CSCI 4805 - Project Proposal
Froogal: The Premier Finance Web Application

By: Johnathan Dickson, Chris Adamson, & Dylan Commean

Table of Contents

Page #

Project Name

Being a finance-centric web application, primarily focused on managing personal finances, the word frugal came to mind. Playing off this keyword, we decided on the name of **Froogal** for our application.

Team Member Names

See Figure 1 in the appendix for detailed description.

Abstract

Financial literacy is the cornerstone of wealth building. Budgeting is a powerful tool in one's financial journey; however, it alone is not sufficient. An analysis on one's spending habits is a key metric that reflects who the consumer is at their core. The process of analyzing spending can be a daunting task; where does one even start? Froogal aims to streamline this process by providing rich financial reports about one's spending, over various periods (day, weeks, months, and years), so our users can have a deeper understanding of their spending habits. This gives our users the upperhand in controlling their personal finances, thus combatting overspending.

Description

Froogal is the all-in-one solution to reviving your personal finances! Budgeting, albeit a powerful tool, is not sufficient by itself for controlling finances. The reason being is that it merely suggests approximated spending targets but does not address the root problem of dying finances - poor spending habits! It does not matter how much money one makes, if he spends more than what is coming in. Froogal aims to highlight a user's spending habits by providing rich, data-driven models which pinpoint a user's spending based on certain categories. Additionally, Froogal attempts to consolidate a user's expenses such as bills and subscriptions in one place. Keeping all of this data in a centralized dashboard further gives users' a high-level overview of upcoming expenses without switching between multiple platforms, thus allowing them to monitor more than just spending habits.

Feature List

Froogal's mission is to simplify the process of spending analysis and give deeper understanding of a user's personal spending habits. To support this goal, the application will include:

- The ability to create, delete, update, and read user profiles
- Data-driven charts crafted from a user's profile
- The ability to record and categorize each transaction
- Creation of custom categories to enhance spending analysis
- Dashboard for users to view various different models and graphs based on their spending habits
- Various push and email notifications that will notify users based on certain criteria such as reaching the max budget allocation for X category or when bills/subscriptions are reaching their due dates.
- Financial reports for different periods of time (such as days, weeks, months etc.), all pulled from data. This allows users to visualize their spending habits.

Technology

Being a reactive based web-application which needs to store and query user data, our tech stack includes various technologies. For frontend functionality, React will cover the need to create a reactive based Single Page Application (SPA). For styling, we will either use Tailwind Cascading Style Sheets (CSS) or Chakra User Interface (UI) to style our UI components easier - adhering to either a predefined or custom design system. For backend and database needs, we are considering the Firebase platform.

Firebase is a versatile platform that has several key products, which are mostly free, that we can use. Firestore is a No Structured Query Language (NoSQL)-like cloud database which will store our user records. AppCheck, a Firebase service, will add an extra security layer for user data while meeting major compliance and security standards. Firebase Auth will securely handle user authentication, enabling various methods of login (e.g. Google login, Facebook login, email/password, and more). Firebase Hosting will handle our need of serving the initial SPA as well as any pages after that (e.g. 404, user login). Moreover, the Firebase ecosystem contains many crucial packages that we can easily integrate to satisfy the application's needs of querying data, handling data visualization (e.g. chart generation based on user data), and sending push notifications and emails to users.

An additional technology we may consider is TypeScript. This will provide us with a typing system which can be used to reduce compile time errors and minimize the amount of faults and errors pushed to our pre-production branch. From there, we will use testing

frameworks to test UI components and the Firebase Console to test and verify our database is being manipulated correctly. After testing we will push to the production branch.

Server Information

Firestore, a product from the Firebase platform, will support the application. As of right now, this will be the only database we plan to integrate with the application. The following are some benefits that Firestore will provide: real-team syncing and storing of data between users, collaboration across various devices (mobile, web, etc.), user-based security integrated with Firebase Auth rules, and usage of security rules. Because Firestore ships with a mobile and web Software Development Kit (SDK), this allows our database to be serverless. Changes to the database will happen through cloud functions which will manipulate data as needed. Moreover, Firestore can be configured to work in offline mode. When this happens, the SDK will automatically switch to local cache when devices are offline. When users reconnect to the web, local cache will be synced to the cloud database. This is beneficial if we plan to extend the app to mobile devices as well.

Data Sources

Firestore will be our only source of data as of now. If the need arises, we can consider other databases that are aligned with the project's requirements.

Team Members' Backgrounds

Dylan Commean:

He has enterprise-level experience in web development, server development and Structured Query Language (SQL). He is experienced in crafting modular UIs, Create, Read, Update, Delete (CRUD) operations, and database to server connections. He has knowledge many languages: React.js, ReactNative, Tailwind CSS, HyperText Markup Language (HTML), CSS, C#, C++, Java, JavaScript, nodeJS, ExpressJS, mySQL, SQL. They are comfortable with any of the responsibilities whether that be front-end, back-end, Application Programming Interface (API), or database.

Johnathan Dickson:

Highly interested in web-based development, he has experience crafting reactive frontends using React.js and Gatsby. For styling, he has experience using Tailwind CSS, Daisy UI, Chakra UI, Syntactically Awesome Style Sheets (SASS), and plain CSS. For backend

languages, he has experience in Java, C#, C++, JavaScript & experience in nodeJS using the ExpressJS framework. He has experience in NoSQL-like databases, specifically Firebase.

Chris Adamson:

His primary experience is in Java and C#. He has dabbled into other languages such as HTML, CSS, PHP: HyperText Preprocessor (PHP), and SQL. He is driven and willing to learn the needed languages, packages, and frameworks needed for the project. He is eager to further expand his technical knowledge through hands-on experience.

Dependencies, Limitations, and Risks

Dependencies

See Figure 2 in the appendix for detailed description.

Limitations

Practical limitations that could arise in the short-term are:

- Inexperience of developers regarding certain languages and frameworks
- Issues setting up Firebase's Firestore
- Lack of a typing system during development
- Incurring costs using Firebase products
- Improper state management on the frontend (can be medicated using packages)
- Currently needing to find a data visualization package

Risks

Practical limitations that could apply are:

- Lack of knowledge in backend development if Firebase does not work
- Learning curve of using state management packages & 3rd party data visualization packages
- User Information Storage
- Potential Security Vulnerabilities

Timeline

See Figure 3 in the appendix for detailed description.

Appendix

Figure 1: Team Member Names

Name	Role
Dylan Commean	Developer, Architect
Chris Adamson	Developer, Architect
Johnathan Dickson	Developer, Architect

Figure 2: Froogal Tech-Stack

Place In System	Dependency Name	Dependency Type	Why
Frontend	React	JS Library	Implement component based, reactive UI's to configure pages based on user's state.
Frontend	Chakra UI	CSS Framework	Styling React components with a consistent predefined or custom theme
Frontend	Tailwind CSS	CSS Library	CSS utility first library for more expressive styling.
Frontend	React-Spring	JS Library	Incorporating Spring-based animations for components
Frontend	Framer Motion	JS Library	Incorporating animations, is a dependency of Chakra UI
Frontend	Redux	React Library	Needed to reduce the complexity of handling app state for users.
Backend	Firebase Platform	Cloud Platform	Utilizing the Firebase platform to handle the harder parts of backend development (e.g. user auth, security practices, etc.)

Data Source	Firestore	Database	NoSQL real-time cloud-based database to handle user's information using JavaScript Object Notation (JSON).

Figure 3: Tentative Schedule*

Week 1: August 21st - 27th	Application proposal, product discovery, determining softwares and languages, assessing team members strengths and abilities, information gathering.
Week 2: August 28th - September 3rd	Team setup with the tools for development. GitHub/Trello setup for sprints, research and resolve any issues with hardware or tools. Information gathering.
Week 3: September 4th - 10th	*No Class - Labor Day (5th)* Setup application base in GitHub. Creation of master branch and development branches. Discovering/thinking about the Front-end components. Designing database architecture and back-end structure. Information gathering.
Week 4: September 11th - 17th	Production officially starts. Establishing connections from the back-end to database. Getting a feel for the languages we are using, asking questions, and solving any issues that arise. Information gathering.
Week 5: September 18th - 24th	Production of components, server-side components/connections, and database materials. Information gathering.
Week 6: September 25th - October 1st	Production of components, server-side components/connections, and database materials. Testing and debugging of initial work alongside programming.
Week 7: October 2nd - 8th	Production of components, server-side components/connections, and database materials. Testing and debugging continue.
Week 8: October 9th - 15th	*No Class - Fall Break (10th - 11th)* Production, testing, and debugging continues. Assessments are done to determine how the team is doing. Teammates assist in other areas that are falling behind if necessary.

Week 9: October 16th - 22nd	Production, testing, and debugging continues.
Week 10: October 23rd - 29th	Production of the base site is nearing completion. Areas needing help are accessed and completed.
Week 11: October 30th - November 5th	Base site and base components are ready for testing. Solving any problems that arise with the initial designs and remedying those issues.
Week 12: November 6th - 12th	*No Class - Veterans Day (11th)* Optional components are looked over for their viability and ordered by deliverability. As a team, decide which components should and could be implemented in a timely manner.
Week 13: November 13th - 19th	Production of optional components. Bug fixes and polish work.
Week 14: November 20th - 26th	*No Class - Thanksgiving Holiday (23rd - 27th)* Cut off week for the production of optional components. Testing, bug fixes, and polish work.
Week 15: November 27th - December 3rd	Finishing touches and polish work. Gear up for presentations the following week.

*Schedule subject to change