Version 1

CECS 443 Software Project Management and Testing, Spring 2023

Budget Buddy Implementation

By
Ibrahim Al Balushi
Beatriz Sonsoles Encinas Muñoz
Maxi Guillermo
John Mintsa
Brandon Wiitanen
Sarah Zacky

30 April 2023

Index

Preface

The decision to implement the Budget Management Application project (“Budget Buddy”) has been based on two main principles: the usefulness of the product, and the feasibility of the project in real life. It is designed to help customers manage their finances more easily and efficiently. It is also worth mentioning that the experience of the team members with Web Development and APIs, among others, will reduce the amount of time needed to implement the project.

Moreover, the purpose of this document is to provide a better understanding of any changes made to the architecture, design, and/or requirements, that have been considered appropriate by the developers. In addition to that, the reasoning behind these changes will be included as well. Thus, the document also serves as a way of informing the client of any changes that might affect the final product.

This document will also explain any extra functionalities added by the developers to ensure product differentiation within the market, as well as also allow the stakeholders to know what to expect from the finished product.

Overall, the Budget Management Application project is designed to provide a valuable solution for its users, while offering several benefits to the client. This document will provide a clear understanding of the project's implementation and changes made, helping all stakeholders set clear expectations for its development.

Introduction

BudgetBuddy is a budget management application that represents a useful tool for users who are interested in effectively managing their money. It seeks to ease the responsibility of monitoring a user's expenses, as it provides a user-friendly, easy-to-use user interface and requires few inputs from the users.

It allows users to create budgets by providing a name, and amount, and selecting the category it belongs to, as well as add their expenses by specifying the name, amount, and corresponding budget name. Optionally, users are also offered the option of setting a reset period for both budgets and expenses. The category a budget belongs to can be selected from a dropdown menu that includes the categories: Automobile/Transportation, Bills/Utilities, Business, Education, Entertainment, Food, Gifts/Donations, Health/Fitness, Home, Income, Kids, Shopping, Travel, and Other.

The software then updates the corresponding budget after an expense has been added, resets a budget or expense if a reset period has been specified by the user, and notifies the user when more than 80% of the budgeted amount has been expended, together with the time left until the budget is reset.

Some additional functionalities of this application take into account further necessities of a user. Therefore, a user of Budget Buddy will also be able to upload receipts so that they do not need to enter all their expenditures manually. Moreover, both budgets and expenses can be edited and deleted, in case a user would like to make any modifications they deem appropriate.

Budget Buddy also offers users the possibility of switching from a dark theme to a light theme and vice-versa, as well as setting their own profile picture for a more customized experience. The home page will also display all the expenses created by the user together with the total budgeted amount and the total amount left after the expenses. This way, the user can have an overview as soon as they open the app. Finally, the user can access their budgets by going to the "Budgets" page, where these budgets are organized by the categories chosen by the user for easier navigation through the application.

Architectural Change

During the implementation of Budget Buddy, we encountered some problems related to the architecture of the application. The modifications we decided to apply, as well as the reasoning behind making these changes, are mentioned and explained in the paragraphs below.

We have changed the **patterns and styles** that are chosen for the architectural design, we've chosen a Layered Architecture Pattern and it is considered a Transaction Processing Application. Although the Layered Architecture Pattern is said to have a higher implementation difficulty and lower performance (due to requests having to be interpreted by multiple layers), it seemed to be the pattern that fitted our application the most. Also since the users will be making transactions which refers to a sequence of operations that a user must perform in order to satisfy a goal, the budget application app is a good fit for a Transaction processing application.

Although the Layered Architecture Pattern includes a layer dedicated to the **database**, our database was implemented by using Chrome localStorage instead of creating a relational database, which might seem more conventional. Therefore, that layer is composed of the users database, the budgets database, and the expenses database, which are all encompassed in localStorage. The reason why we decided on localStorage instead of a database is that localStorage is a much simpler approach that can easily be run locally. Furthermore, not only is it faster, cutting down all response time when retrieving data, and more reliable for users, but it is also more secure since we won't face the issue of centralized databases being compromised.

In addition, the **components** that were **removed from the layered architecture** are the following:

- API manager
- API connection
- Budget browser

Both the API manager and API connection were meant to be implemented to allow users to link their Venmo account. Given that integration with Venmo is also part of the requirements, a more detailed explanation will be provided in the Requirement Change section in order to avoid repetition.

Regarding the **Budget browser** component, it was an optional extra feature that we thought about implementing so that users could search their created budgets by budget name. However, we thought it was not a critical functionality of the system, as budgets are organized by categories and that is already a way of facilitating the user's search for a specific budget.

Detailed Design Change

This section will cover the detailed design changes made to Budget Buddy during its implementation, as well as the rationale behind the decision to make these changes.

Users were not required to give a **name** to their **expenses** at first, but it was later added as a required field because identifying each individual expense was necessary for the reset expense functionality to work.

Although we created a **bell button for notifications** to be shown there and later accessed by the user, we decided to show a pop-up message on the web page instead. We felt that this would be more noticeable to the user and would be beneficial because they couldn't miss the notification.

It was first said that the **receipt scanning** feature could be able to distribute each of the items in the corresponding budget category, but this functionality was deemed too complex so we decided to simply create a button to upload the receipt and obtain the total amount spent from it.

In the beginning, we also wanted to allow users to set **multiple reset periods for a budget**, as the user might want to track the money spent in a particular subcategory of a budget. However, no subcategories of a budget have been implemented. Therefore, instead of a budget having multiple categories, a category can have multiple budgets. This change was applied because we believed this design to be more coherent and more intuitive for the users.

Requirement Change

The Requirement Change section is meant to indicate and explain any requirements that have been adapted from their original state. Some requirements were difficult to carry out on time for the delivery of the product. Therefore, it has been necessary to slightly modify them.

The requirements first asked to create a **mobile application** for users who wanted to keep track of their expenses. After some debating, we came to the conclusion that most of us had more experience with web applications rather than mobile applications. Consequently, we made the decision to create a web application. Doing this means users can only access the application on their personal computer, so a future project to work on could be adapting the application to mobile devices.

Although one of the requirements was to **integrate** Budget Buddy **with Venmo**, the Venmo API could not be used due to it no longer being offered. Therefore, we chose to go with PayPal instead. This was not a huge problem, since Venmo and PayPal would have a similar output on our application.

Nevertheless, **PayPal** could not be fully **integrated** with our system due to our decision to not host our application on a custom domain. Because the “Log in to PayPal” button uses the OAuth 2.0 authorization protocol, which uses access tokens to authenticate and authorize the user, that means we would need a callback URL to store the client’s token. When designing our system, we did not realize that a callback URL was needed to store the token. Our application currently takes the user to the point of logging into PayPal but fails after entering credentials because the token isn’t being stored anywhere for our application to access it. We did not decide to change our design to account for this, but if we ever decide to implement it in the future, we would then need to create a domain, host our application on that domain, configure the callback URL, and update our code to retrieve the user’s token. We would also need to update the Expenses page to display the PayPal transactions.

Another requirement that we needed to modify was the **receipt scanning** feature. Instead, we decided to ask users to upload their receipts to the application.

The ability to read text live from a camera has yet to be fully realized for Javascript. The closest we could come to that was using an Optical Character Recognition (OCR) library, called Tesseract, to read text from an uploaded image to the best of its ability. This changes the design because it makes the feature operate completely differently. Rather than needing access to the live camera of the device, we only need access to the given images, and in some cases, it could prove to be more inconvenient for users than the method we originally wanted to implement. The change has been implemented, however, and it is currently how the feature operates in the application.

The last requirement that was changed is **user notification**. Originally, the intention was to notify users of the amount left until the budget was reset or fully expended as a percentage. Nevertheless, it was changed to notify users when more than 80% of the budget is expended to avoid overwhelming users with too many notifications.