Software Requirements Specification (SRS)

**Project Title:** Budget Tracker Web Application

# Introduction

* 1. **Purpose**

The Budget Tracker Web Application provides users with an intuitive platform to efficiently manage personal finances. It enables tracking incomes and expenses, budget setting, and generation of financial reports to help users understand and control their spending habits effectively.

# Document Conventions

* **MongoDB**: NoSQL Database
* **Express.js**: Web framework for Node.js
* **EJS:** Embedded JavaScript templating engine
* **Node.js**: JavaScript runtime environment
* **API**: Application Programming Interface
* **UI**: User Interface
* **JWT**: JSON Web Token for authentication
* **Bcrypt:** Password-hashing function for secure storage

# Intended Audience and Reading Suggestions

* **Developers**: To understand technical and functional requirements for system implementation.
* **Testers**: To plan test cases and testing strategies.
* **Project Managers**: For planning, tracking, and managing development progress.
* **Users**: To understand the application's features and usage.

Readers should have a foundational understanding of web development, database systems, and API concepts.

# Project Scope

The Budget Tracker is a web-based application accessible through any internet-enabled device. It provides functionalities such as:

* User registration and secure login.
* Adding, editing, and deleting income and expense entries.
* Categorization, filtering, and recurrence of transactions
* Visual representation of financial data through graphs and charts.
* Pinned transactions for quick access
* Data storage using MongoDB Atlas.

The application is developed using the MongoDB, Express.js, EJS templating engine, Node.js to ensure scalability, flexibility, and performance.

# References

* Lecture 2 SRS Guidelines Document
* Official MERN Stack Documentation
* EJS Documentation
* OWASP Web Security Guidelines
* GDPR and Australian Privacy Principles (APP) Guidelines

# Overall Description

* 1. **Product Perspective**

The Budget Tracker is a standalone product. It incorporates third-party libraries like Chart.js for data visualization and JWT for secure authentication. It is intended to be deployed on cloud platforms and optimized for all major browsers.

# Product Functions

The core functionalities include:

* User registration, login, and logout.
* Dashboard displaying an overview of finances.
* Adding, editing, deleting, and categorizing financial transactions.
* Recurring Transactions - Set transactions to recur monthly (optional toggle)
* Filtering transactions based on date and category.
* Pinned Transactions - Mark important records for quick access
* Visual financial reports through charts.
* Secure data storage and retrieval.

# User Classes and Characteristics

* **Registered Users**: Full access to application features, including transaction management and budget management.
* **Guests** (Future Scope): Access to a demo version without saving data.
* **Administrators** (Future Scope): Manage users and monitor platform usage.

# Operating Environment

* Web Browsers: Chrome, Firefox, Safari, Edge
* Database: MongoDB Atlas
* Server: Node.js hosted on cloud services (e.g., Render, AWS, Vercel)
* Devices: Desktops, tablets (responsive design)
* Frontend: EJS + Tailwind
* Backend: Node.js + Express.js RestAPIs

# Design and Implementation Constraints

* MERN Stack is mandatory.
* Backend APIs must be RESTful.
* Authentication must use JWT securely.
* Compliance with security best practices (e.g., input validation, password encryption).

# User Documentation

* Basic User Manual upon launch.
* In-app tooltips and help section for user guidance.

# Assumptions and Dependencies

* Users have access to a stable internet connection.
* The application will be compatible with major modern browsers.
* Hosting services will ensure uptime and data reliability.
* Payment gateway integration is not included in the initial release.

# Use Case Diagram

The use case diagram outlines the interactions between different user roles and the system functionalities of the platform. It highlights three main actors: **Registered Users**, **Guests**, and **Administrators**. Each actor interacts with specific use cases, which represent the system’s functional requirements.

# Actors s Their Use Cases

1. **Registered Users**

These users have full access to the platform’s core features. Their use cases include:

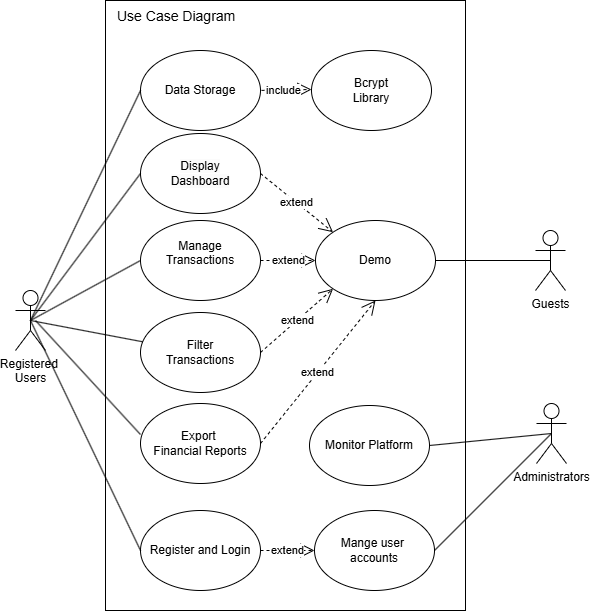
* + **Register and Login** *(extends* Manage user accounts): Enables users to securely authenticate into the system. This use case implies encryption standards (e.g., bcrypt).
  + **Display Dashboard**: Shows a summarized view of transactions, reports, and platform activity.
  + **Manage Transactions** *(extends* Demo): Allows users to add, update, or delete financial transactions.
  + **Data Storage** *(includes* Bcrypt Library): Ensures secure storage of data, particularly user credentials.

## Guests

* + **Demo**: Guests can access a limited demo of the system’s core functionalities. The use cases Manage Transactions extend from this demo for preview purposes.

## Administrators

* + **Monitor Platform**: Enables administrators to oversee system activity, performance, and usage analytics.
  + **Manage User Accounts**: Admins can manage user data, deactivate accounts, and reset passwords.



# Specific Requirements

* 1. **Functional Requirements User Authentication**
* **FR1**: Users must be able to register using email and password.
* **FR2**: Users must be able to securely log in and log out.

# Transaction Management

* **FR3**: Users can add income and expense transactions.
* **FR4**: Users can edit and delete transactions.
* **FR5**: Users can categorize transactions.
* **FR6:** Users can make transactions recurring
* **FR7**: Users can Pin important transactions

# Data Visualization

* **FR8**: The application should display charts summarizing income, expenses, and overall balance.

# Security

* **FR9**: Passwords must be securely hashed using bcrypt before being stored in the database.
* **FR10**: Sessions must expire after a defined period of inactivity.

# External Interface Requirements

* + 1. **User Interfaces**
       - Responsive design using EJS.
       - Mobile-first UI approach with accessibility features.

# Hardware Interfaces

* + - * No special hardware required; any device with a modern browser is sufficient.

# Software Interfaces

* + - * **MongoDB Atlas**: Database service.
      * **Express.js**: Server-side API handler.
      * **JWT**: For secure user sessions.
      * **Chart.js**: For visual data representation.

# Communications Interfaces

* + - * Data exchanged between client and server must use HTTPS protocol.

# Performance Requirements

* Application should load within **3 seconds** under standard conditions.
* API responses should be below **500ms** under normal load.
* The initial system should support up to **100 concurrent users**.

# Logical Database Requirements

* **Users Collection**:

Fields: UserID, FullName, Email (unique, validated format), Password (hashed), ProfileImageUrl, IsAdmin, CreatedAt, UpdatedAt

# Income Collection:

Fields: IncomeID, UserID, Source, Amount (validated to ensure non-negative), Icon, Date, CreatedAt, UpdatedAt

# Expense Collection:

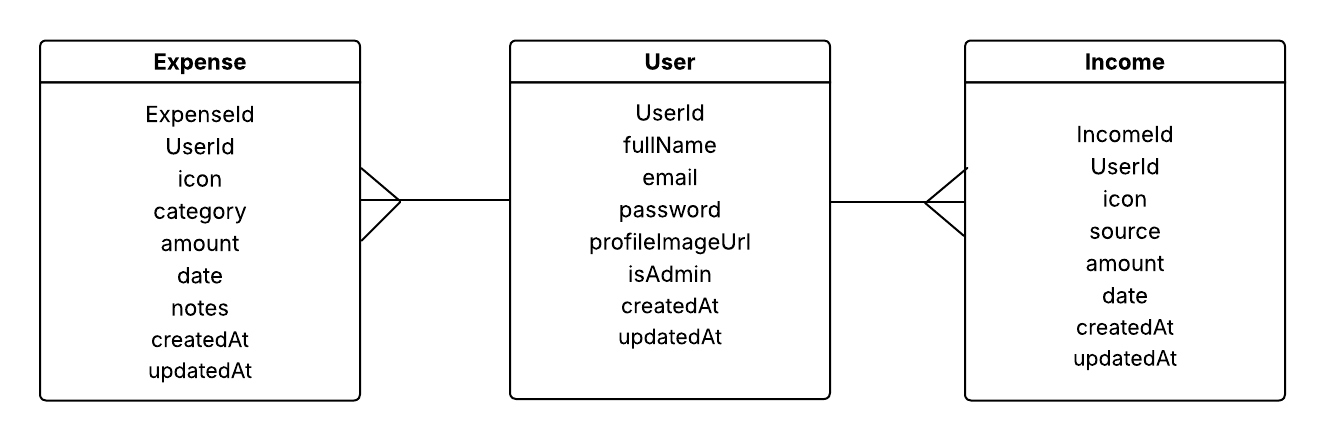
Fields: ExpenseID, UserID, Category, Amount (validated to ensure non-negative), Icon, Notes, Date, CreatedAt, UpdatedAt

* All collections will implement validation using Mongoose ORM.
* Required fields will be enforced (e.g., Email and Password in Users, Amount in Income/Expense).
* Passwords will be stored as hashed strings using bcrypt.
* Numeric fields like Amount will be validated to ensure positive values.

# Design Constraints

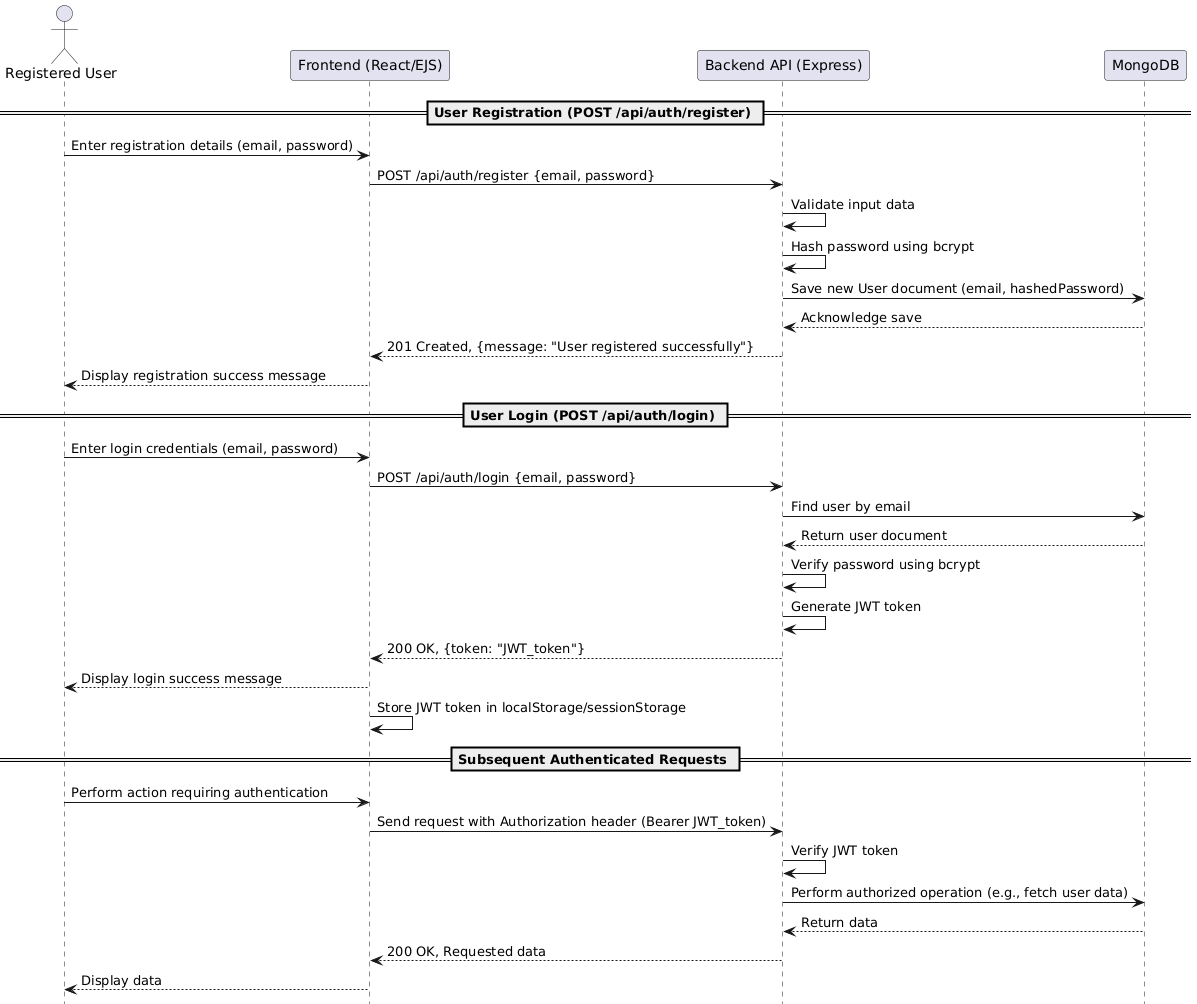
* Mandatory use of the MERN stack
* RESTful API design.
* Adherence to data security and privacy regulations.

# Entity Relationship Diagram (ERD)

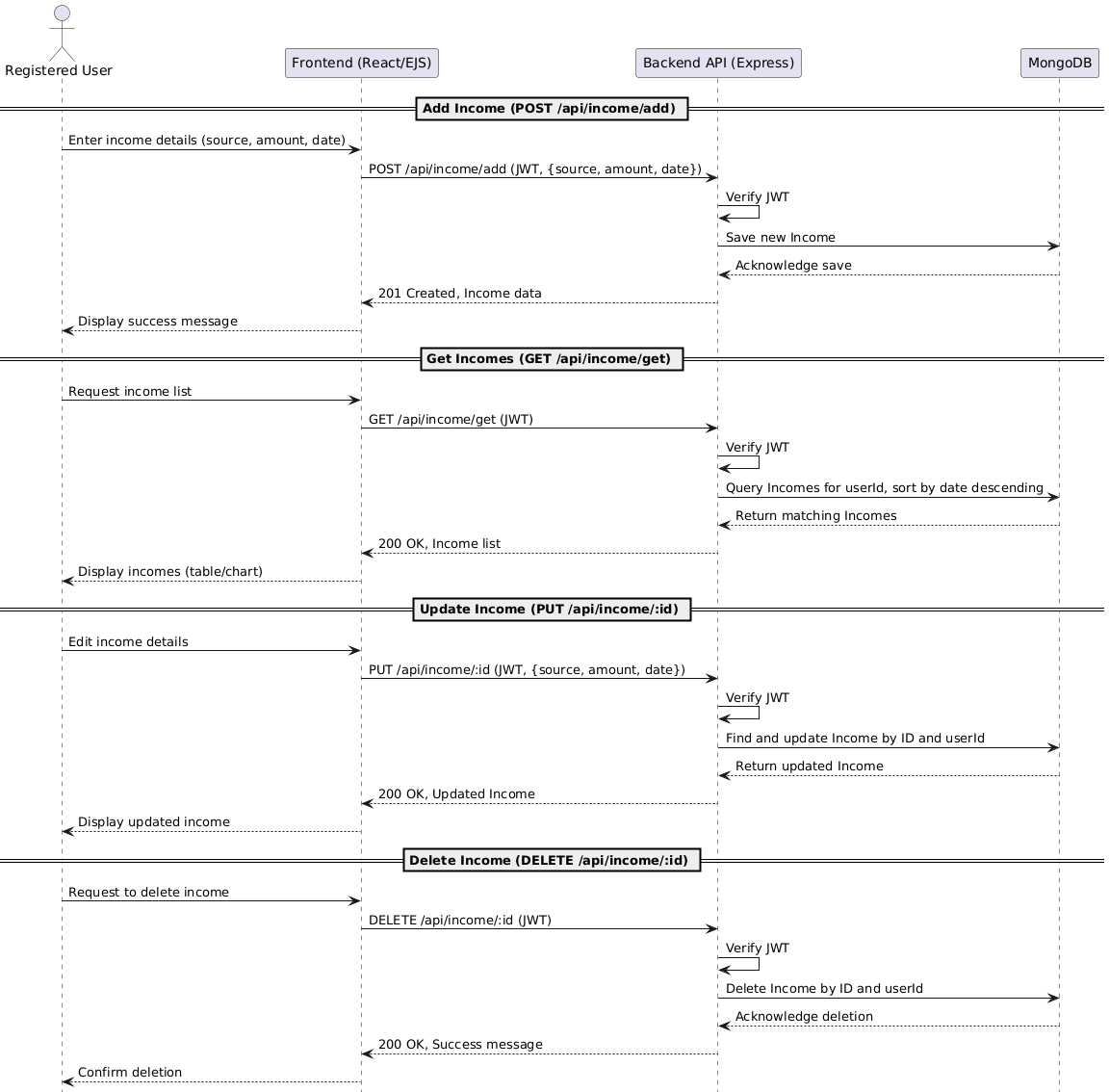


# Sequence Diagrams

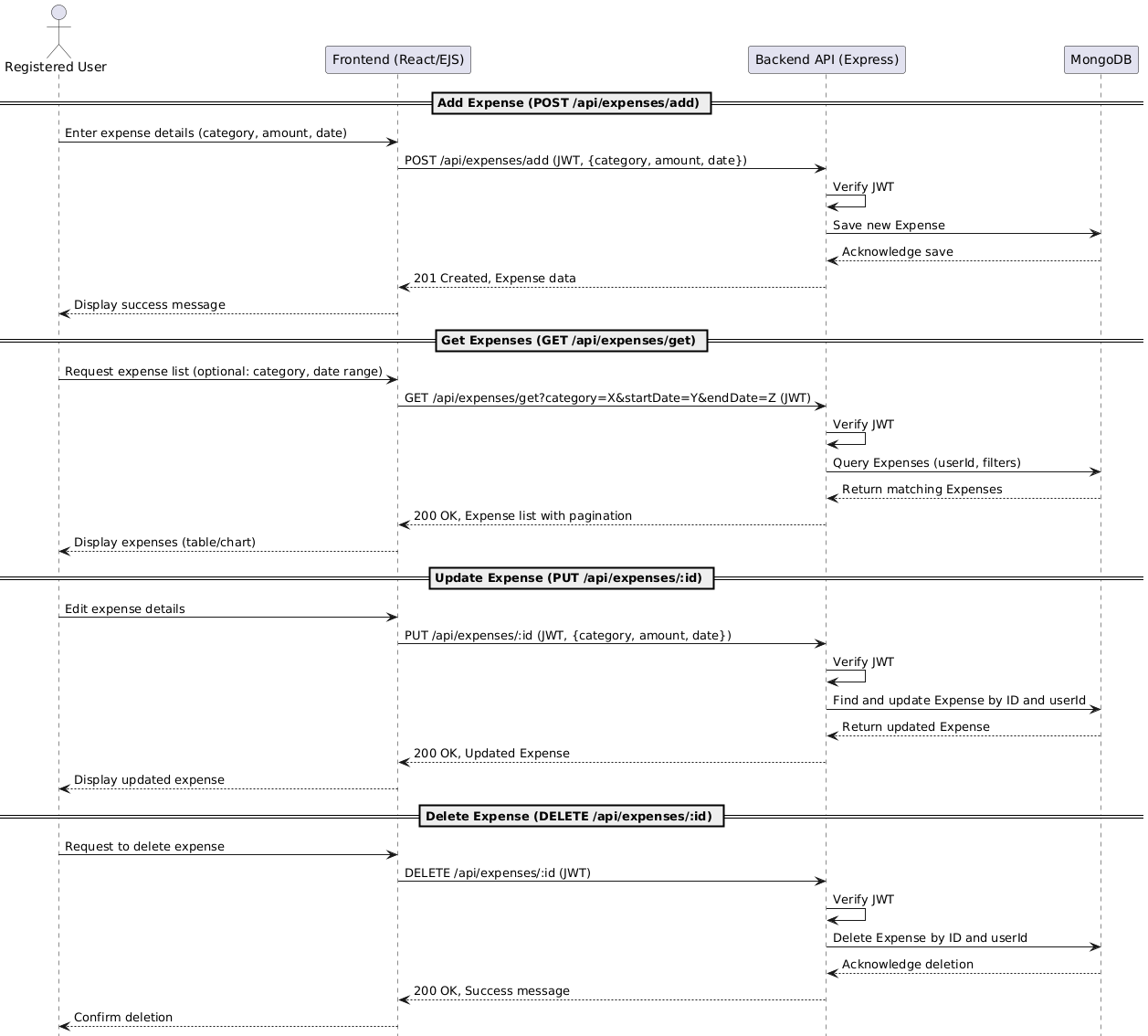
**For authentication:**



**For income:**



**For expense:**



1. **Appendices**

* Future features may include:
  + Multi-currency support.
  + Recurring transactions management.
  + Payment gateway integrations.

# Data Privacy Compliance

The system will comply with the Australian Privacy Principles (APPs) under the Privacy Act 1988 and aligns with international standards such as the General Data Protection Regulation (GDPR). User data will be securely stored, encrypted where appropriate, and will not be shared with third parties without explicit consent.

**Member Contributions to the SRS**

|  |  |
| --- | --- |
| **Member Name** | **Contribution** |
| Muhammad Haider Ali | Drafted the Introduction, Overall Description, and Specific Requirements sections. |
| Duo Zhang | Defined External Interfaces, Performance, Database Requirements, User Case Diagram and Sequence Diagrams. |
| Anushri Jawahar | Structured document added User Documentation and Design Constraints. Updated the final SRS to Include new features |
| Anjitha Varghese | Structured Logical Database Requirements, Data Privacy Compliance and ERD. |
| Pascaline Jepkemboi | Reviewed and refined assumptions added References. |