

College Capital

CS307 Design Document

**Muhammad Bokhari, Jeremy Chen, Justin Chen,
Charlie Newell, Matthew Story, Jethro Zhou**

Index

- Purpose	2
Functional Requirements	
Non Functional Requirements	
- Design Outline	7
Components	
High-Level Overview	
- Design Issues	10
Functional Issues	
Non Functional Issues	
- Design Details	14
Data Class Diagram	
Data Classes Description	
Sequence Diagrams	
UI Mockups	
Database Design	
API routes	

Purpose

College students today have more to manage than ever with rigorous course loads, hectic social lives, and an increasingly present sense of independence. Along with these increased responsibilities comes a larger financial burden, one which can cause stress, forcing students to deal with unnecessary chaos while attempting to juggle various sources of income and expense. Things like meal plans, tuition payments, and scholarships are just a few examples of what an average student has to keep track of - on top of everyday expenses and profits. Students often end up using several different applications to keep track of all these things, which leads to a disjointed and jarring financial management experience where mistakes are bound to happen.

College Capital is a web application that alleviates these burdens by aggregating students' financial information into one place, allowing for quick access to **all** of their financial information. While this is an extremely saturated domain, none of the competitors are specifically designed for college students. Furthermore, the competition lacks several key features that we intend to provide in our app, such as predictive spending models and tools for students who are trying to launch their own businesses. For example, the most popular app in this space, Mint, simply acts as an information hub for finances. It offers reminders, credit scores, and a full financial overview, but no features specifically for college students or predictive spending models. Another competitor in this space is PocketGuard, which specializes almost exclusively in budgeting with set amounts. While they exceed in this area, they completely lack other features our application will provide.

As a final product, College Capital will be a one-stop shop for a college student's financial needs, featuring seamless integration with their existing services in order to provide a holistic view of their finances. Furthermore, as seen in the user stories and throughout this document, we plan to include several features that will help our users manage their income and expenditures - features that will make our app much more useful than the competition.

As a group of college students ourselves, the everpresent pressure of financial security is one that we are all intimately aware of, hence why we decided to work on this project. We felt that options available today didn't quite match what we would want out of them, so we decided to make our own!

Functional Requirements

1. Users can login or sign up for the app and create different pages for their finances.

As a user:

- a. I would like to sign up for an account.
- b. I would like to be able to use my google account to sign up for an account.
- c. I would like to reset my password if forgotten.
- d. I would like to be able to login and logout of my account.
- e. I would like to transfer funds between different types of accounts.
- f. (If time permits) I would like to add a profile picture.
- g. I would like to be informed of any unusual activity.
- h. I would like to deactivate my account.
- i. I would like to reactivate my account.
- j. I would like to have separate categories for my finances (checkings, savings, etc).

2. Users can check their funds and set alerts to notify them when their expenditures exceed a certain limit.

As a user:

- a. I want to see how much money I have in my account.
- b. I would like to establish a limit on my expenditures.
- c. I would like to be notified if I surpass my limit.
- d. (If time permits) I would like to request money from other users.
- e. I would like to view my previous transactions.
- f. I would like to be able to visualize my daily, weekly, and monthly expenditures.

3. Users can make use of additional functionalities to manage their finances.

As a user:

- a. I would like to deposit funds into my account.
- b. I would like to withdraw funds from my account.

- c. (If time permits) I would like to create memos related to particular finances.
- d. I would like to create different categories for my expenditures.
- e. (If time permits) I would like to donate money to different organisations.
- f. (If time permits) I would like to link my Paypal account to my account on this app.
- g. I would like to create a spreadsheet of regular daily, weekly, and monthly expenditures.

4. Users can use this app for business to pay and create financial reports and keep business expenditures in check.

As a user:

- a. (If time permits) I would like to create a group for myself as well as my employees.
- b. (If time permits) I would like to repeat certain transactions.
- c. (If time permits) I would like to create profile pages as an administrator and an employee.
- d. (If time permits) I would like to create finance reports.
- e. (If time permits) I would like to correct mistakes in the payroll.
- f. (If time permits) I would like to be notified when any changes are requested.
- g. (If time permits) I would like to link several accounts to form a business platform.

5. Users can look up their college funds and meal swipes in order to plan their school-specific expenses.

As a user:

- a. I would like to see the number of meal swipes (or cash amount) I have left.
- b. I would like to see the amount in my other school-specific accounts (boiler express for a Purdue student, for example).
- c. I would like to plan my finances for the week using a calendar.
- d. I would like to add and remove funds from these accounts.

Non - Functional Requirements

1. Client Requirements:

As a developer:

- a. I would like this application to be usable across all browsers.
- b. I would like this application to be accessible on mobile browsers, as well.

2. Server Requirements:

As a developer:

- a. I would like the server to be able to parse incoming data and store it in the I would like the server to communicate between clients and the database.
- b. I want the server to be able to handle ~100 concurrent users.
- c. I want the web application to respond to all user requests within 300-400 ms.

3. Usability:

As a developer:

- a. I would like the application to be user-friendly.
- b. I would like the application to run on native mobile apps (if time permits).
- c. I would like the application to be responsive and to be minimalistic, style-wise.

4. Security:

As a developer:

- a. I would like users to be able to authorise others to access their account.
- b. I would like to encrypt passwords when saving them in the database using google firebase in conjunction with our own algorithm.
- c. I would like to restrict users' access to information based on their authorization level.

Design Outline

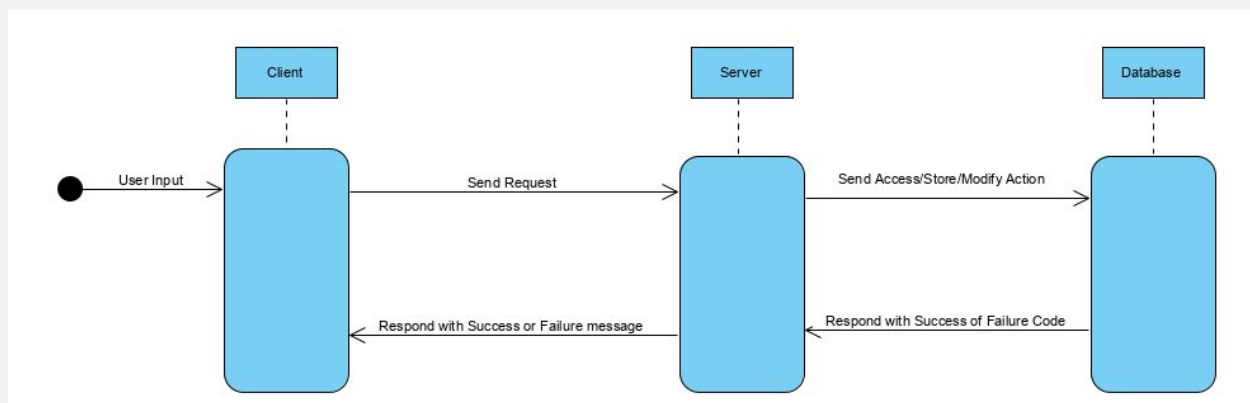
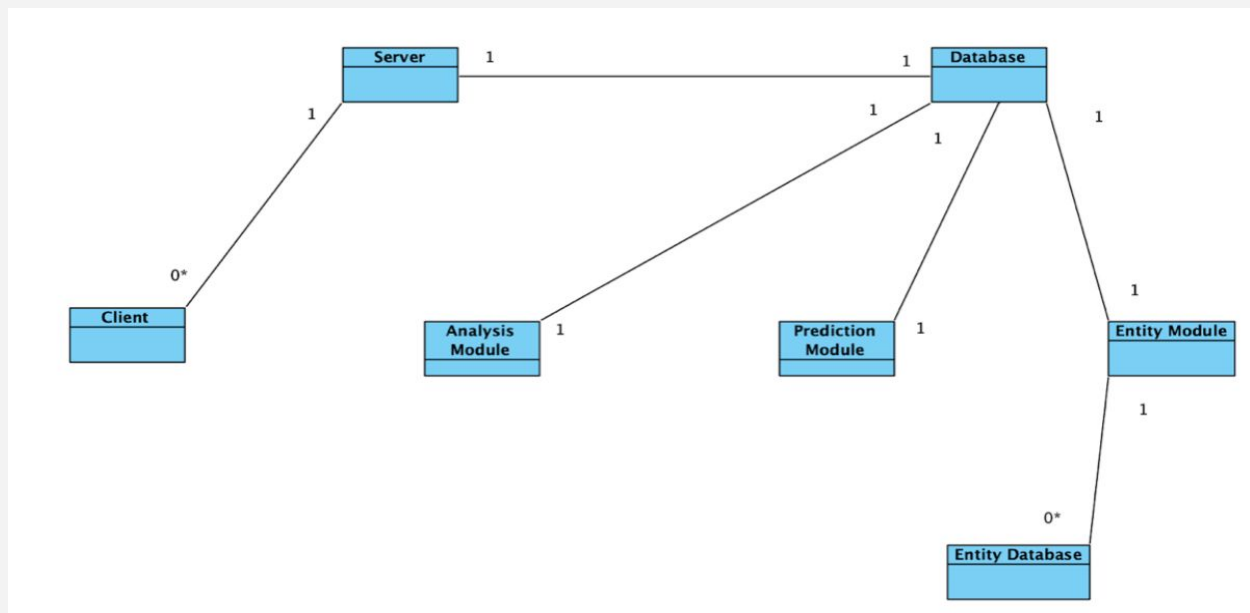
Components

1. Database
 - a. Will store all necessary data (user credentials, financial data etc.).
 - b. Hosted on Google's Firebase platform.
2. Website
 - a. UI for users to interact with the application, such as by viewing balances and graphs.
 - b. Implemented via React.js, which is developed by Facebook.
3. Visualization Module
 - a. This module will utilize various APIs in order to display financial data that will be included on the website.
 - b. Allow users to add/remove instances of the module.
4. Predictive Spending Module
 - a. This module will access bulk user transaction data, analyze it, and use the analysis to develop predictive spending models that will be stored in the database.
5. Suggestive Spending Module
 - a. This module will work together with the predictive spending module as well as user-input information about budgets in order to provide suggestions on decreased spending while maintaining current standards.

High Level Overview

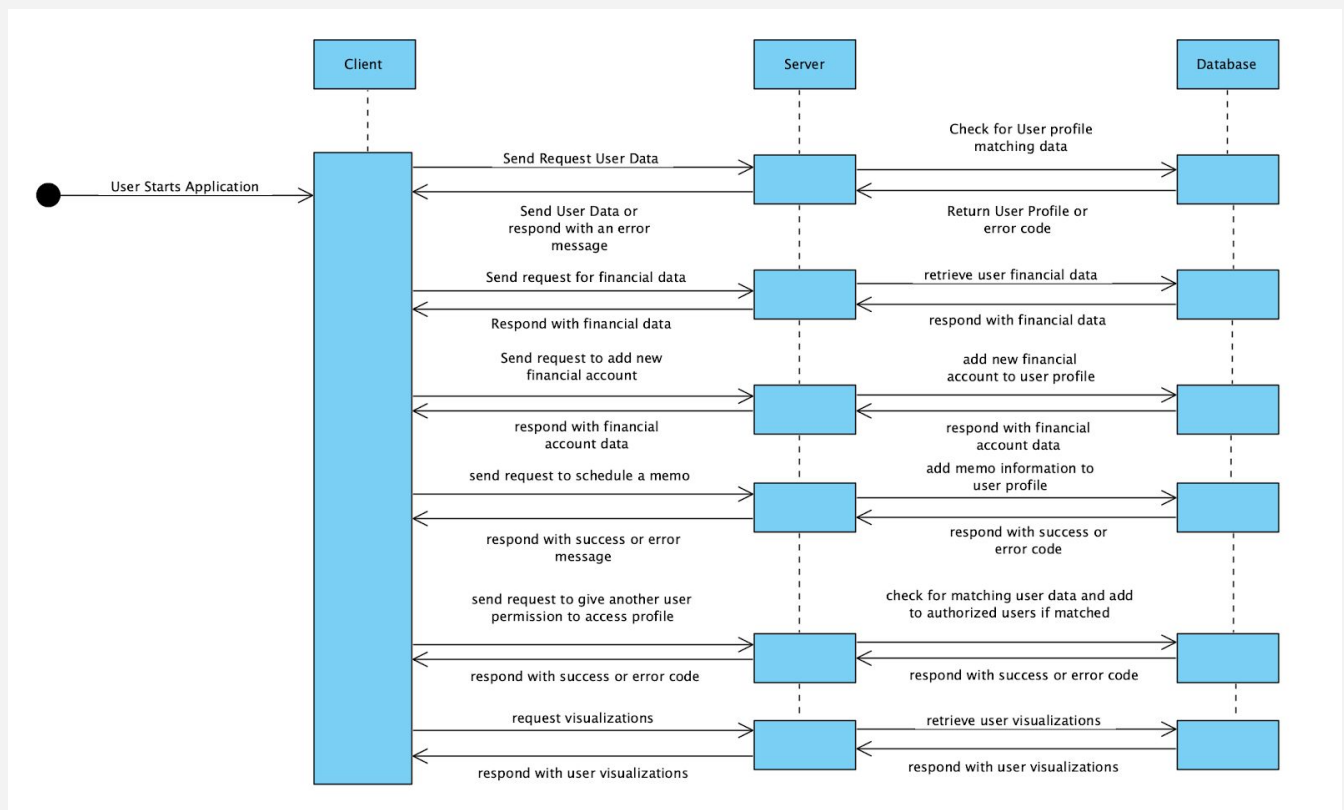
This project will be a web application that allows users to add to, remove from, plan out, and manage their personal finances. This application will use the client-server model in which one server simultaneously handles access from a large number of clients using google's "firebase database". The server will accept client requests, store data in the database and respond to the clients accordingly.

High Level - UML Diagrams



Sequence of Events Overview

The sequence diagram below shows interactions among clients, the server, and the database. The sequence begins when the user starts the application by accessing the website. As the user logs in/signs up, the client sends a request to the server which, in turn, sends a query to the database to store the data. The data will be encrypted/decrypted (passwords etc.) as needed, and the server can then handle the request by logging the user in. After being logged in the client can send additional requests to the server such as managing profile data, adding funds, and authorising different users. To respond to these queries the server sends the queries back to the database to update the data. The result is then returned to the client.



Design Issues

Functional Issues:

1. What data should the user be required to provide in order to create an account?
 - Option 1: Username, password
 - Option 2: Username, password, bank account
 - Option 3: Username, password, bank account, phone number, social security number

We chose option 1, as it isn't necessary for users to provide us with extensive financial information upon first creating their account, and that information can be gathered later should the user decide to add additional financial information. Asking for a bank account number or social security number at first would be an unnecessary barrier for account creation, and would likely decrease user interest and, as a result, the user base. Additionally, options 2 and 3, while possibly more secure given the information provided, would also require us to store, retrieve, and manipulate far more information, potentially impacting overall performance.

2. What kind of financial information should the web application support?
 - Option 1: Just bank balance, including checkings/savings account
 - Option 2: Just school currencies, such as number of meal swipes, dining dollars, etc...
 - Option 3: Both bank balances and school currencies
 - Option 4: Bank balances, school currencies, and other financial information that can be reasonably implemented

We chose option 4. We want to make the application as flexible as the user wants it to be, and being able to access many different types of financial data would offer the greatest utility to users of the application. The other options may have the

benefit of being more focused however, limiting our application to just bank balance or school currencies would defeat the purpose of the “all-in-one” website to more easily keep track of one’s financial data, that we plan to achieve.

3. How should the user’s financial information be displayed?

- Option 1: Exclusively through detailed graphs (bar graphs, histograms, etc)
- Option 2: Exclusively through numerical data
- Option 3: A combination of the above

We chose option 3 in order to make the app accessible and novice-friendly while remaining informative of all facets of a user’s financial information. Choosing to provide both numerical data and visualizations about a user’s finances may limit the level of detail we would be able to achieve should we focus on one or the other, as the other two options do, however, providing both gives our users a holistic view and provides something for both visual and textual based learners.

4. How should we predict user financial information?

- Option 1: Have user provide their best guess on personal future expenses
- Option 2: Predict user financial information using a self-developed algorithm

While nobody understands their financial situation better than themselves, humans are prone to error, especially when it comes to anything involving extensive numerical calculation. This fact leads us to choose option 2, which will greatly simplify the process for the user and provides a guarantee on correctness given accurate information. Furthermore, by writing our own algorithms for user spending, we can integrate these methods with our other algorithms that provide budgeting suggestions, improving the overall app experience. Additionally, when faced with their financial future, human-only input would likely be very biased. Using an algorithm instead allows for far more concrete, factual information

Nonfunctional Issues:

1. What front-end languages should be used to implement our web application?

- Option 1: Angular
- Option 2: React.js
- Option 3: Vue.js

We chose option 2. React.js is an open-source declarative JavaScript library that is supported by Firebase, which is the platform we chose for our back-end implementation and database. Additionally, it allows assimilation of javascript using JSX, making it easier to learn and implement. React.js is also very popular, and therefore would offer more support should we run into issues - it is also very beginner friendly. This is in stark contrast to the steep learning curve of Angular as well as the relatively new community surrounding Vue.

2. What language should the back-end use?

- Option 1: Python
- Option 2: Java/Javascript
- Option 3: PHP

We chose option 2 as most of us have extensive experience in Java as a result of Purdue course work. As a result, as a team, we feel confident in using Java. Java/Javascript also provides different frameworks which will help us implement efficient functions. In addition, both PHP and Python aren't as easy to port to mobile apps as Python is.

3. Which Database should we store all the users' data in?

- Option 1: MongoDB
- Option 2: MySQL
- Option 3: Firebase

We chose option 3 because our project is already deeply integrated with Firebase on the backend, so storing the information in their databases allows for seamless workflow and easy manipulation of information. This will greatly simplify our implementation as well as our logistics given that simply having everything in one place allows for quicker and easier communication and understanding. Furthermore, while both MongoDB and MySQL are excellent databases with several rich features, their functionality stops in that realm - databases. Firebase provides a wide-ranging selection of features that the other two simply don't have or support.

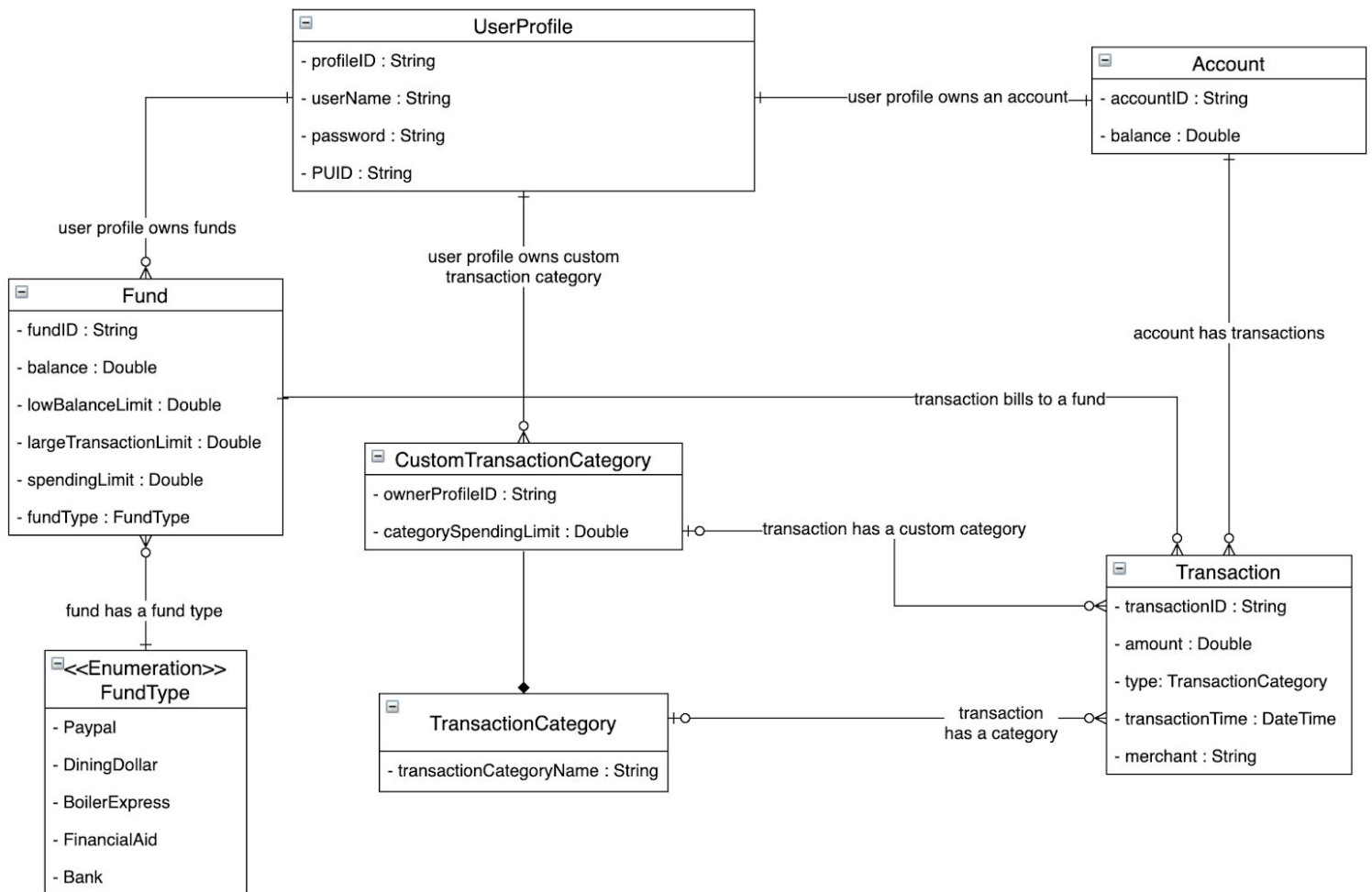
4. Which API should we use to access bank accounts?

- Option 1: Paypal API
- Option 2: Venmo API

We chose Option 1. Paypal provides a very nice API which ensures secure transactions. Using this API will help with the main functionality of the app, moving finances, without the hassle of creating another network and getting it approved by each bank. Paypal is a proven and credible app trusted by many that can simplify the process of facilitating transactions. Venmo is a subsection of Paypal, so utilizing the larger parent API provides us with more features than just the Venmo API can or would.

Design Details

Data Class Diagram



Data Classes Description

Our data class design focuses on objects that will be used in the Java backend and domain in our database, since our front end languages do not support classes. They represent the data structures that we plan to utilize in order to compartmentalize information. As a side note, the API module will be handled by Firebase and therefore is left out of context.

UserProfile

- Represents information related to a user
- Holds user login information
- Created during initial user login

Fund

- Represents user's monetary accounts in financial entities
- Contains the available funds in those accounts
- Allows user to set spending limits
- Created by user after initial login

FundType

- Represents the type of financial accounts (checkings, savings, etc)
- Can be modified by user
- Created as enumeration during development

Account

- Serves as a centralized hub to aggregate all expenditures and funds
- Created during initial user login
- This is the building block of the class hierarchy

Transaction

- Represents debit and credit activities
- Contains amount and transaction date time
- Created by user manually or parsed from spreadsheet uploaded by user

TransactionCategory

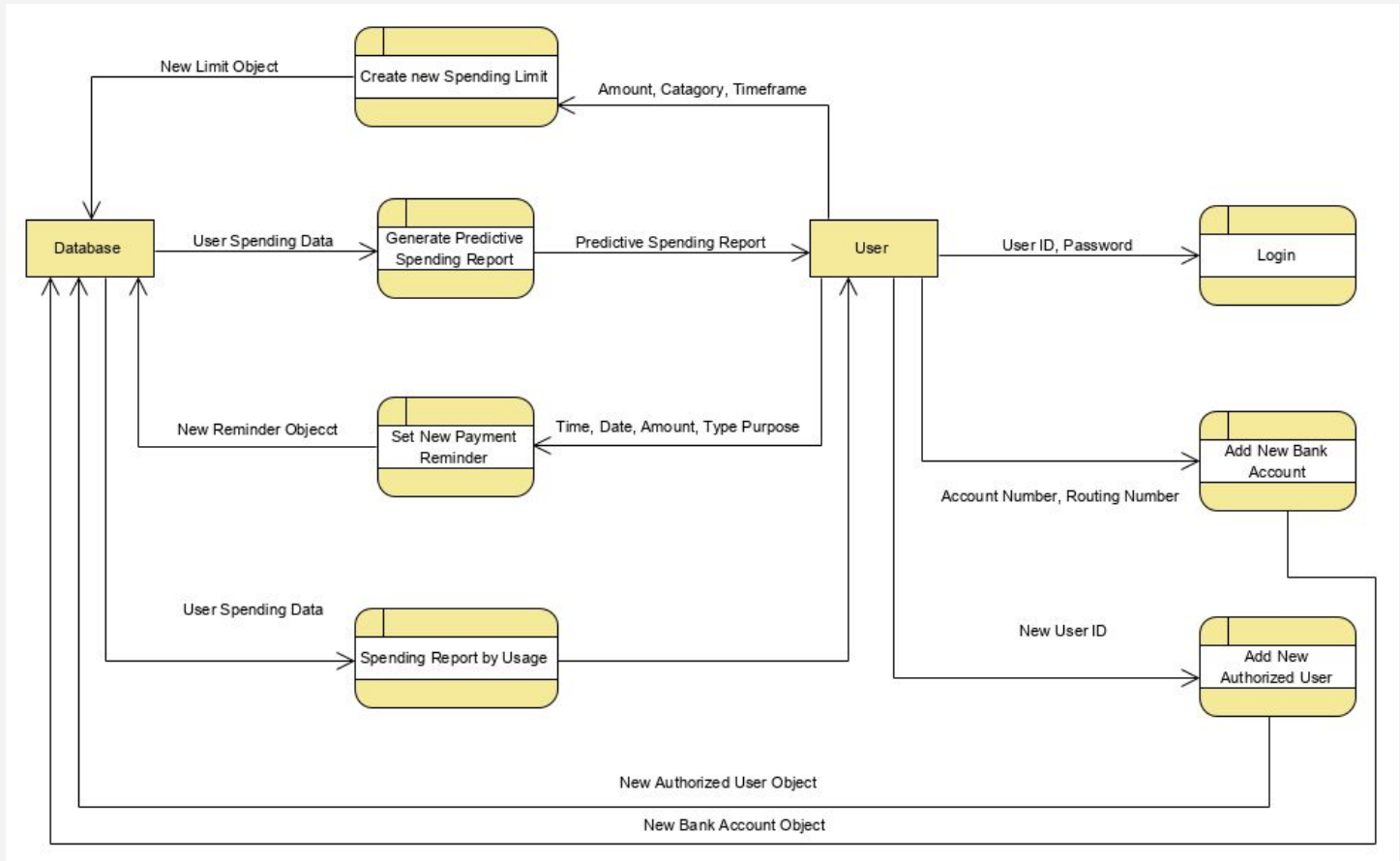
- Represents initial transaction category available for user to assign to transaction
- Created during development

CustomTransactionCategory

- Represents customized transaction category
- Contains customizable spending limit information
- Can be created by user after initial login

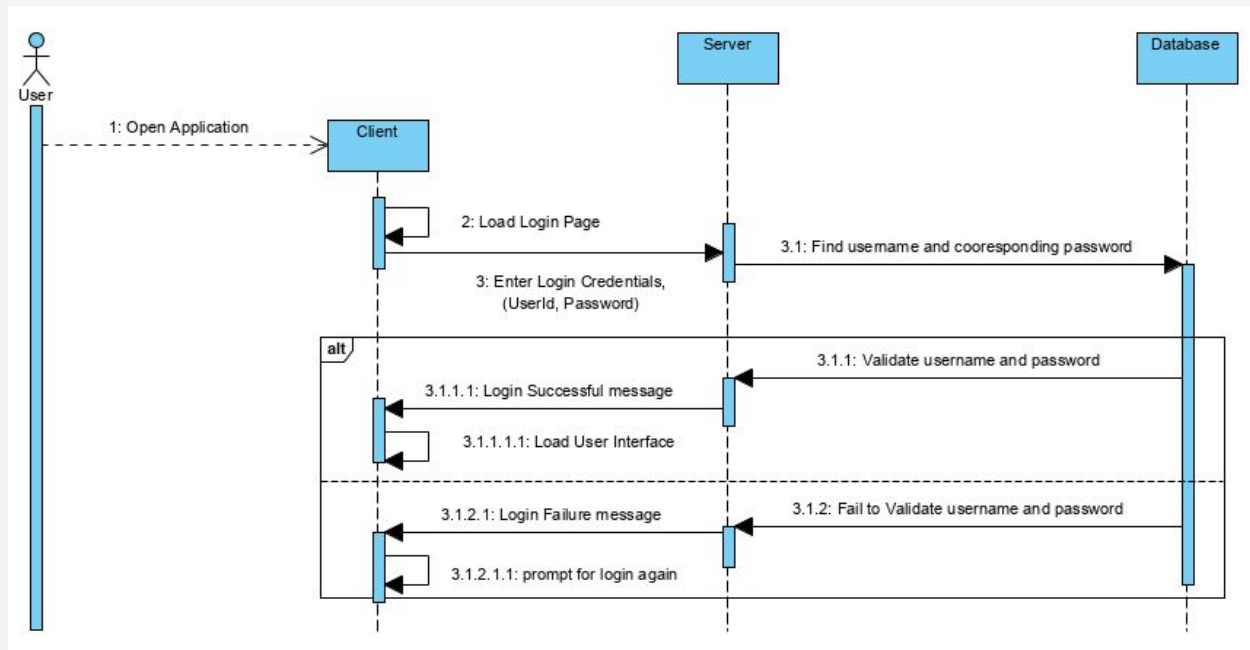
Data Flow Diagram

Data Flow between Processes and Entities



Sequence Diagrams

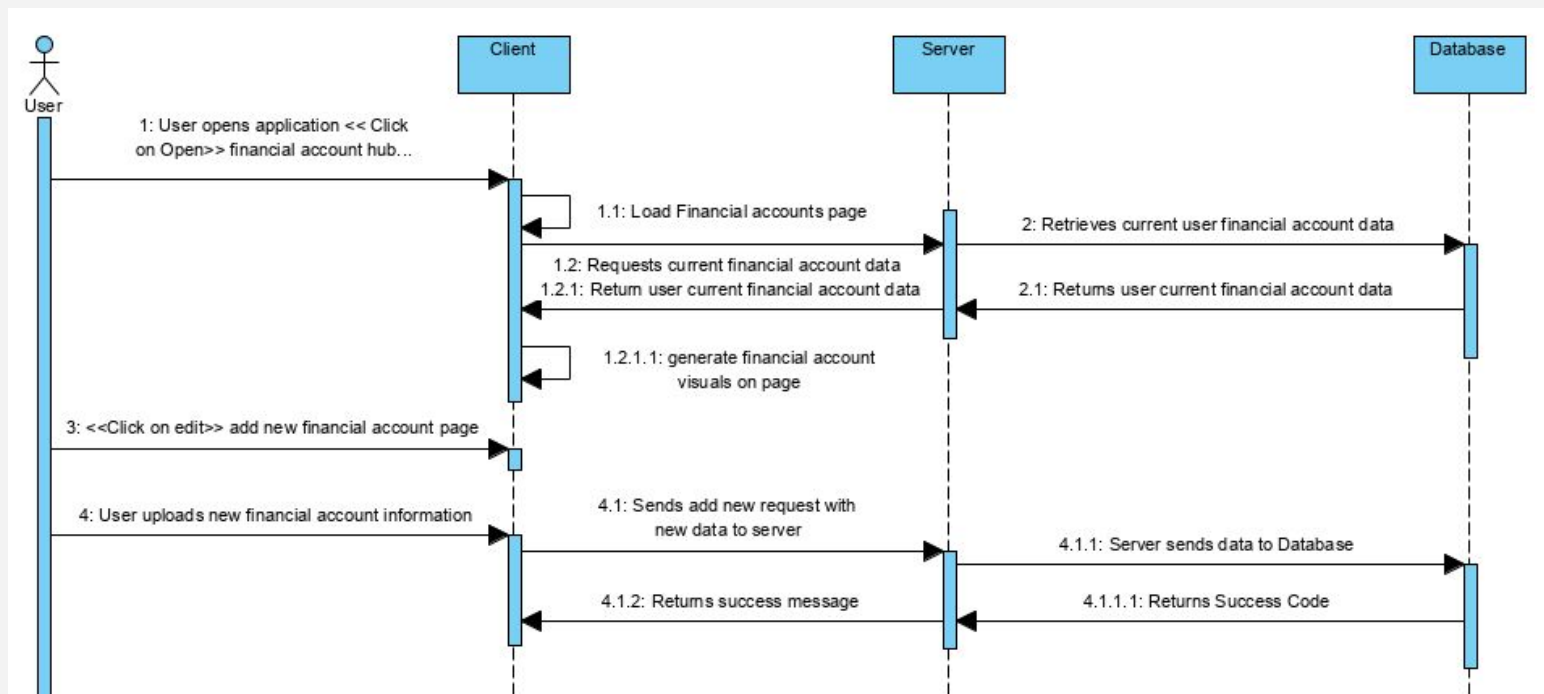
User Sequence for Login



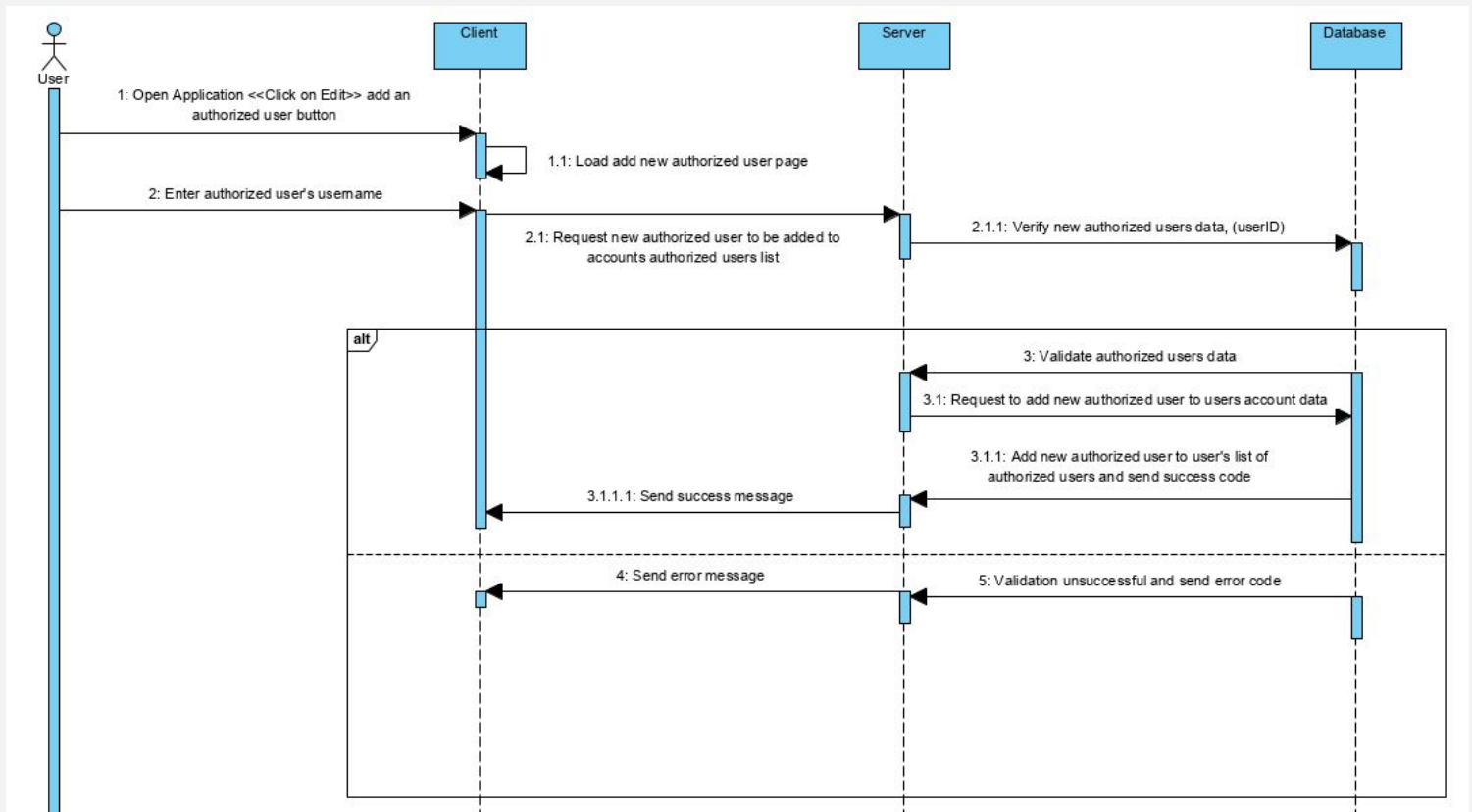
The above demonstrates the sequence of events when a user logs in, including the interactions between the client, server, and database.

User Sequence for Requesting Financial Information

The above demonstrates the sequence of events when users request additional financial information that isn't immediately shown from our database. Similarly, it also shows the sequence of events when users add new financial information.

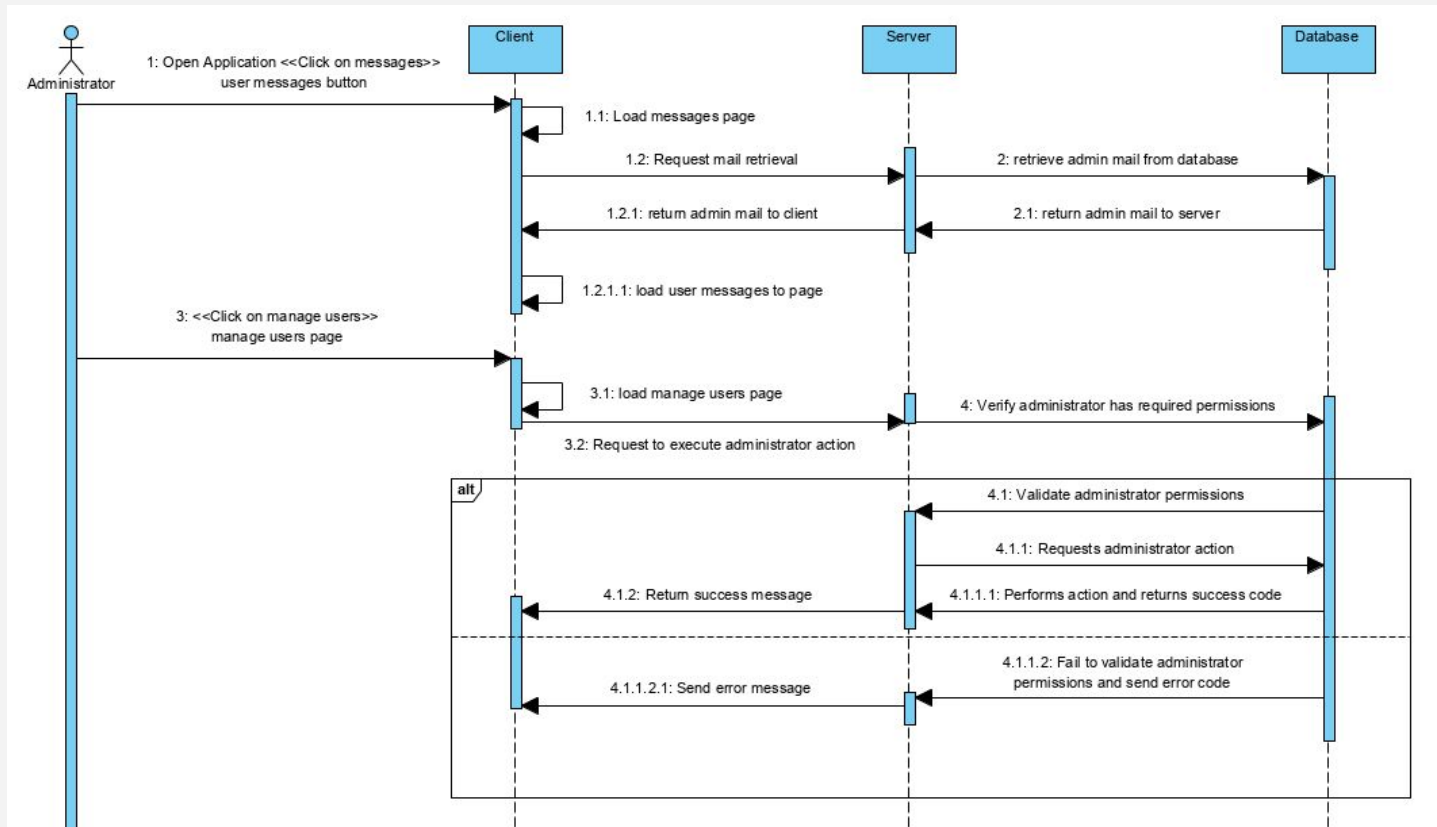


User Sequence for Adding or Interacting with Authorized Users



The above demonstrates the user sequence for adding or interacting with authorized users on the primary user's account.

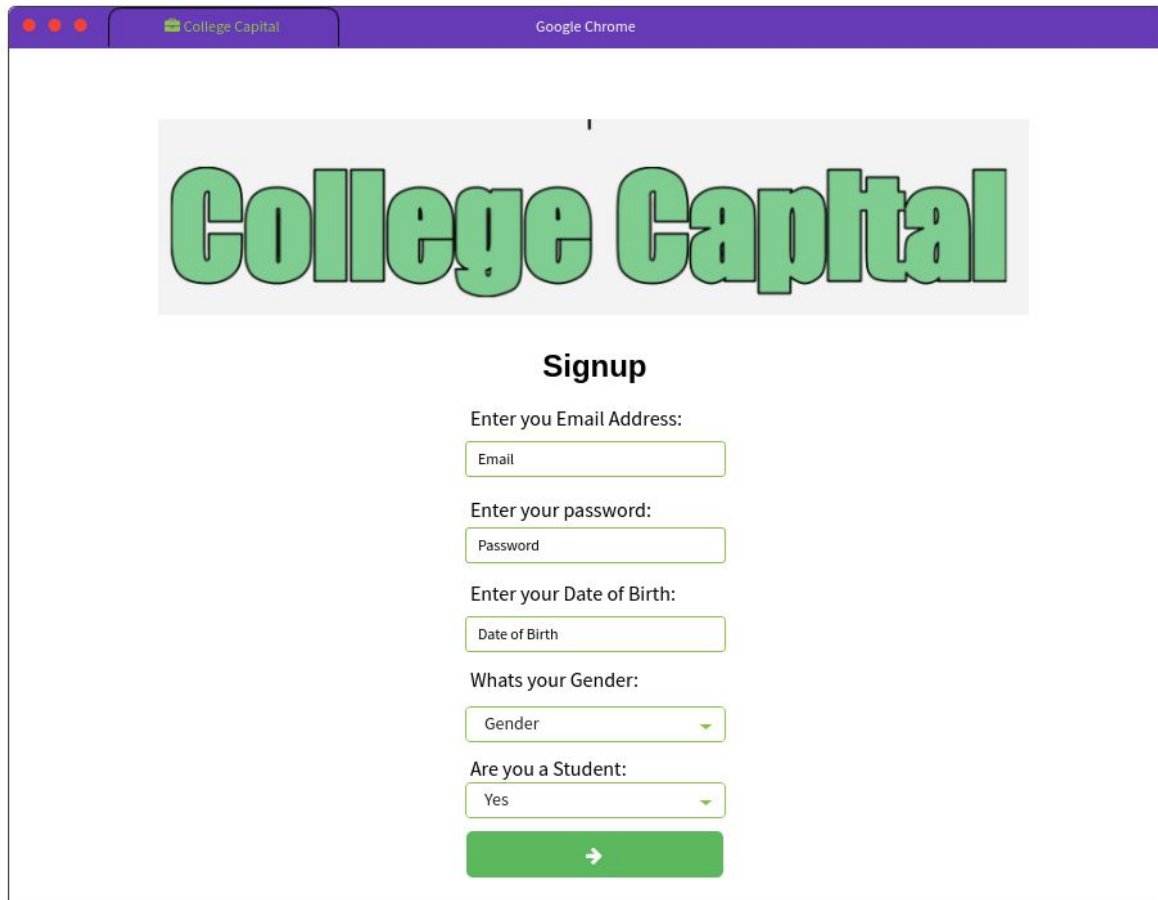
User Sequence for Interacting with Customers



The above shows the user sequence for providing support to customers from an administrator's perspective.

UI Mockups

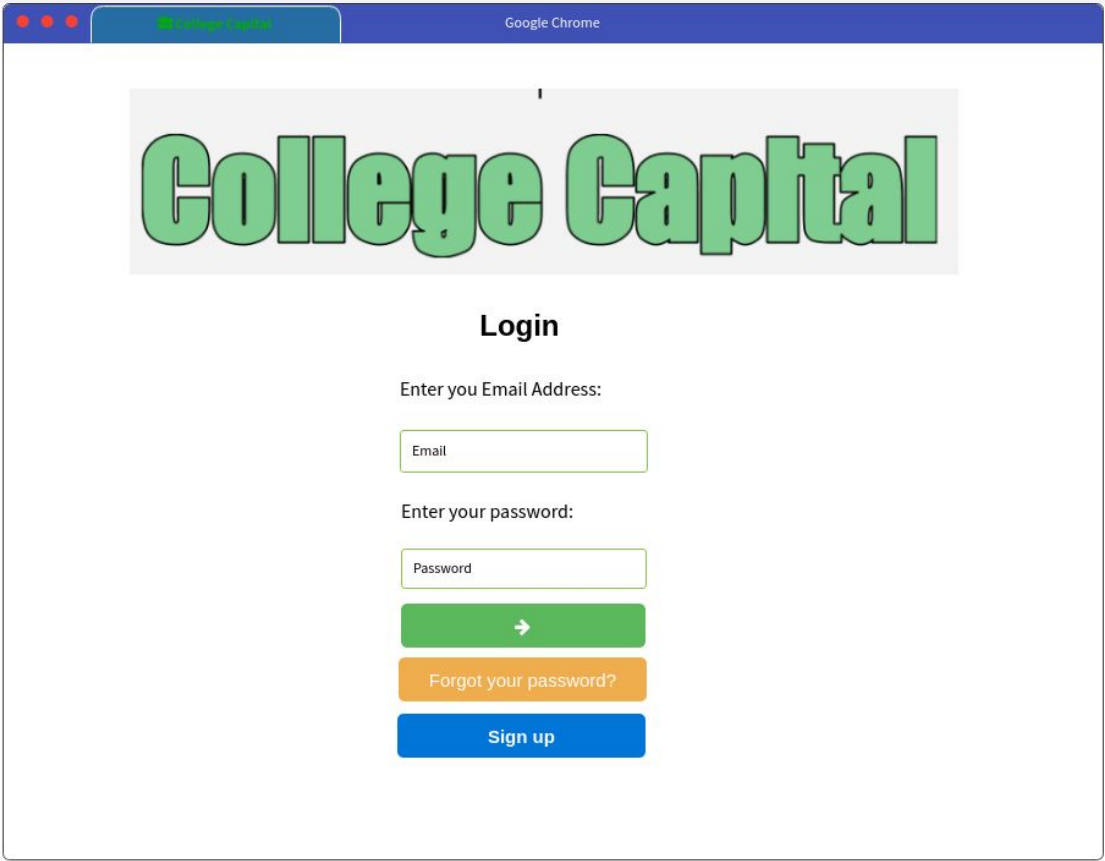
Account Creation Page



The mockup shows a web browser window with a purple header bar. The browser's address bar shows 'College Capital' and the page title is 'Google Chrome'. The main content area features the 'College Capital' logo in a large, green, bubbly font. Below the logo is a 'Signup' section. It contains five form fields: 'Enter you Email Address:' (with a placeholder 'Email'), 'Enter your password:' (with a placeholder 'Password'), 'Enter your Date of Birth:' (with a placeholder 'Date of Birth'), 'Whats your Gender:' (with a dropdown menu showing 'Gender'), and 'Are you a Student:' (with a dropdown menu showing 'Yes'). At the bottom of the form is a green button with a white right-pointing arrow.

On this page, we are demonstrating what a user may see when signing up for a new account. Basic information, such as email and password, will be stored for the user to login in the future. Data of birth and gender will be stored for security purposes, while the student box will tell us what kind of account needs to be created for the user - non students don't need features such as meal plan tracking, so we will omit them if necessary.

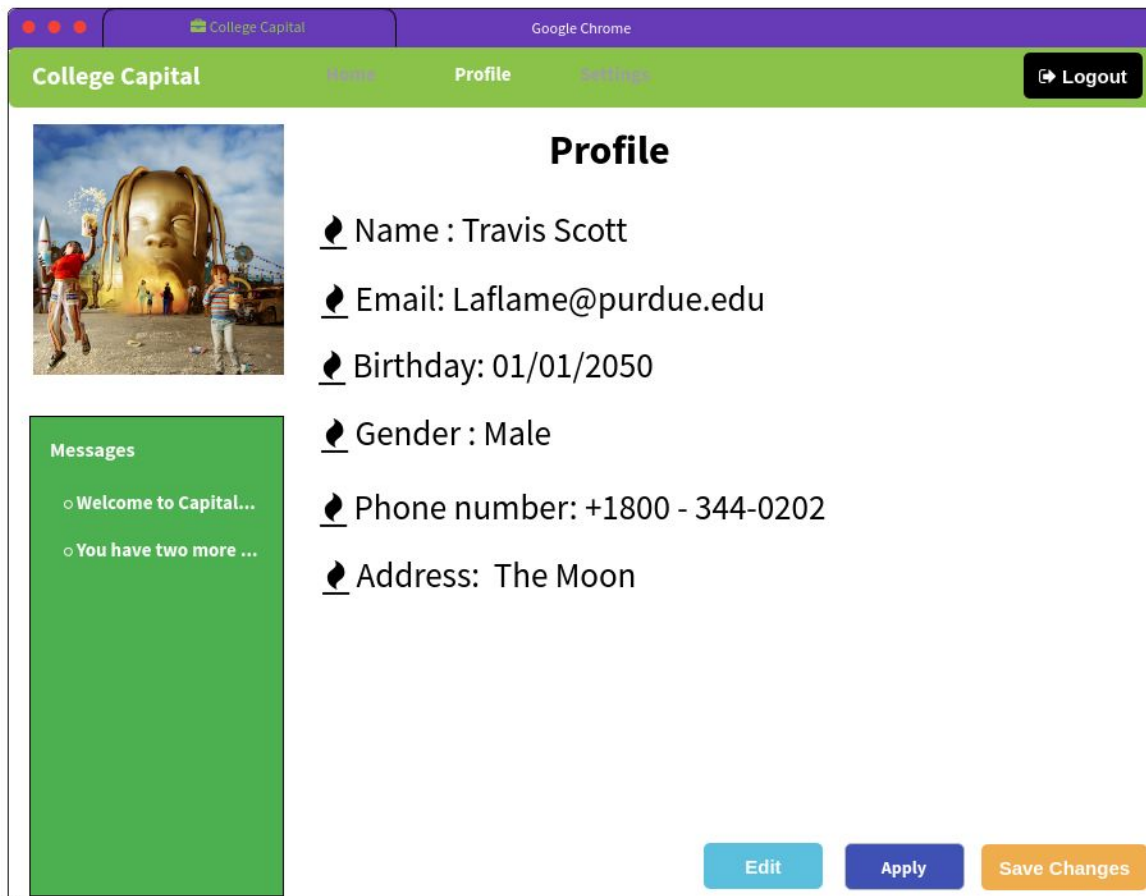
Account Login Screen



The screenshot shows a web browser window with the title 'College Capital' and 'Google Chrome' in the address bar. The main content area features the 'College Capital' logo in a large, green, stylized font. Below the logo is a 'Login' section. It starts with the text 'Enter you Email Address:' followed by an input field labeled 'Email'. Below that is the text 'Enter your password:' followed by an input field labeled 'Password'. There are three buttons: a green button with a right arrow, an orange button labeled 'Forgot your password?', and a blue button labeled 'Sign up'.

This is what users will see after signing up for an account in order to login. They will use the email and password they signed up with to login with the option to recover and reset their password should they forget it. Furthermore, if a user does not have a preexisting account with College Capital, there will be a link to redirect them to the previous screen.

User Profile Page



After creating their profiles and logging in to the website, users can view their personal information on the “profile” tab. On this page, they have the option to update their current information. We’ve added 3 options - the first to edit their data, the second to apply the changes to their personal profile, and the third to save the new information into our database.

User Home Page

Available Funds		Checking	Savings
\$ 64.12		\$32.06	\$32.06

Transaction Number	Location	\$\$	Available
1	Panda Express	-8.19	91.81
2	Villa	-7.10	84.71
3	Sports Clips	-20.59	64.12

This is an example of what the average user may see on their home screen. Readily available are their funds and recent transactions, with the rest of our features available on a sidebar to the left. This allows users to get a quick assessment of their financial situation while providing our extra features in a readily accessible area.

API Routes

The API routes are made available by Firebase's built-in API feature, and they define the standard communication between the frontend and backend. Each route represents objects that are available to view (via GET method), update (via POST and DELETE method), and request (via POST method) from the backend. The objects are identified by their IDs which match with the generated IDs in the database.

Route	Supported Method
/profiles/{id}	GET, POST
/profiles/{id}/account/{id}	GET
/profiles/{id}/account/{id}/transaction	GET, POST
/profiles/{id}/account/{id}/transaction/{id}	GET, DELETE
/profiles/{id}/fund/{id}	GET, DELETE
/profiles/{id}/customtransactioncategory	GET, POST
/profiles/{id}/customtransactioncategory{id}	GET, DELETE
/profiles/{id}/visualizations	POST
/login	GET
/register	GET

State (Activity) Diagram

