**Smart Expense Tracker**

**Github Link:** [github.com/Sivaneshmani/smart\_expenseive\_tracker](http://github.com/Sivaneshmani/smart_expenseive_tracker)

## **1. Project Overview**

The Smart Expense Tracker with Budget Optimization is a web application designed to help users track their expenses, manage monthly budgets, and receive AI-driven insights and alerts regarding their spending behavior. The application leverages AI to recommend personalized monthly budgets and detect anomalies in user spending patterns.

## **2. Objective**

* **Track user expenses** with the ability to categorize, filter, and manage financial data.
* **Optimize monthly budgets** based on user spending history.
* **Alert users to unusual spending behavior** by identifying anomalies in transaction history.

## **3. Technologies Used**

### **Frontend:**

* **React with Vite**: React is used for building the frontend, with Vite for fast development and bundling. Functional components and React Hooks are used for state and lifecycle management.
* **TailwindCSS**: A utility-first CSS framework used for styling the application, enabling responsive and modern UI design.
* **Redux Toolkit**: For efficient state management across the app, particularly for handling user authentication and managing expense data.

### **Backend:**

* **Node.js with Express**: The backend server is built using Node.js and Express, providing an API for the frontend to interact with.
* **JWT-based Authentication**: Secure user authentication using JWT tokens for safe login and session management.
* **MongoDB**: A NoSQL database used for storing user data, including expenses, budgets, and other financial information.

### **AI Features:**

* **Spending Pattern Analysis**: Analyzes the user's past transactions to identify patterns in spending.
* **Monthly Budget Recommendations**: Provides AI-driven suggestions for personalized budgets based on previous spending history.

## **4. Core Features**

### **User Authentication:**

* **Signup/Login**: Allows users to sign up or log in to their accounts securely using JWT tokens.
* **User Profile**: Users can set their profile details, such as preferred currency and expense categories.

### **Expense Management:**

* **Add Expenses**: Users can add expenses with key details: amount, category, notes, and payment method.
* **View Expenses by Date Range**: Users can filter their expenses based on a selected date range.
* **Edit/Delete Expenses**: Users can modify or remove their expense records.
* **Filter by Category or Payment Method**: Users can filter their expenses by category (e.g., food, entertainment) or payment method (e.g., credit card, cash).

### **Budgeting:**

* **Set Monthly Budget Goals**: Users can set budget limits for each category (e.g., $500 for groceries, $200 for entertainment).
* **Track Current vs Budgeted Spending**: The app tracks how much users have spent compared to their set budget for each category.

### **AI Budget Optimization:**

* **Analyze Previous Months' Spending**: The AI component analyzes past spending data to determine trends and user behavior.
* **Recommend Personalized Monthly Budgets**: Based on historical spending patterns, the AI suggests personalized monthly budgets for each category.
* **Alert for Exceeding Budgets/Anomalous Activity**: The system alerts users if their spending exceeds the recommended budget or if it detects anomalies, such as unusually high spending in a category.

### **Dashboard:**

* **Charts for Category-Wise Spending**: The dashboard displays interactive charts for users to visually analyze their spending across different categories.
* **Monthly/Weekly Financial Summary**: A summary view of income, expenses, and budget tracking for different time intervals.

## **5. Application Flow**

1. **User Signup/Login**: The user is required to authenticate through the frontend, where they can create an account or log in using JWT tokens.
2. **Expense Management**: After logging in, users can start adding expenses with necessary details like amount, category, and payment method. They can view, edit, and delete entries.
3. **Budget Setting**: Users can set monthly budgets for each category. The system compares real-time spending to these budgets.
4. **AI Analysis**: The AI processes the user's spending data to recommend optimized budgets and detect anomalies.
5. **Dashboard**: The user can view their spending summary, including category-wise breakdown, monthly/weekly summaries, and AI recommendations for budget adjustments.

## **6. Folder Structure Guidelines**

To ensure clarity and maintainability, the project will follow a structured folder layout:

* **/src**: This is where all the source code for your application lives.
  + **/components**: Reusable UI components like buttons, form inputs, etc.
  + **/pages**: These represent complete pages/views in your app (e.g., homepage, about page).
  + **/context**: Typically used for managing global state in a React app.
  + **/utils**: General utility functions that can be used across your app (e.g., date formatters, API request helpers).
* **/backend**: This contains all your backend-related code.
  + **/middleware**: Functions to handle requests, such as authentication or logging.
  + **/models**: Defines your database schemas or models.
  + **/routes**: Handles different API endpoints for CRUD operations or services.
* **/public**: Static files (e.g., images, stylesheets) that are publicly accessible by the frontend.

## **7. Customer API Tests**

**1. Customer Signup**

API Endpoint: POST /api/signupRequest Payload:json

{ "name": "abchh",

"email": "abggc@gmail.com",

"password": "ewb$aLtNMq" }

Status Code: 201 CreatedResponse: Customer successfully created.

**2. Customer Login**

API Endpoint: POST /api/loginRequest Payload:json

{ "email": "abggc@gmail.com",

"password": "ewb$aLtNMq" }

Status Code: 201 CreatedResponse: Customer successfully logged in.Business Logic API Tests

**3. Expenses**

API Endpoint: GET /api/expensesStatus Code: 200 OKResponse: Successfully fetched expenses.

**4. Income**

API Endpoint: GET /api/income/getStatus Code: 200 OKResponse: Successfully fetched income.

**5. User Profile**

API Endpoint: GET /api/user/profileStatus Code: 200 OKResponse: Successfully fetched user profile.

**6. Expense Edit**

API Endpoint: PUT /api/expense/{expenseId}Request Payload:json

{ "amount": 500,

"description": "Updated expense" }

Status Code: 200 OKResponse: Successfully updated expense.

**7. Income Edit**

API Endpoint: PUT /api/income/{incomeId}Request Payload:json

{ "amount": 2000,

"source": "Salary" }

Status Code: 200 OKResponse: Successfully updated income.

**8. Budget Retrieval**

API Endpoint: GET /api/budgets/getStatus Code: 200 OKResponse: Successfully fetched budget details.

**9. Expense Retrieval**

API Endpoint: GET /api/expensesStatus Code: 200 OKResponse: Successfully fetched expenses.

**10. Income Retrieval**

API Endpoint: GET /api/income/getStatus Code: 200 OKResponse: Successfully fetched income.

**11. Balance**

API Endpoint: GET /api/user/balanceRequest Payload:json

{ "income": 0,

"expenses": 74942,

"savings": -74942,

"total": -74942,

"budget": 2500,

"currency": "INR" }

Status Code: 200 OKResponse: Successfully fetched balance.

**12. Monthly Budget Status**

API Endpoint: GET /api/monthly-budget?userId=${userId}Status Code: 200 OKResponse: Successfully fetched monthly budget status.

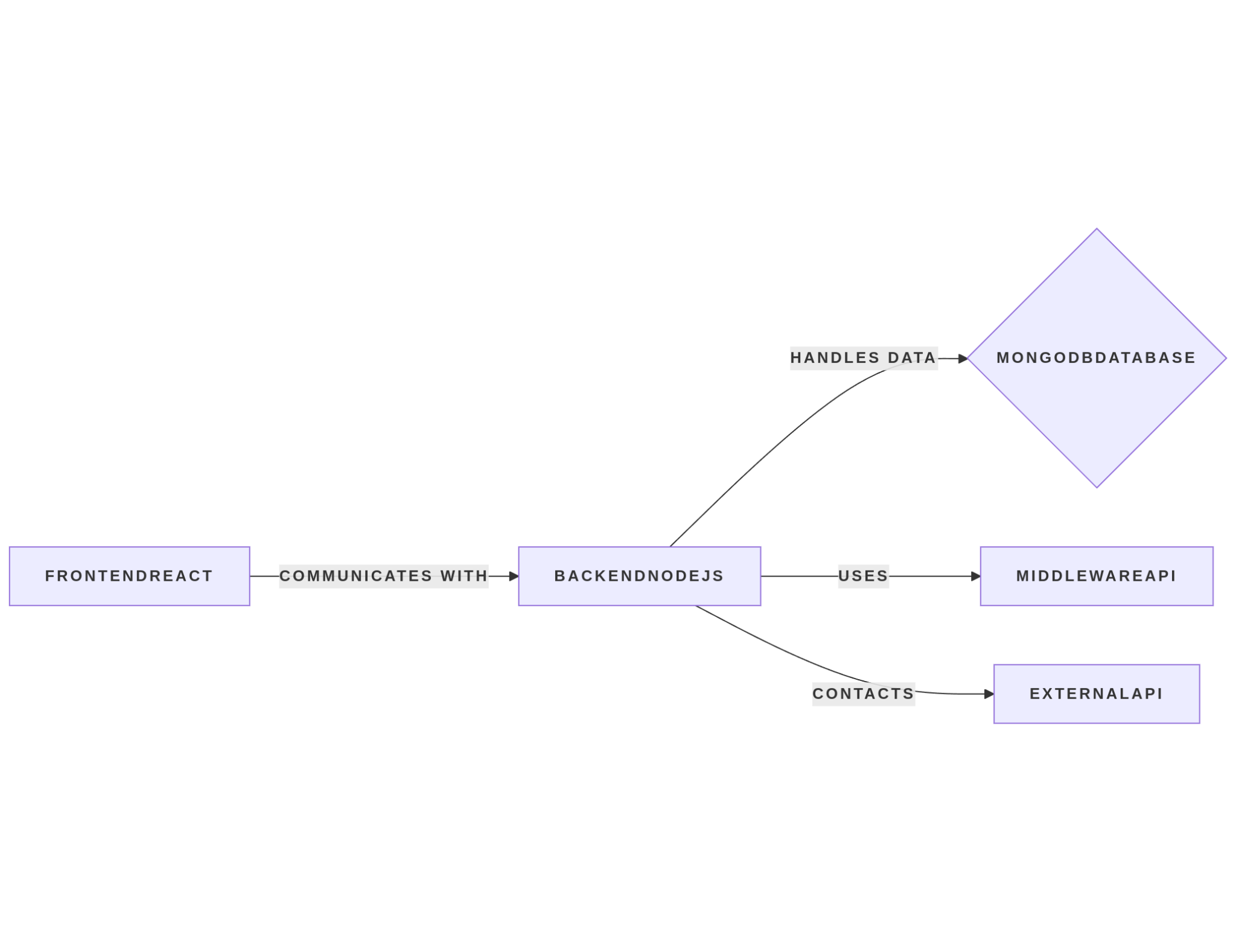
## **8. System Architecture**

### **Overview:** This application follows a **Client-Server Architecture** with a **React frontend** interacting with a **Node.js backend**, and data is stored in a **MongoDB** database. The application is designed with a modular approach for scalability and maintainability.

### **Key Components:**

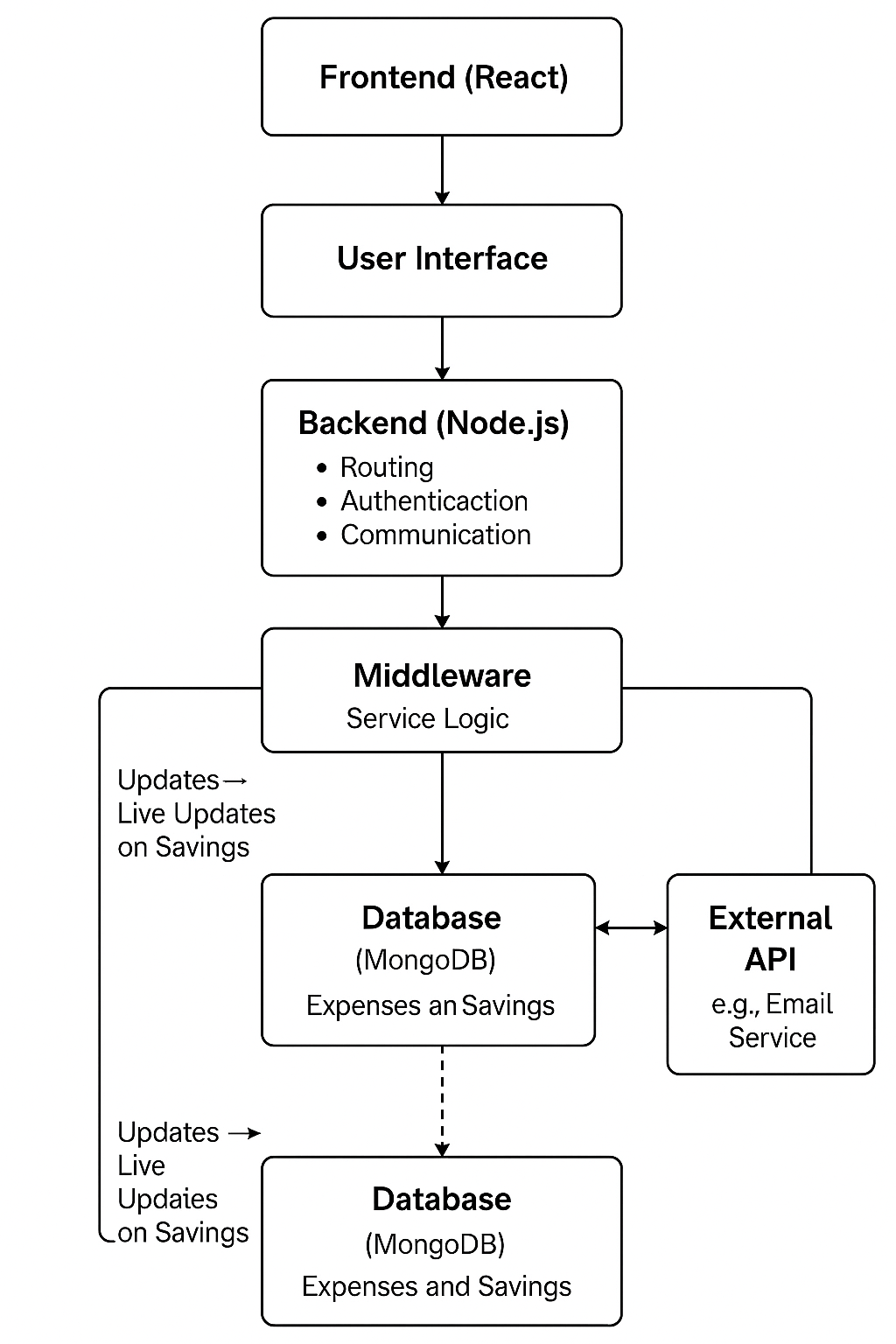
1. **Frontend (React)**: The user interface where customers interact with the application, including pages for user signup, login, expense tracking, income tracking, etc.
2. **Backend (Node.js)**: Handles API requests, authentication, and business logic. The backend is responsible for managing the database and providing the necessary responses.
3. **Database (MongoDB)**: Stores user data, expenses, income, budget, and other necessary information.
4. **Middleware**: API services and business logic such as handling expense editing, income tracking, and calculating balances.
5. **External API (Optional)**: If you are using third-party services (e.g., for email notifications, authentication, etc.), these can be part of the system architecture.

**Architecture Diagram**

****

***Fig: Architecture Diagram***

## **9. Flow Diagram:**



*Fig: Flow diagram*

## **10. Conclusion**

The **Smart Expense Tracker with Budget Optimization** integrates advanced AI-driven features to help users manage their finances effectively. It combines real-time expense tracking, budget recommendations, and anomaly detection to provide users with insights into their spending habits. The application leverages **React**, **Node.js**, and **MongoDB** for a full-stack solution, while **AI** ensures that the financial data is used to its maximum potential for budgeting and anomaly detection.