Software Requirements Specification

for

Money Gremlin

**Version 1.6 approved**

**Prepared by Eric Jestel, Saurav Pokhrel, Joshua Wilbur, Siddhartha Bajracharya**

**COS 420, Fall 2024**

**October 3rd, 2024**

**Table of Contents**

**Table of Contents ii**

**Revision History ii**

**1. Introduction 1**

1.1 Purpose 1

1.2 Document Conventions 1

1.3 Intended Audience and Reading Suggestions 1

1.4 Product Scope 1

1.5 References 1

**2. Overall Description 2**

2.1 Product Perspective 2

2.2 Product Functions 2

2.3 User Classes and Characteristics 2

2.4 Operating Environment 2

2.5 Design and Implementation Constraints 2

2.6 User Documentation 2

2.7 Assumptions and Dependencies 3

**3. External Interface Requirements 3**

3.1 User Interfaces 3

3.2 Hardware Interfaces 3

3.3 Software Interfaces 3

3.4 Communications Interfaces 3

**4. System Features 4**

4.1 System Feature 1 4

4.2 System Feature 2 (and so on) 4

**5. Other Nonfunctional Requirements 4**

5.1 Performance Requirements 4

5.2 Safety Requirements 5

5.3 Security Requirements 5

5.4 Software Quality Attributes 5

5.5 Business Rules 5

**6. Other Requirements 5**

**Appendix A: Glossary 5**

**Appendix B: Analysis Models 5**

**Appendix C: To Be Determined List 6**

**Revision History**

| **Name** | **Date** | **Reason For Changes** | **Version** |
| --- | --- | --- | --- |
| Eric | 10/2 | Added functional and non-functional requirements | 1.0 |
| Josh | 10/3 | Added 7 functional requirements and 3 nonfunctional requirements. | 1.1 |
| Saurav | 10/3 | Added 7 functional requirements and 3 nonfunctional requirements. | 1.2 |
| Eric | 10/8 | Added link to mockups in 3.1 | 1.3 |
| Siddhartha | 10/8 | Added 7 functional requirements and 3 nonfunctional requirements. | 1.4 |
| Team | 10/10 | Revised document and added title page | 1.5 |
| Josh | 10/19 | Revised requirements based on feedback. Wrote section 1. | 1.6 |
| Eric | 10/21 | Wrote sections 2 and 4 | 1.7 |
| Saurav | 10/21 | Wrote section 3 | 1.8 |
| Siddhartha | 10/21 | Wrote sections 5 and 6 | 1.9 |
| Team | 10/22 | Revised changes | 1.91 |

# Introduction

## Purpose

Money Gremlin is a web application focused around personal finance and budgeting. The purpose of Money Gremlin is to provide users with a way to track finances using a system similar to the envelope method of budgeting. Using physical money envelopes may be inconvenient for some people, especially due to the rise of digital currency. Money Gremlin allows users to keep track of their budget digitally. The specifications and requirements of Money Gremlin are described in this document.

## Document Conventions

The IEEE template for SRS documents serves as the foundation for this document. Information from class instructions have also been used. Each requirement statement has related priorities listed in Section four.

## Intended Audience and Reading Suggestions

The following document is intended for an audience of developers, designers, product managers, testers and users. Section two of this document describes Money Gremlin and its use cases. Section three describes external interfaces that are required to access and interact with the application. Section four describes features of Money Gremlin that users can interact with. Section five describes non-functional requirements in the context of safety, security and quality. Finally, section six describes any requirements not covered in prior sections. It is recommended to read each section in order to understand this application.

## Product Scope

Money Gremlin will be a free-to-use budgeting website intended to help users set budgets and track their spending. The platform will utilize Google's Firebase backend service for secure user authentication and data storage. With a user-friendly interface, Money Gremlin will encourage regular engagement, allowing users to efficiently manage their finances. This application could be monetized by the use of advertisements or premium features.

## References

All of the documentation regarding Money Gremlin is stored in a GitHub repository. Specifically, within the folder *Document/Deliverable\_x*. This is a public repository and is linked below.

Repository: <https://github.com/COS420-Fall24/TeamG>

# Overall Description

## Product Perspective

Budgeting is something desirable to almost all Americans, but almost universally done poorly. The money management market is well established and full of options, but there’s no dominant force in the market (like Google for search engines or Amazon for shipping). Many users of these tools don’t succeed at budgeting, eventually giving up on the tool or not changing behaviors, leaving a market opportunity for a more effective budgeting app. Money Gremlin seeks to fill that role.

Prior to digital money, people used “money envelopes” to budget. After receiving a paycheck and cashing it out, the user would place different portions of their money into physical envelopes, each labeled with a purpose. This would limit spending, cut down on impulse purchases, and require the user to remove money from a category in order to increase spending on something else. The goal of this app is to replicate that functionality to increase budgeting effectiveness.

## Product Functions

The major functions of this app relate to a few sections:

1. Categories - A way to associate transactions with a budget per category
   1. Stores: name, period, and budget
   2. Create category
   3. Delete category
   4. Edit category
2. Transactions - A specific instance of spending
   1. Stores: Amount, date, memo, category, recurring
   2. Log transaction
   3. Delete transaction
   4. Edit transaction
3. Visualizations - Visual representations of the budget will be reported at many levels
   1. Shows overall budget as spending / time
   2. Shows category specific budget as spending / time
   3. Shows breakdown by category (pie chart)
4. Notifications - Notify the user at specific spending breakpoints (80% spending before 80% time)
   1. Set overall notification
   2. Set category notification

## User Classes and Characteristics

There are 3 main user cases for this app: persistent users, infrequent users, and inactive users. The goal of the app is to make budgeting with it easy and accessible, so prior technical experience or education should not be required for maximum performance. However, once users begin engaging with the app, 3 classes will emerge based on frequency of use.

Inactive users are the class that we’re attempting to minimize, as they’ve stopped using the app and will not be able to budget with it. Persistent users are those who successfully budget with the app, using whatever subset of features they need. This class will almost certainly use categories, logging (recurring) transactions, and visualizing their financial history. By having a longer history, they’ll be able to see long term trends and get better predictive results. Finally, infrequent users are somewhere in between. They may use the app, but either fail to log enough transactions or ignore the categories. Trying to use the app without categories makes it no better than any other budgeting app, meaning this user is likely to become an inactive user.

## Operating Environment

The system will run on Chrome, Safari, or Firefox. The backend will use Google Firebaseand login information will be stored in Google Firebase. The website will run on any system running Windows 10 or MasOS 11 or later.

## Design and Implementation Constraints

The primary concern is the secure storage of financial information. While having an “important from credit card” option to bring transactions in would be very helpful to the user, doing so would require that we store credit card information and safely access it. This necessitates very, very high security measures as credit card information must be 100% secure for user trust.

Additionally, we must securely store people’s transactions. While this is less sensitive than credit card information, since it will not lead to financial damage, it is still private information that must be secured. Encryption of some form will be needed for this database.

As an app designed to store data, there is a need for a database. At small levels, this can be hosted on Github or other free web alternatives, but expansion of the app would require the purchase of server space. As such, some form of income would need to be extracted from users, whether it be donations, ad revenue, a premium mode, or a purchase price.

## User Documentation

There will be an in-site tutorial when an account is first created. Additionally, there will be a “Replay tutorial” button on the site allowing users to rewatch it. This tutorial will demonstrate the main functionalities (create/edit/view category/transaction), but an advanced tutorial that demonstrates every single button may be helpful, though less important. The app will also focus on ease of use and intuitive functionality to minimize the need for a tutorial.

There will also be documentation on Github.

## Assumptions and Dependencies

The product assumes that the user can access the site immediately after making transactions or will change the date for transactions entered later. The project will use Google Firebase. The server needs to be stable and accessible via the internet. The client must also remain on the internet.

# External Interface Requirements

## User Interfaces

UI mockup: <https://docs.google.com/presentation/d/1yvxT47wots3ccU64k7fqQTBp045WM9lHmpsLHmnby3c/edit#slide=id.p>

Backend/functional mockup:

<https://docs.google.com/spreadsheets/d/1i_vgkgTgXrafaWyAOq1pc-KmzBoONCCTVwmPLCYwS9c/edit?gid=0#gid=0>

*Money Gremlin will feature an intuitive and minimalistic web interface designed to facilitate easy budget tracking and management. The main user interface components include:*

Homepage: The homepage prominently displays the app’s name, "Money Gremlin," along with a graphic or video section that presents the app’s features with eye-catching visuals and text. The page includes a call-to-action button for users to Log in / Sign up.

Login and Sign-Up: The login interface offers multiple options, including signing in with Google or creating an account with an email and password. The interface layout is straightforward, with clearly marked buttons for Google authentication and manual input fields.

Dashboard: The dashboard provides a central area called the Budget Dashboard that displays an overview of the user's current budget categories and spending status. Key actions, such as Log Transaction, Update Category, and Create Category, are accessible as separate buttons on the side. Hovering over these buttons displays brief descriptions to guide users.

Transaction and Category Management: When logging a transaction, users are presented with a simple form that includes fields for selecting a category, entering an amount, and adding an optional memo. Similarly, the Update Category interface allows users to choose a category, view associated information, and update settings such as budget limits and notifications.

Account Settings: The account options page lets users update their profile details, such as changing the username, email, password, and notification preferences. A dedicated section allows users to permanently delete their accounts after a confirmation.

Each of these interfaces is designed with a focus on accessibility, with clear headings, labeled fields, and visually distinct buttons to ensure ease of navigation for all user classes.

## Hardware Interfaces

Money Gremlin is a simple web based application designed to operate on any device with basic web-browsing capabilities. There are no direct hardware interfaces beyond basic input devices like a mouse, and keyboard. No special hardware is required to use Money Gremlin.

## Software Interfaces

Money Gremlin connects with several key software components to deliver a cohesive and user-friendly budgeting application. These interfaces include:

Firebase (Google): The application integrates with Firebase for managing authentication, data storage, and hosting services. Firebase handles user login and sign-up via email/password and Google OAuth, ensuring secure user verification.

Firestore Database: A NoSQL database that stores user profiles, budgets, categories, and transactions.

Cloud Functions: Executes backend tasks like automated notifications or data processing.

React.js (Frontend Library): The frontend is built using React.js, providing an interactive and modular user experience. React.js manages user inputs, navigation, and dynamic data visualization.

External APIs: Money Gremlin may utilize APIs for external financial data, such as exchange rates, in future versions.

## Communications Interfaces

The application communicates over standard HTTP/HTTPS protocols to interact with Firebase services and transmit data securely. Key communication interfaces include:

User Authentication: Communicates login and registration requests using Firebase Authentication's secure protocols.

Data Storage and Retrieval: Exchanges requests with Firestore to store and retrieve user-generated data such as transactions and budget categories.

Notifications: Pushes notifications to users based on their spending thresholds via Firebase services.

All communication between the front end and the Firebase backend is encrypted using HTTPS to ensure data integrity and security.

# System Features

## Create and Edit Profile

4.1.1 Description and Priority

This feature will let users create a private account, with an associated email and password. The user will be able to log into the account, log out of the account, and delete all associated data. The user will also be able to change their email or password. All stored financial data will be associated with the account.

This feature is **high priority**

4.1.2 Stimulus/Response Sequences

* User clicks on “Create profile”
  + System goes to “Create profile page”
    - Display: “Enter email”, “Enter password”, “Confirm password”, “Create account”, and “Sign up with Google”
* A) User enters information and clicks on “Create account”
  + If the account doesn’t exist, account is created
  + If the account does exist, display “Account already created. Log in instead”
* B) User clicks on “Sign up with Google”
  + Bring up Google account creation popup
* After A or B:
  + User is logged in
  + User is directed to tutorial experience
* User clicks on “Log in”
  + System goes to “Login” page
    - Display: “Enter email”, “Enter password”, “Sign in”, “Forgot password”, and “Sign in with Google”
* A) User enters information and clicks on “Sign in”
  + If information is correct:
    - User is logged in
    - User is directed to main page
  + If information is incorrect:
    - Display “Wrong email or password”
    - Clear password field
* B) User clicks on “Sign in with Google”
  + Bring up Google signin popup
  + User is logged in
  + User is directed to main page
* C) User clicks on “Forgot password”
  + Go to “Forgot password” page
  + User enters email
  + System sets users password to randomized password
  + System sends new password to email
* User clicks on “Account options”
  + System goes to “Account options” page
    - Display: “Change email”, “Change password”, “Set profile pic”, and “Delete account”
* A) User clicks on “Change email”
  + System displays “Enter password” and “Enter new email”
  + User enters password and new email
  + System changes email associated with account
* B) User clicks on change password
  + System displays “Enter old password”, “Enter new password”, and “Confirm new password”
  + User enters passwords
  + System changes password associated with account
* C) User clicks on “Set profile pic”
  + System displays a range of profile picture options
  + User selects one
  + System changes profile pic associated with account
* D) User clicks “Delete account”
  + System displays “Are you sure?” with Yes and No buttons
  + User clicks Yes
  + System displays “Enter password”
  + User enters password
  + System deletes account and all associated data

4.1.3 Functional Requirements

REQ-4: The system shall allow the user to make a new account with an email and

password.

REQ-5: The system shall let the user update their email and/or password.

REQ-8: The system shall allow the user to log in using their account credentials.

REQ-22: The system shall allow the user to reset their password using the email associated with the account.

REQ-23: The system shall allow the user to log out of the system.

REQ-24: The system shall allow the user to delete their account.

REQ-27: The system shall let the user can select a profile picture

REQ-29: The system shall allow the user to change their profile picture.

REQ-31: The system shall allow the user to log in with Google

REQ-32: The system shall explain its functions to new users

## Create and Edit Categories

4.2.1 Description and Priority

This feature will let the user create a “category”, which is a specified portion of a budget. This will have a name, spending limit, and period. For example, you might have “Food” with $500 for 1 month. Users can also set notifications for different percentages spent, based on time elapsed. For example, if more than $250 (50% of the budget) is spent before 15 days (50% of the time), an alert would go off. The user can edit the name, spending limit, period, and add, remove, or modify notifications. There is always a notification at 100% of the budget spent.

*This feature is* ***high priority***

4.2.2 Stimulus/Response Sequences

* User clicks on “Create category”
  + System goes to “New category” page
    - Display: “Category name”, “Category limit”, “Category period”, “Add notification”, and “Create category”
* A) User enters information and clicks on “Create category”
  + If any information is not yet filled, display “Please enter all information” and highlight/bold all unfilled boxes
  + If the category doesn’t exist, category is created
  + If another category with the same name does exist, display “Another category already exists with that name”
* B) User clicks on “Add notification”
  + Add a new notification to the list
  + Display “Notify me at {fillable int}% spent before {fillable int}% time”
  + Display X icon next to new notification to remove it
* User clicks on a category
  + System displays category information page
  + User clicks on “Edit category”
    - System goes to prefilled “edit category” page (identical to “Create category”, but with all tabs filled and a “Delete category” button)
  + A) User edits information as desired
    - User clicks on “Save category”
    - System returns to category information page
  + B) User clicks on “Delete category”
    - System displays “Are you sure?” with a yes/no button
    - User clicks on yes
    - System removes category and returns to main page

4.2.3 Functional Requirements

REQ-1: The system shall allow the user to create spending categories.

REQ-9: The system shall allow the user to set custom notification settings for when they approach the spending limit.

REQ-13: The system shall notify the user when the user has exceeded their budget.

REQ-18: The system shall report the predicted rate and the predicted date when spending will exceed budget to the user.

REQ-19: The system shall allow the user to set a spending limit for each category.

REQ-20: The system shall allow the user to set spending warnings at different dollar values or percentages of the spending limit.

## Log and Edit Transactions

4.3.1 Description and Priority

This feature will let the user create a transaction. The user will enter a dollar amount, category, date (defaulting to the current day), and an optional memo, then the transaction will be added to the correct category. For example, $75 for “Food” on 10/19/2024, with a memo of “Grocery shopping”. The user will also be able to edit these transactions.

This feature is **high priority**

4.3.2 Stimulus/Response Sequences

* User clicks on “Create transaction”
  + System goes to “New transaction” page
    - Display: “Transaction name”, “Transaction date”, “Transaction category”, “Memo (optional)”, “Recurring transaction?”, and “Create transaction”
* A) User enters information and clicks on “Create transaction”
  + If any mandatory information is not yet filled, display “Please enter all information” and highlight/bold all unfilled boxes
  + If all information is there, create the transaction
* B) User clicks on “Recurring transaction?”
  + Display “Repeats every {fillable int} {days/weeks/months} starting on {fillable date}”
* User views the transaction log
  + System displays transaction log
  + A) User clicks on a transaction
    - System goes to prefilled “edit transaction” page (identical to “Create transaction”, but with all tabs filled and a “Delete transaction” button)
  + User edits information as desired
  + User clicks on “Save transaction”
    - System returns to transaction log

4.3.3 Functional Requirements

REQ-6: The system shall allow the user to input transactions.

REQ-10: The system shall allow the user to categorize each transaction. (ie. food, gas)

REQ-12: The system shall allow the user to undo a transaction for accidental transaction entries.

REQ-14: The system shall allow the user to edit transaction amounts.

REQ-16: The system shall track the date, amount, and category of each transaction.

REQ-15: The system shall allow the user to assign a transaction to only 1 spending category

REQ-26: The system shall allow the user to schedule recurring transactions, such as monthly bills, subscriptions, or income, with customizable frequencies (e.g., monthly, weekly)

REQ-21: The system shall allow the user to attach a memo to a transaction

## Visualize Spending

4.4.1 Description and Priority

This feature will let the user view their spending. They will be able to set a monthly spending limit, see spending over time for different intervals in a category or overall, predict future spending, and identify the highest spending categories.

This feature is **high priority**

4.4.2 Stimulus/Response Sequences

* User on main page
  + System displays a graph of spending against time compared to the sum of all categories' spending limits. This is a stacked graph with each category being a different color and a key. Additionally, a line of best fit will be displayed for the future.
  + User double clicks on y axis, or x axis
    - User clicked on Y axis
      * Display “Enter monthly spending limit” and a list of categories with check marks for each category
      * User clicks on a check mark
        + System hides that given category
      * User enters a new monthly spending limit
        + System sets Y axis max to that spending limit
    - User clicked on X axis
      * Display “Set period”
      * User enters new period in days
      * System changes the x axis length to cover that many prior days
  + User double clicks on a category
    - System displays a graph with only that category included

4.4.3 Functional Requirements

REQ-2: The system shall let the user define a monthly spending limit.

REQ-3: The system shall provide a visualization of the user’s spending over the last 30 days.

REQ-17: The system shall predict a rate of spending based on the transaction history.

REQ-28: The system shall provide a financial summary that includes their total income, total expense, and their expenditure breakdown.

REQ-11: The system shall provide budgeting recommendations based on the user’s spending trends.

## View Transaction Log

4.5.1 Description and Priority

This feature will let the user view a list of each transaction logged. They will also be able to filter by memo, category, amount, recurring, and date. The user can also export the log of all transactions.

This feature is **low priority**

4.5.2 Stimulus/Response Sequences

* User clicks on “Transaction log”
  + System goes to Transaction log
    - Display header: “Transaction category” (dropdown), “Transaction date”, “Transaction quantity”, “Transaction Memo” (fillable text), “Transaction recurring?”, and up/down arrows next to category, date, quantity, and recurring. Down arrow next to date initially selected.
    - Display each transaction, initially sorted by date from newest to oldest
    - User clicks on up/down arrow next to a category
      * Sort all transactions by that category in the correct direction (alphabetical or numerical)
    - User enters text in “memo” box
      * System hides all transactions that do not contain the given string in memo
    - User clicks on category dropdown
      * System displays list of categories with (filled) checks next to each
      * User clicks on a check
      * System hides all transactions with that category
* User clicks on “Export data”
  + System outputs a .csv file containing all currently displayed options

4.5.3 Functional Requirements

REQ-7: The system shall allow the user to export and download their transaction log.

REQ-25: The system shall allow the user to view transactions by date, category, or amount.

REQ-30: The system shall provide a search bar to find specific transactions

# Other Nonfunctional Requirements

## Performance Requirements

PERF-1: The system shall respond to user input within 1 second, 99% of the time.

PERF-2: The system shall update the transaction log, within 2 seconds of the user inputting a new transaction, 95% of the time.

PERF-3: The system shall decrease the available monthly spending limit after the user inputs a new transaction within 2 seconds, 95% of the time.

PERF-4: The system shall remain active 99% of the time between 8am and midnight.

PERF-5: The system shall remain active 95% of the time between midnight and 8am.

PEFR-6: The system shall display visual alerts or error messages for failed operations within 1 second of detecting the issue, 99% of the time.

PEFR-7: The system shall load the user’s profile page within 2 seconds, 95% of the time.

## Safety Requirements

SAFE-1: The system shall encrypt all sensitive user data.

SAFE-2: The system shall incorporate a “forgot password” feature associated with the users email address

SAFE-3: The system shall require an additional authentication to access sensitive financial information

## Security Requirements

SEC-1: The system shall operate within GDPR and CCPA regulations to safeguard the user’s data.

SEC-2: The system shall store only the hash of the password.

SEC-3: The system shall securely encrypt the hash table and salt it.

SEC-4: The system shall comply with the OWASP Top 10 Security Standards to mitigate common security vulnerabilities and ensure secure handling of user data.

SEC-5: The system shall comply with PCI DSS for processing, storing, and transmitting credit card information to ensure data protection and regulatory compliance.

## Software Quality Attributes

SQA-1: The system shall be designed with a modular code structure so that individual components can be updated and replaced without impacting the whole system.

SQA-2: The system shall have an intuitive and easy-to-understand interface, requiring no more than 30 minutes for new users to get used to the interface.

## Business Rules

BR-1: Administrative users have access to system-level settings, including user management, and system monitoring.

BR-2: Administrative users can view anonymized data for reporting and system analysis but cannot access individual user details without consent from the user.

BR-3: Users must accept the Terms of Service before making an account.

# Other Requirements

**Appendix A: Glossary**

**CCPA (California Consumer Privacy Act)**: A California law that enhances privacy rights and consumer protection for residents of California, regulating how businesses collect and handle personal data.

**GDPR (General Data Protection Regulation)**: A European Union regulation that sets guidelines for the collection and processing of personal data from individuals who live in the European Economic Area (EEA).

**PCI DSS (Payment Card Industry Data Security Standard)**: A set of security standards designed to ensure that all companies that accept, process, store, or transmit credit card information maintain a secure environment.

**SRS (Software Requirements Specification)**: A document that describes the system's functionality, performance, and operational and security requirements, as well as constraints and non-functional requirements.