House Wallet App Requirements Document

Table of Contents

1. Introduction

The purpose of this document is to outline the requirements for the development of the House Wallet web application. This document provides an overview of the system, its functions, and its requirements.

2. Glossary

This section provides a list of terms and definitions used throughout the document.

3. User Requirements Definition

This section outlines the requirements from the user's perspective. It includes a list of functional and non-functional requirements.

3.1 Functional Requirements

- Authentication for different Users Registration.
- Stay logged in for a year.
- Editing profile and budgeting settings.
- Budget Tracking & Analysis
- Expense Tracking.
- Filtering purchasing.
- Financial Limit & Goal Setting.
- Financial Reporting.
- Storing old budgeting details.

3.2 Non-functional Requirements

- **Security**: protect user data.
- **Performance**: fast and responsive.
- **Usability**: easy to use and user friendly.
- **Scalability**: able to handle large amounts of data and users as the app grows.
- **Compatibility**: compatible with different devices and browsers.

4. System Architecture

The House Wallet web application will follow the Model-View-Controller (MVC) architecture, consisting of three components:

4.1 Model

The model will represent the data and business logic of the application. It will consist of a database to store the user's financial data and functions to handle the data logic.The model will be implemented using MYSQL as the database .

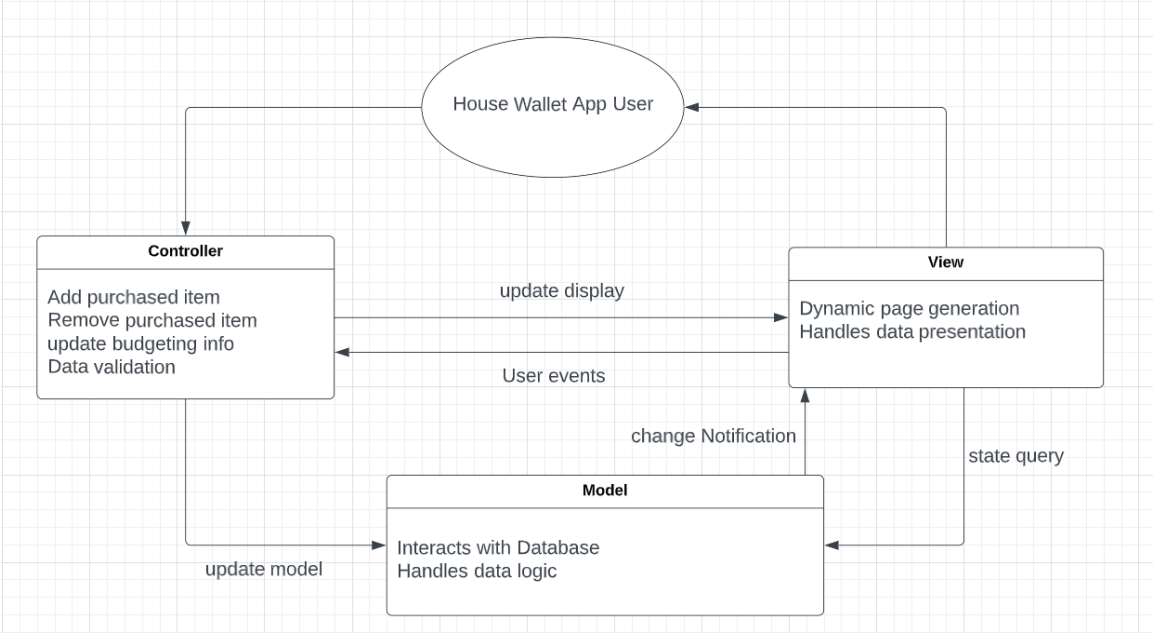
4.2 View

The view will be responsible for presenting data to the user and accepting user input. It will be built using HTML, CSS, and JavaScript on the front-end, and PHP on the back-end. The view will communicate with the controller to retrieve and update data.

4.3 Controller

The controller will handle user requests, retrieve and manipulate data from the model, and return responses to the view. It will be implemented using PHP. The controller will contain the business logic of the application and will communicate with the model to retrieve and manipulate data. The controller will also communicate with the view to present data to the user and receive user input.

The use of MVC architecture provides a clear separation of concerns between the model, view, and controller, allowing for easier maintenance and modification of the application.



5. System Requirements Specification

This section provides a detailed list of the system requirements, including functional and non-functional requirements.

5.1 Functional Requirements

- User authentication system and interfaces to allow users to create their accounts and log in securely.
- User profile management system to allow users to manage their account information and preferences.
- Session management system to keep the user logged in for a specified period.
- User profile management system to allow users to update their personal information and budgeting settings.
- Budget tracking system to allow users to create and manage their budgets.
- Budget analysis system to provide users with detailed reports and visualizations of their budgets.

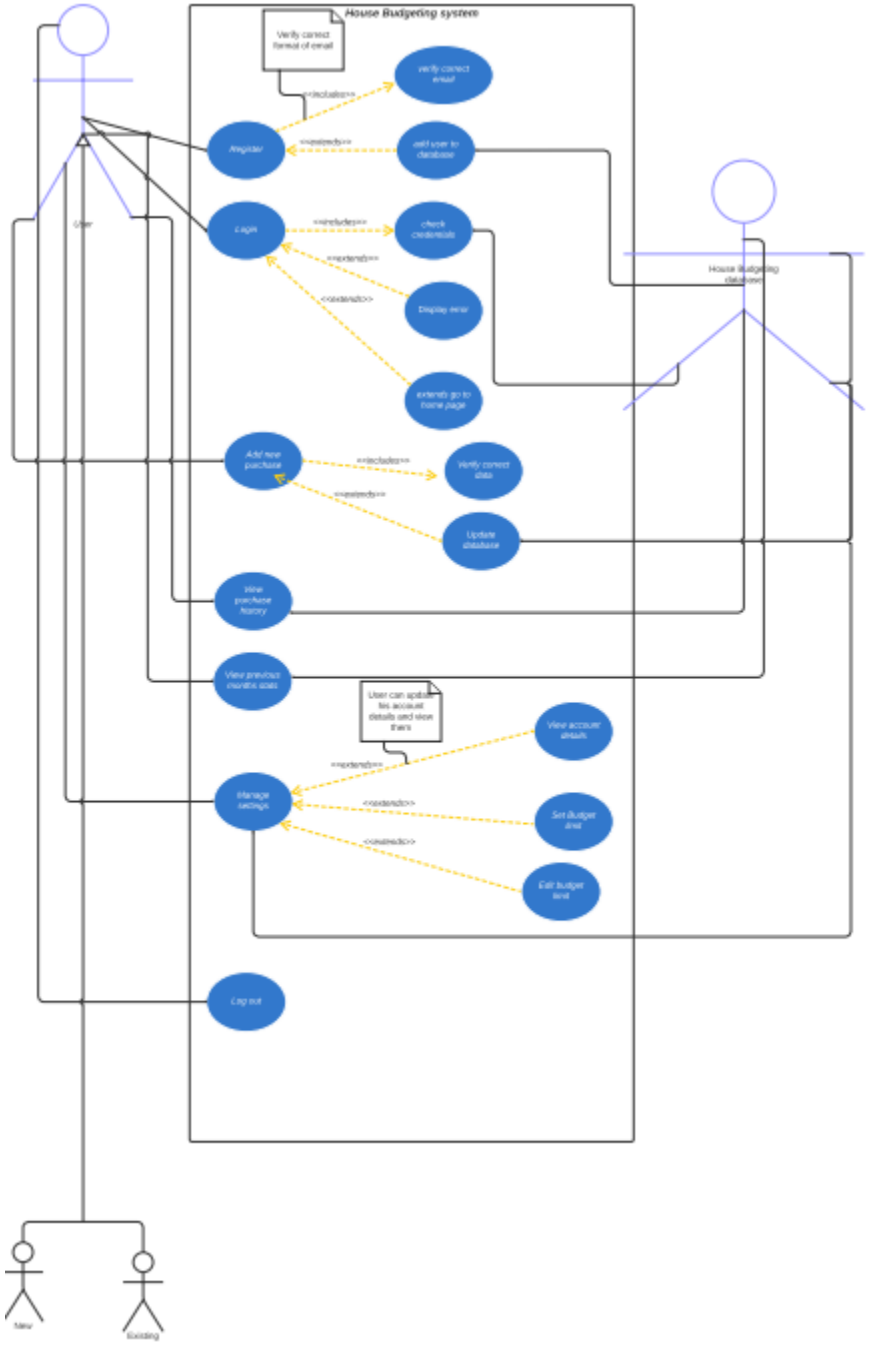
- Expense tracking system to allow users to track their expenses and categorize them for better analysis.
- Purchase filtering system to allow users to filter their purchases by date, category, or other criteria.
- Goal & Limit setting system to allow users to set financial goals and track their progress towards them.
- Reporting system to generate reports and visualizations to help users understand their financial situation and progress towards their goals.
- Data storage system to store old budgeting details for future reference.

5.2 Non-functional Requirements

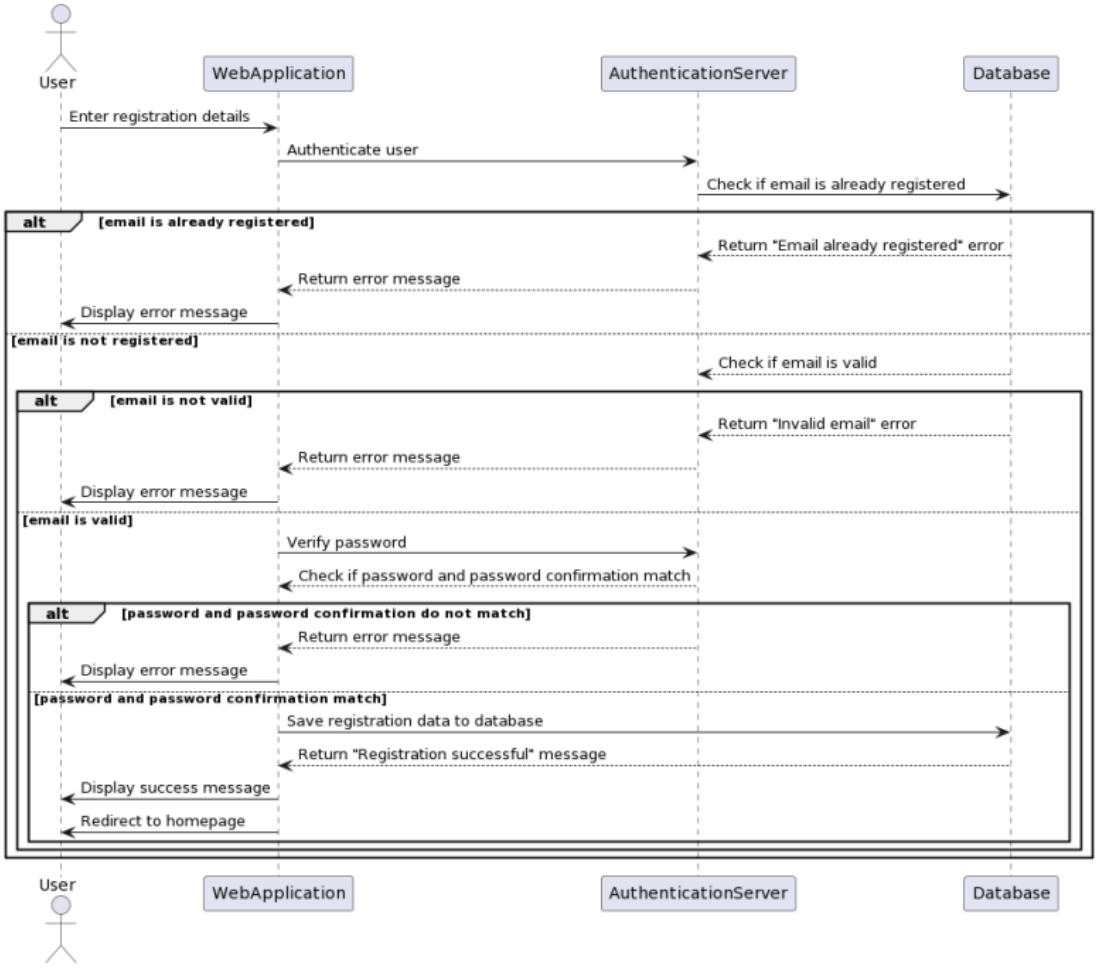
- Secure authentication system to protect user data, including login credentials and financial information.
- Secure data storage system to protect user data from unauthorized access.
- Fast and responsive application design to minimize lag and downtime.
- Efficient database design to handle large amounts of data.
- Scalable database design to handle an increasing number of users and data.
- User-friendly interface design to improve the user experience and ease of use.
- Responsive design to support different devices and screen sizes.
- Cross-browser compatibility to ensure the app works well on different browsers.
- Cross-device compatibility to ensure the app works well on different devices.

6. System Models

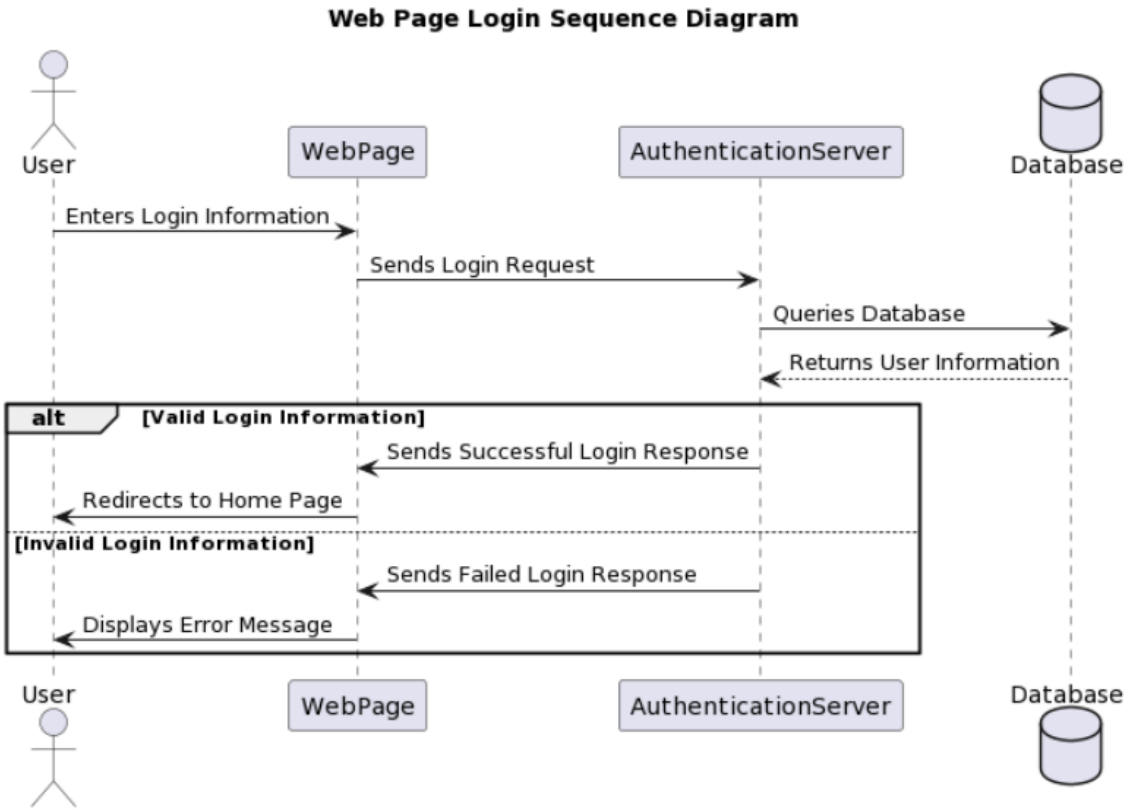
6.1 Use Case Diagram:



6.2 Registration Sequence Diagram:



6.3 Login Sequence Diagram:



7. System Evolution

This section outlines the system's evolution over time, including planned updates, upgrades, and maintenance.

8. Appendices

This section includes additional information that may be helpful for understanding the requirements, such as use cases, user stories, and data flow diagrams.

9. Index

This section provides an index of key terms and topics covered in the document.