NGESA IAN

PROJECT: CASH WRAP

System Requirements Specification (SRS)

Chapter 1 Contents

1. Introduction

1.1 Purpose

The purpose of this document is to outline the requirements for the development of a **Cash Wrap**, a Budget Planner web application. The application will allow users to manage their budget by creating and tracking various activities, each with an allocated budget. The application will provide a visual representation of expenses through a flowchart and progress bars.

1.2 Scope

The Budget Planner web application will allow users to:

- View a flowchart displaying the percentage of their total budget spent across different activities.
- Add, edit, and delete activities such as travel, drinks, rent, etc.
- Allocate a budget to each activity.
- Track the spending for each activity with a progress bar that updates as expenses are logged.
- Perform CRUD (Create, Read, Update, Delete) operations on activities.
- Use a MongoDB database via Mongoose to store data.

1.3 Definitions, Acronyms, and Abbreviations

- CRUD: Create, Read, Update, Delete
- Mongoose: A MongoDB object modeling tool designed to work in an asynchronous environment.

- React: A JavaScript library for building user interfaces.

2. Overall Description

2.1 Product Perspective

Cash Wrap will be a standalone web application developed using React for the front-end and Node.js with Mongoose for the back-end. It will integrate with a MongoDB database to store and retrieve user data.

2.2 Product Features

- ✓ Flowchart Display: A visual representation of the total budget and the percentage spent on each activity.
- ✓ Activity Management: Users can add new activities, set a budget for each, and track spending.
- ✓ Progress Bars: Each activity will have a progress bar showing the amount of the allocated budget that has been used.
- ✓ CRUD Operations: Users will have the ability to create, read, update, and delete activities.
- ✓ Responsive Design: The application will be accessible on various devices, including desktops, tablets, and mobile phones.

2.3 User Classes and Characteristics

General Users : Users who want to manage and track their budget. They may not have technical expertise but should be familiar with basic web applications.

2.4 Operating Environment

- ✓ The application will run in any modern web browser (Chrome, Firefox, Safari, Edge).
- ✓ The back-end will be hosted on a server capable of running Node.js applications.
- ✓ MongoDB will be used as the database.

2.5 Assumptions and Dependencies

The user has access to the internet and a modern web browser.

The application will be developed using React, Node.js, and MongoDB.

3. Functional Requirements

3.1 User Registration and Authentication

Users shall be able to register using an email and password.

Users shall be able to log in using their credentials.

Passwords shall be securely hashed before being stored in the database.

3.2 Dashboard

Upon logging in, users shall be directed to a dashboard showing a flowchart of their budget distribution.

The flowchart shall display percentages of the total budget allocated to each activity.

3.3 Activity Management

Users shall be able to create a new activity by providing a name and an allocated budget.

Users shall be able to view all their activities in a list.

Users shall be able to edit the details of an activity (e.g., change the budget).

Users shall be able to delete an activity.

3.4 Expense Tracking

Users shall be able to log expenses against each activity.

The progress bar for each activity shall update based on the expenses logged.

The flowchart shall update to reflect changes in the overall budget distribution.

3.5 Data Persistence

All user data (activities, budgets, expenses) shall be stored in a MongoDB database.

The back-end shall use Mongoose to interact with the database.

4. Non-Functional Requirements

4.1 Performance Requirements

The application should load within 2 seconds on a standard broadband connection.

CRUD operations should be completed within 1 second.

4.2 Security Requirements

All user data must be transmitted over HTTPS.

User passwords must be securely hashed using an industry-standard algorithm

The application should prevent unauthorized access to user data.

4.3 Usability Requirements

The application shall have an intuitive user interface that follows modern design principles.

The application shall be accessible on both desktop and mobile devices.

4.4 Reliability Requirements

The application should be available 99.9% of the time.

The system shall have mechanisms for data backup and recovery.

5. Design Constraints

The front-end must be developed using React.

The back-end must be developed using Node.js and Mongoose.

The database must be MongoDB.

6. Interface Requirements

6.1 User Interfaces

The landing page shall display the flowchart and a form to add new activities.

The dashboard shall list all activities with their corresponding progress bars.

Forms shall be provided for logging expenses and editing activity details.

6.2 Hardware Interfaces

The application shall not require any specific hardware beyond a device capable of running a modern web browser.

6.3 Software Interfaces

The application shall interface with the MongoDB database via Mongoose.

The application shall interface with external services for authentication if required (e.g., OAuth).

7. Other Requirements