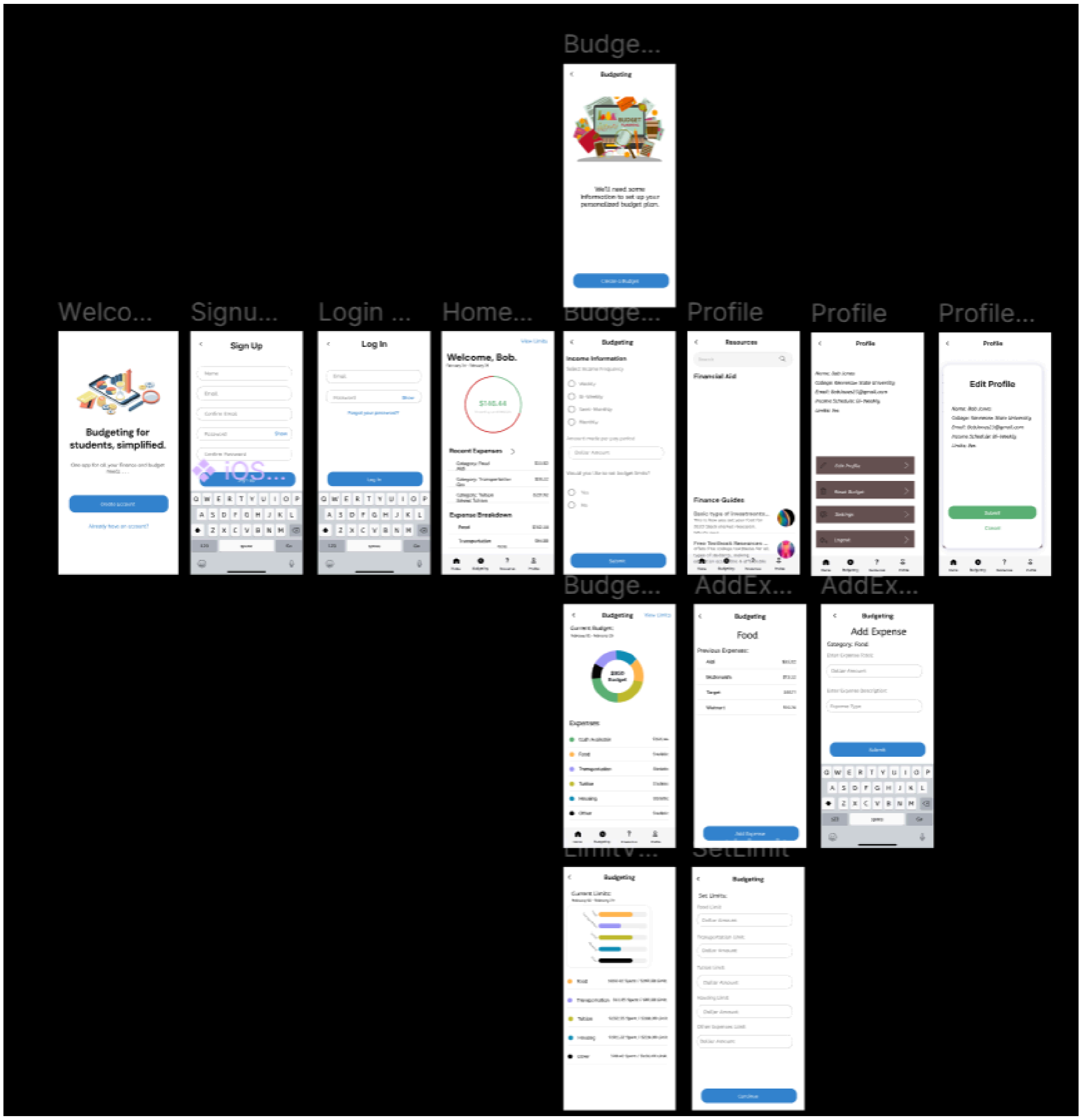
Password: Hidden at first

First, we're gonna need some basic information... What is your income type?

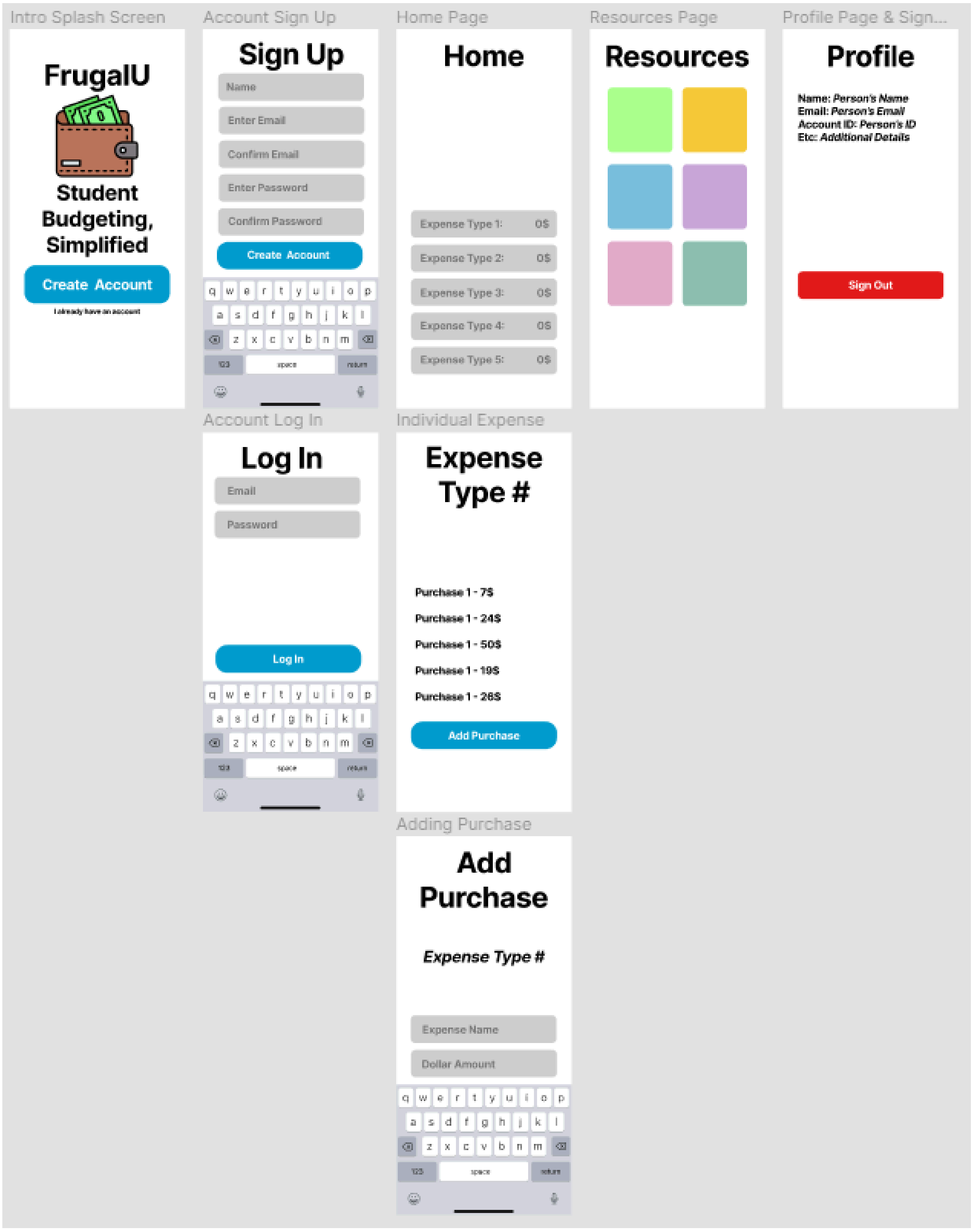
Want Budget Limits? (A limit on expenses you can add per pay period

Adding/Removing An Expense For: X (X being the category they clicked. Depending on which dollar sign they clicked this screen is either adding an expense or removing one.

This is Chase's Mockup:



This is Logan's Mockup:



React Native Proof

Chase created probably 95% of the whole application and he says he has a history of creating applications. Logan worked on the code for username and password authentication.

Joshua Schoen worked on the API, he’s got it to connect and display data to confirm it worked. He used postman and Iheclicked on the fetch to confirm it works. He’s got the Resources tab to display two buttons that’ll open up a navigation to “Budgeting advice” entries and a separate layout within resources for “financial advice entries”. He’s got the code so the user can click on each entry and it’ll redirect them to the article/information in the database. The layout of each entry has a title, description and a picture of the article.

5.0 Development

As a team, everyone had our roles to play for the assignment. Joshua was one of two documentation team members and Chase was the developer. Everyone got together almost every week to discuss plans and how exactly they were going to create this budget app. Since it was Joshua’s idea, the team decided he would write all the requirements for exactly the kind of app it would be, with Logan working to create the design and Chase to develop it.

The requirements were very straight-forward, it would be a budget app for college students, this means that it needed to be exactly what a college student would want. The team all got together and thought, what do college students spend money on? They talked about the concept in the meeting and then Joshua would be the one to write all the requirements functional and non-functional. Joshua came up with a bunch of predetermined categories for the app: Tuition, transportation, food, housing and other miscellaneous costs.
He also determined non-functional requirements as high scalability, a dynamic and fast database and useability of the app. This being a full-stack application, the team needed to have a fully functional and strong database to be the bedrock for the app.

It wouldn’t just be an app with college expenses, but an app that the everyday college student would use. It would have a structure similar to a logging system, so they needed a database to store everything. Joshua and Chase talked about the different ways to have a database, Chase saying that he had experience creating mobile apps in the past and thought of using firebase for the database.

Firebase had an advantage here, being a database that's dynamic and very scalable with a high query speed was perfect for an app like this. In a logging app the user will be entering in expenses for a specific account and manage those expenses whenever the app is active. Chase also said it was very easy to use and he did an ample amount of research on it and the firebase was perfect for our project. It also had built in security features which wasn't necessary since there's no hyper-sensitive information being logged but was still a good feature in case later down the line in case the team wanted to use fintech to connect a bank account to the app.

Along with the database, the team needed a framework for the app. Chase, as a developer who has made a few mobile applications in the past, said to use Expo and react native as a framework, and javascript and typescript as the programming languages. Since it was a mobile app, the documentation members Joshua and Logan also had to learn Expo, React native as well as learn or brush up on typescript and javascript.

After Joshua and Logan finished the requirements and design documents respectively, they talked and agreed to the requirements and design structure.

They each also had to create mockups in an app called Figma. Out of all the created mockups, Chase's mockup visually and functionality looked the best, so the team decided to go with that. After meeting again to discuss prototype projections,

Chase, being the developer, got started with the app. He created a repo in the organization github the team had made earlier. He consistently provided updates to the team regarding progress and new features he planned on adding. After a few weeks of hardwork the app's skeleton looked great!

The app had the ability to create an account and login to the account, and had all the features discussed by the team in all of the meetings as a baseline. The team then got together to discuss other features for the app. There were three current tabs upon the creation of an account: Homepage, Profile, and Budgeting the tab where the user would enter expenses for the predetermined categories. Logan in the meeting suggested there would be a fourth tab called "Resources" this would use an API that would fetch from a database full of entries of information for the User.

It was decided that Joshua would be the one to program the API, and Logan would work on password authentication and the creation of "budget limits".
With the prototype presentations approaching, Chase worked more on the functionality of the app, while Joshua studied up on APIs and how to create them.

He and Chase would talk about a database called "strapi" which was perfect for the api. This server would hold entries of articles of information that the user could click on and go to learn about. Chase went ahead and created the skeleton for the "Resources" tab, the user could click on the tab but nothing would appear.

In the meantime, Logan worked on some functionality for password authentication, the group then went onto prototype presentations and after the presentations Joshua and Chase talked about implementing the API. Joshua started working on the section for resources and first learned how to fetch the API using youtube videos and the react native website, also about setting up the strapi database and how to fill it with entries. After a while he set up the database and implemented a fetch api call to the database, the app could now fetch all the entries he put into the app.

He looked up how to display the entries on a screen using something called "stack navigation". He eventually figured out how to display entries of information in the resources tab. It was decided that the entries would be broken up into two separate screens connected to the resources tab. One screen would have Financial advice entries and the other would have budgeting advice entries.
He and Chase then started talking about the final stages of the app, while Joshua was doing the API, Chase had been working on finalizing the homepage tab and updating the profile tab.

Joshua completed the API on the 12th of April and it looked great! The layout for each category is a bunch of clickable divs that are stacked and display the title, description and a picture of the article or information piece.
The resource now had interactive budget/financial advice tabs which the user could click on and directly go to the article referenced!

Logan began working on the budget limits in the app, which were the final feature. He started to work on them on the 14th of April.
While Chase worked extremely hard on the homepage and fixing extra bugs in the app like the firebase not communicating with the homepage, he ended up completing everything else remaining for the app by the 26th of April!

Logan couldn't finish the budget limits in time, so the group decided not to add it to the app. Joshua himself will add the budget limits to the app in early May.

Thus, Frugal U was born! A fully functional simple mobile budgeting app for the average college student. The group worked really hard to make this app a reality with Chase doing the majority of the work, Josh completing the API for the app and Logan working on some authentication for the app!

6.0 Version Control

We are using github for our repository and version control. Joshua Schoen created the organization page, and Chase Mo created the repo. Chase made multiple different iterations of the app and as such has created multiple version iterations on the github page.

Both Joshua Schoen and Logan Getzfred downloaded the github code versions recommended by Chase for testing as well as editing to implement new features into the code.

Logan's first pull request was in regard to account creation credentials, specifically making sure emails matched, passwords were long enough, and that the passwords matched as well.

Joshua has finished adding API integration and sent a pull request to put that into a new version of the code on github. Similarly, Logan worked on the limits part of the app but couldn't get it done in time.

7.0 Applications Testing

7.1 Functional

So for testing of the functional requirements, we've been constantly downloading the latest version of the project using github and using "Expo Go". This is an app that simulates what the user would see if they were to download the app. It gives you errors and real time feedback using the development mode. This has made it real easy to test, all we have to do is see all the functional requirements and check if they work on the app.

Most of the functional requirements actually were implemented and are currently in the final stages of implementing. Things like: A functional login screen, with password authentication and signup are fully functional. The main features of the app like adding expenses and budget tracking look great and are also implemented. Only a few things are left to test like the API, the homepage once created and budget limits.

7.2 Non-Functional

Chase was the main one testing the app's non-functional requirements. Being the firebase admin, Chase constantly updated and checked all basic features of the firebase authentication system and ability to retrieve usernames and passwords for multiple users.

Josh tested the strapi api's ability to fetch data, while a large amount of entries hasn't been tested, the dynamic ability of strapi allows the server admin to update the stack allowing the admin to add and remove entries that the user will see. This means that if there is a problem with fetching large sets of entries for the user to interact with, they can be removed at any time if they present a problem to the user.

7.3 UX/UI

UX/UI was tested using the same Expo Go app. Now, it's not fully functional, sometimes it'll throw a firebase error, but most of the functionality like adding expenses/viewing your budget pie chart work and look great! All the tabs for all the features appear on the app footer and are clickable by the user. Adding an expense is also very easy as well, everything is very user friendly that someone who's never used it before could figure out how it works in a few minutes.

7.4 Test Plan

Our test plan was to constantly down the latest versions of the app from the github and run it in Expo Go. We used development mode along with Go versions to test different SDKs of the app. Chase used developer mode since he knew far more about app developer than Joshua. Joshua used the "Go" version which is an easier way of having someone with not as much react native knowledge can still test the app. This was the easiest way to effectively test each part of the app by simulating the user experience of using every single feature.

7.5 Test Report

INDY-12 Test Report	Pass/Fail
Login and Password Authentication	Pass
Sign Up	Pass
Profile Button	Pass
Display Home Page	Pass
Expense Tracking	Pass
Budget Setting	Pass
Financial Tips and Education	Pass
Alerts and Notifications	Fail
Custom-set Budget Limits	Fail

8.0 Conclusion

This document was a combination of requirements, design document, narration of the conception and current development of the application. As we work towards finishing the application, we'll add more content such as a possible improved website, a deployment diagram, an updated test report, and of course feedback from Professor Perry. I'd say the app is ninety to ninety-five percent completed, all we're missing is the budget limits and some more authentication like email-authentication.

We're also going to update the code count once it's all completed and maybe some more functionality for the resources API tab. We'll also decide on a plan for testing non-functional attributes, as of right now we don't have a plan for doing that. We will also add the project plan (if necessary) to the start of this document. We're also going to add the final presentation as we've gathered feedback from our prototype presentation. We hope you like this rough draft and we look forward to the feedback!