High Level Design

Billova

version 3.0

Andreas Drozd, Benjamin Lichtenstein, Sergiu Iordanescu

Document history

Version	Status	Date	Responsible person	Reason for change
01	Done	16.11.2024	Benjamin Lichtenstein	Document Creation
02	Done	02.12.2024	Sergiu-Claudiu Iordanescu	Adjustments of the document.
03	Done	08.12.2024	Andreas Drozd, Benjamin Lichtenstein, Sergiu-Claudiu Iordanescu	Finish up and verify document.

Andreas Drozd, Benjamin Lichtenstein, Sergiu Iordanescu

Abbreviations

HLD …	High Level Design Document
SRS ...	Software Requirements Specification
AI …	Artificial Intelligence
UI …	User Interface
DB …	Database
GUI …	Graphical User Interface
API …	Application Programming Interface

Andreas Drozd, Benjamin Lichtenstein, Sergiu Iordanescu

Table of contents

1. Introduction

1.1. Purpose

The purpose of this High-Level Design (HLD) document is to outline the architecture and design approach for the Billova application, providing a clear framework for the development process. This document aims to bridge the gap between the Software Requirements Specification (SRS) and the detailed implementation by describing the system's overall structure, its primary components, and their interactions.
Key objectives of this document include:

- **Guiding Development**: Serving as a blueprint for developers to implement the system efficiently and consistently.
- **Ensuring Alignment**: Helping stakeholders understand how the system will fulfil the requirements defined in the SRS.
- **Identifying Risks and Dependencies**: Highlighting potential design risks and external dependencies early in the process.
- **Facilitating Review**: Providing a foundation for technical discussions and design reviews before coding begins.

This HLD document ensures that all stakeholders share a common understanding of the system's architecture and its alignment with the project's goals.

1.2. System Overview

The proposed system, Billova, is a personal finance tracking application designed to help users manage and monitor their expenses efficiently. The system leverages Optical Character Recognition (OCR) technology to scan and extract data from receipts, enabling automated input of spending details. It supports manual input for flexibility and accommodates scenarios where OCR fails, or network connectivity is unavailable.

The application offers core functionalities including:

- Adding receipts via scanning or manual entry.
- Categorizing and storing expenses in a secure database.
- Viewing, searching and sorting expense records by category or date.
- Generating monthly spending summaries to provide financial insights.
- Managing user accounts with customizable settings, such as currency preferences and profile details.

The system is intended to function across various device platforms, including mobile, tablet, and desktop, with a responsive user interface. It aims to address the following objectives:

- Empower users to better control their personal finances through streamlined receipt management.
- Provide clear and actionable insights into spending patterns using graphs and summaries.
- Maintain data privacy by ensuring account-specific access to stored information.

The architecture will integrate OCR capabilities, database management, and user authentication as core components, ensuring seamless operation and secure data handling. The system will also allow for future enhancements such as AI-driven insights, multi-user collaboration, and synchronization with bank accounts for automated financial tracking.

1.3. Definitions

This section defines terms and abbreviations used throughout this document to ensure a common understanding among all stakeholders.

- **OCR (Optical Character Recognition)**: Technology used to convert different types of documents, such as scanned paper documents, PDF files, or images captured by a camera, into editable and searchable data.
- **Expense Categories**: User-defined or system-recommended classifications for grouping similar expenses (e.g., Food, Transportation, Utilities).
- **Responsive Design**: An approach to web development ensuring the user interface adapts seamlessly to different device screen sizes, including mobile phones, tablets, and desktops.
- **SRS (Software Requirements Specification)**: A document that outlines the functional and non-functional requirements of the system.

- **Database**: A structured collection of data stored electronically, used to manage and retrieve user and system data.
- **Currency Management**: The ability for users to select and display financial data in their preferred currency.
- **Error Handling**: Mechanisms within the system to detect and communicate issues, such as failed OCR operations or lack of network connectivity.

1.4. Operating Conditions

The Billova application is designed to operate under the following conditions to ensure optimal functionality and user experience:

Technical Requirements

- **Internet Connection**: Required for:
  - Communicating with the AI API for OCR functionality.
- **Device Compatibility**: The system must function on devices with:
  - Modern web browsers (e.g., Chrome, Firefox, Safari, Edge) supporting HTML5.
  - Cameras for scanning receipts (for mobile and tablet users).
- **Server Environment**: The backend must run on a cloud-hosted server capable of handling:
  - OCR processing.
  - Database transactions.
  - User authentication requests.

User Requirements

- **Accounts**: Each user must have a registered account to access the system.
- **Devices**: A smartphone, tablet, or computer with sufficient hardware capabilities to run the application efficiently.

Constraints

- **Offline Mode**: Limited functionality available without internet access (e.g., manual entry of expenses).

1.5. Document Overview

- Architecture Plan
- Class Diagram
- Activity Diagram – Scan Receipts
- Activity Diagram – Change Settings
- Activity Diagram – Sort Expenses
- Sequence Diagram – Scan a document successfully
- Sequence Diagram – Display Expenses
- Sequence Diagram – Manual Receipt Entry

2. Backend Module Overview

2.1.1. Role of the Backend in the overall system

Role: Acts as the core processing unit, managing business logic, data flow, and communication between the frontend and external services (e.g., OCR API, database).
Interaction: Handles requests from the frontend and processes them using the database and external APIs.

The backend will be implemented via a Django API. Using an API ensures separation of concerns between the backend and frontend as well as scalability.

2.1.2. Functionality of the Backend

- Business Logic: Processes user requests, such as adding expenses, sorting data, and generating reports.
- Data Integration: Manages data flow between the frontend, database, and external APIs (e.g., OCR service).
- Error Handling: Detects and reports issues, such as failed OCR scans or network connectivity problems.

3. Database Module Overview

3.1.1. Role of the Database in the overall system

Role: Stores all user data, including accounts, receipts, expenses, categories, and settings.
Interaction: Provides persistent storage for the application and enables quick retrieval and updates of data.

A relational database will be used to store all the information for the application.

3.1.2. Functionality

- Data Storage: Stores user accounts, receipts, expenses, and custom settings.
- Data Security: Ensures that stored data is accessible only to the account holder.
- Data Retrieval: Supports fast and efficient querying of user data for display or processing.

4. Expense Service Module Overview

4.1.1. Role of Expense Service in the overall system

Performs Optical Character Recognition on images of receipts to extract text data.
Interaction: Processes receipt images and returns structured text data to the backend.

4.1.2. Functionality

- Text Recognition: Extracts textual data from receipt images using Optical Character Recognition (OCR) technology.
- Data Structuring: Converts recognized text into structured information, including vendor name, date, and total amount.
- Fallback Mechanism: Notifies the user if OCR fails and provides an option for manual data entry.

5. User Service Module Overview

5.1.1. Role of User Service in the overall system

Manages user registration, login, and session management.
Interaction: Validates user credentials and provides secure access to account-specific data.

5.1.2. Functionality

- User Registration: Allows new users to sign up and create accounts.
- Login/Logout: Manages secure user login and logout processes.
- Session Management: Maintains user sessions for authenticated access to the application.

6. Frontend Module Overview

6.1.1. Role of the Frontend in the overall system

Role: Provides the user interface (UI) for interacting with the system, including adding receipts, viewing expenses, and managing account settings.
Interaction: Sends user inputs to the backend for processing and displays results returned from the backend.

The frontend will be implemented using Bootstrap and JavaScript.

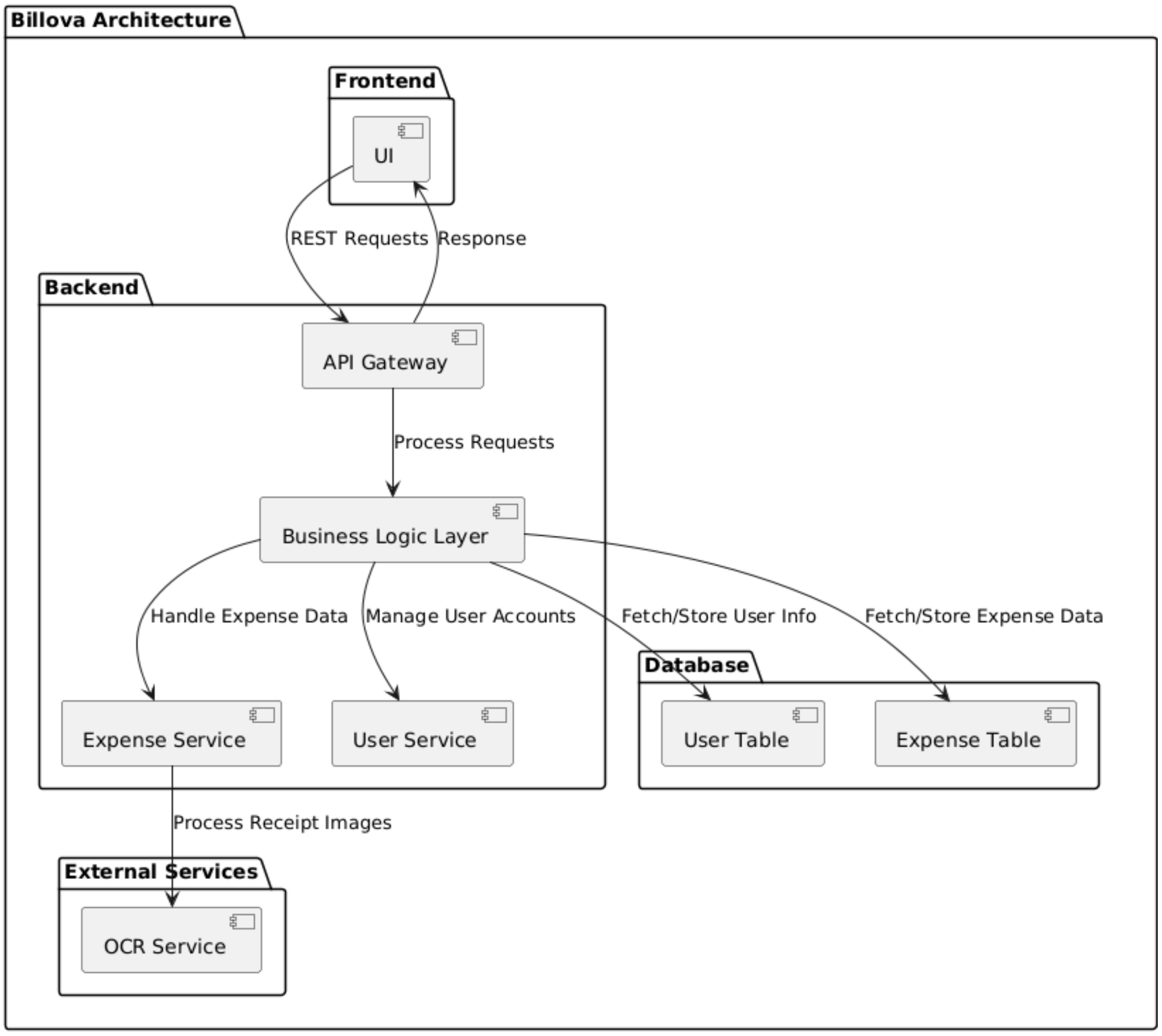
6.1.2. Functionality of the Frontend

- User Interaction: Enables users to interact with the system through an intuitive and responsive UI.
- Receipt Management: Allows users to scan receipts using the camera
- Expense Management: Provides options to manually input expenses and categorize them.
- Expense Visualization: Displays expense summaries and graphs.
- Search and Filter: Allows users to search and sort expenses by date, category, or amount.

- Settings: Supports customization of user preferences, such as currency and categories.

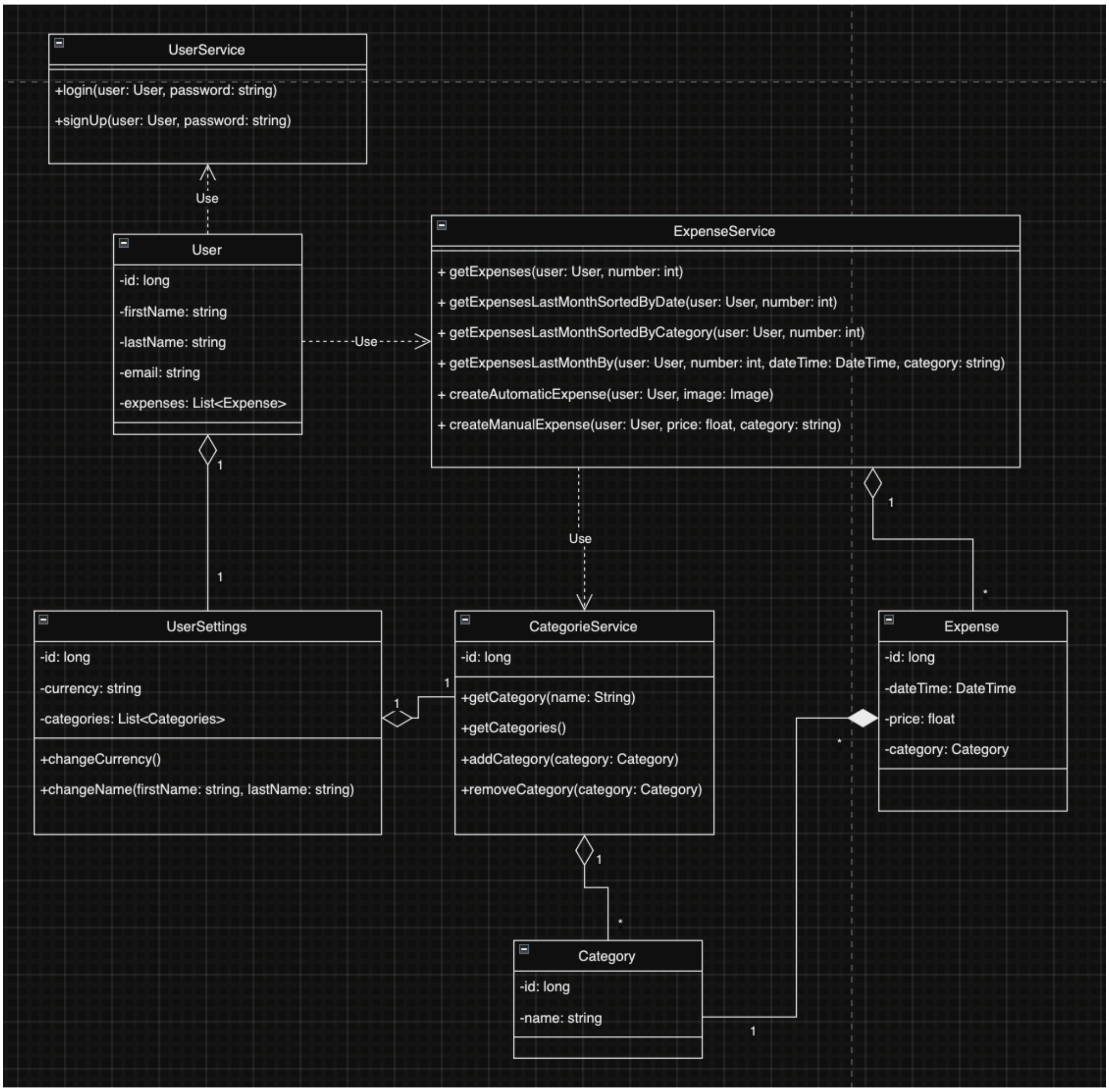
7. Detailed Design

7.1.1. Architecture



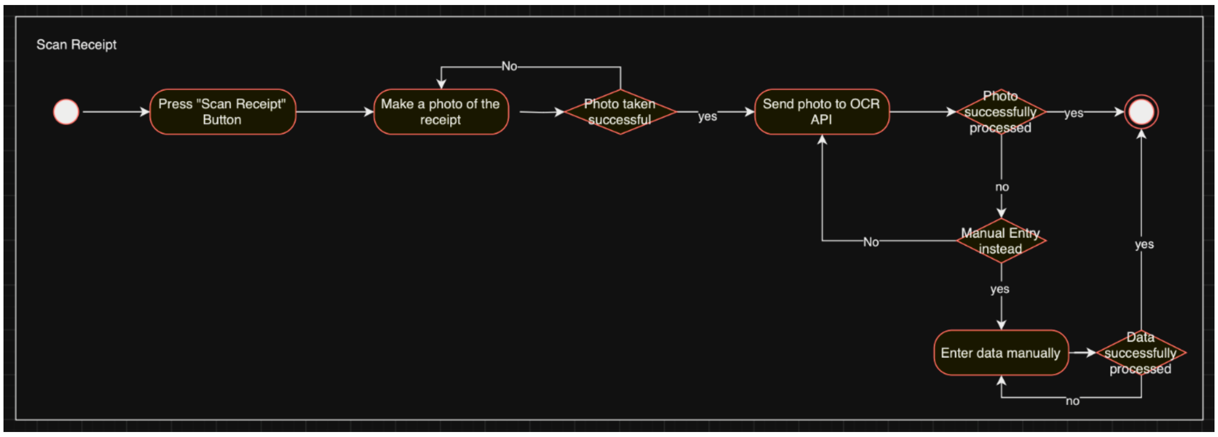
7.2. UML Diagrams

7.2.1. Class Diagram

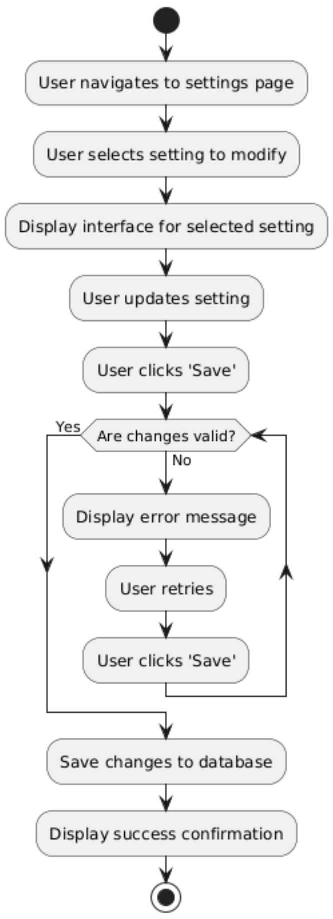

UserService
+login(user: User, password: string)
+signUp(user: User, password: string)
Use
User
-id: long
-firstName: string
-lastName: string
-email: string
-expenses: List<Expense>
Use
ExpenseService
+ getExpenses(user: User, number: int)
+ getExpensesLastMonthSortedByDate(user: User, number: int)
+ getExpensesLastMonthSortedByCategory(user: User, number: int)
+ getExpensesLastMonthBy(user: User, number: int, dateTime: DateTime, category: string)
+ createAutomaticExpense(user: User, image: Image)
+ createManualExpense(user: User, price: float, category: string)
1
1
Use
1
*
UserSettings
-id: long
-currency: string
-categories: List<Categories>
+changeCurrency()
+changeName(firstName: string, lastName: string)
1
1
CategorieService
-id: long
+getCategory(name: String)
+getCategories()
+addCategory(category: Category)
+removeCategory(category: Category)
Expense
-id: long
-dateTime: DateTime
-price: float
-category: Category
*
1
*
Category
-id: long
-name: string
1

7.2.2. Activity Diagram

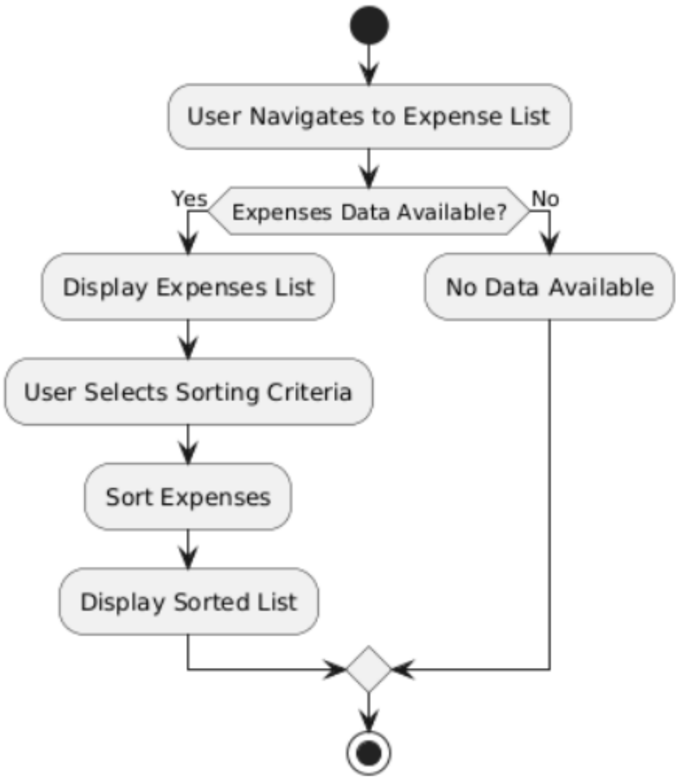
7.2.2.1. Scan Receipts



7.2.2.2. Change Settings

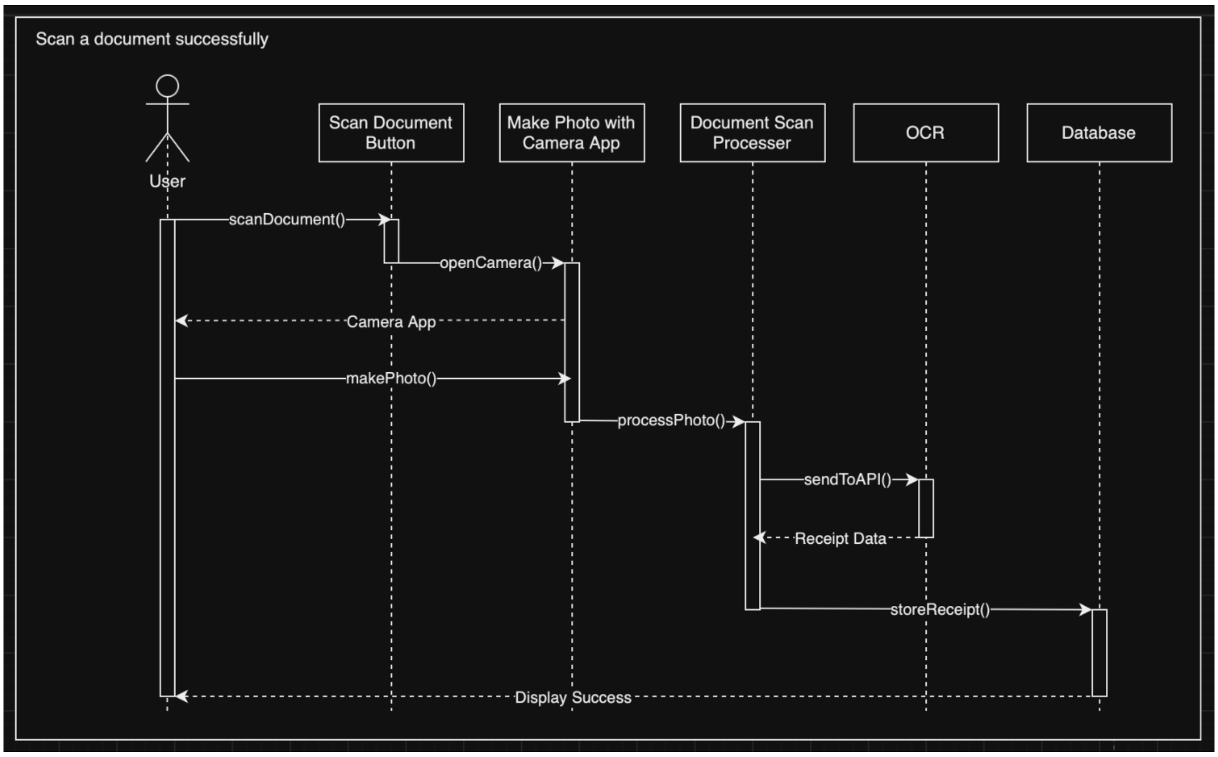


7.2.2.3. Sort Expenses

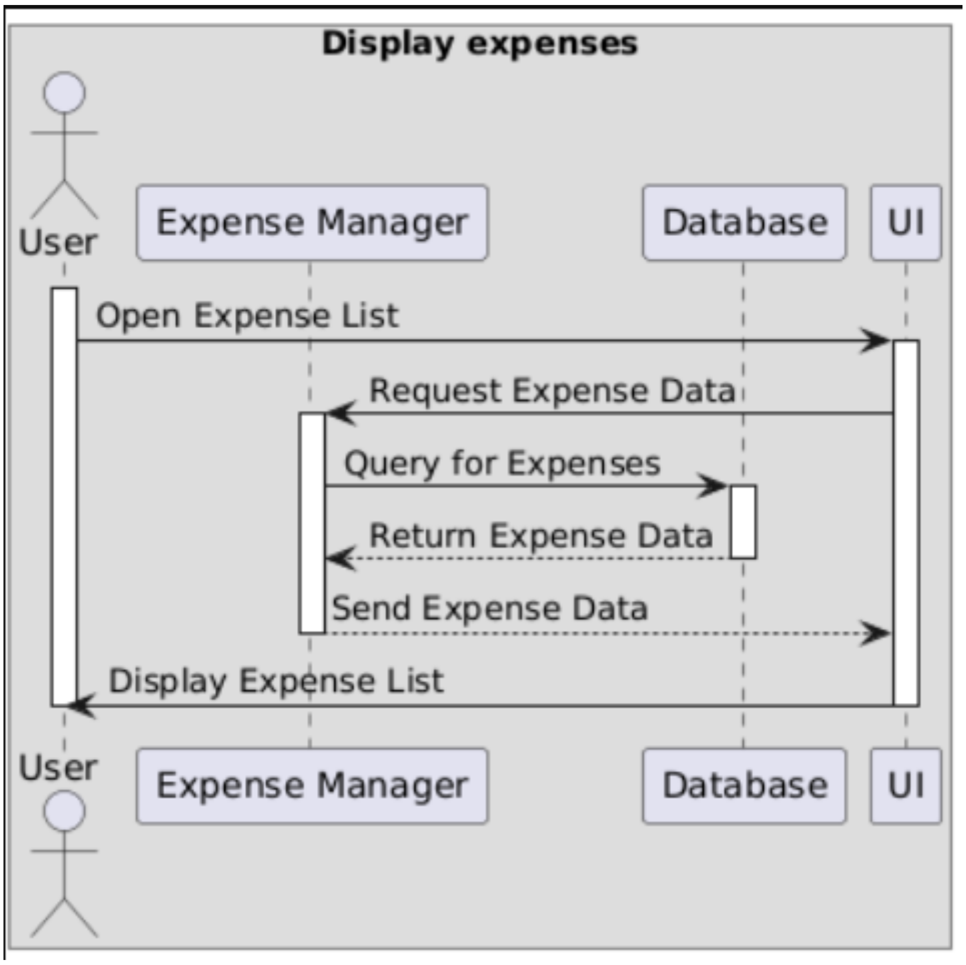


7.2.3. Sequence Diagram

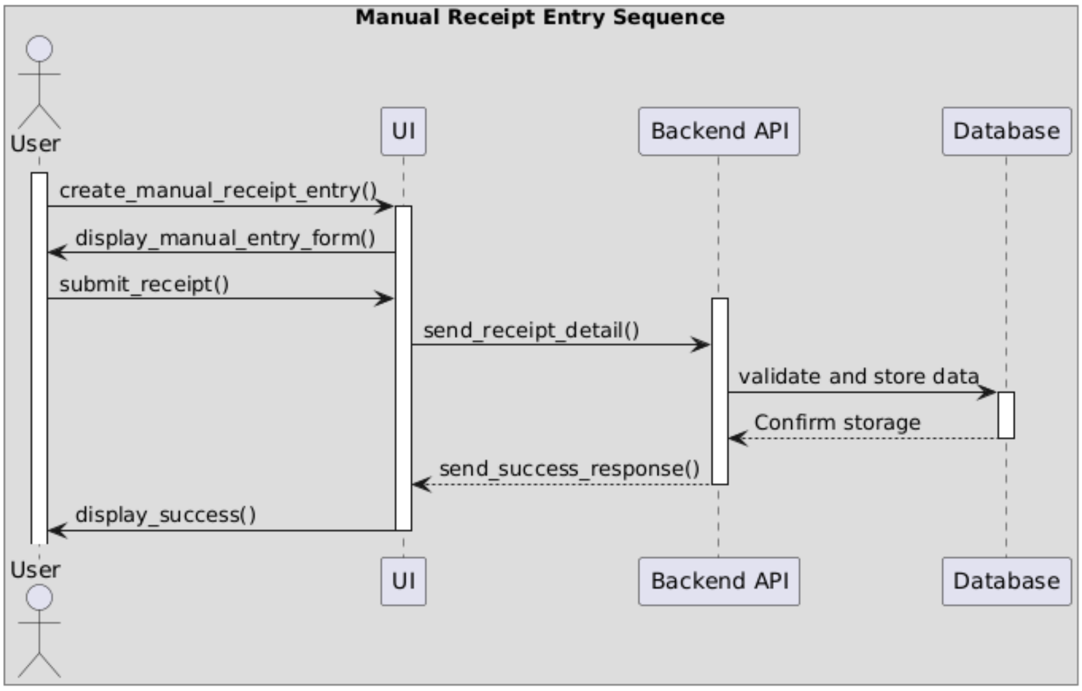
7.2.3.1. Scan a document successfully



7.2.3.2. Display Expenses



7.2.3.3. Manual Receipt Entry


Manual Receipt Entry Sequence
User
UI
Backend API
Database
create_manual_receipt_entry()
display_manual_entry_form()
submit_receipt()
send_receipt_detail()
validate and store data
Confirm storage
send_success_response()
display_success()
User
UI
Backend API
Database