Personal Finance Tracker

(Technical Documentation)

Version: 1.0.0 **Last Updated:** June 22, 2025

1. Project Overview

The Personal Finance Tracker is a full-stack web application designed to help users manage their finances effectively. It allows users to track income and expenses, set monthly budgets, and view analytics on their spending habits. The application features a secure RESTful API backend built with Spring Boot and a responsive, modern frontend built with React.

1.1. Core Features

- **User Authentication:** Secure user registration and login using JWT (JSON Web Tokens).
- **Transaction Management:** CRUD operations for income and expense transactions.
- **Budgeting:** Ability for users to set monthly spending limits for various categories.
- **Recurring Expenses:** Manage and track recurring bills and subscriptions.
- **Financial Analytics:** A dashboard with charts visualizing monthly income vs. expenses and spending breakdowns.
- **Expense Forecasting:** A simple forecast of upcoming expenses based on recurring items.

1.2. Technology Stack

Layer	Technology & Libraries
Frontend	React 18, Vite, TypeScript, Tailwind CSS, React Router, Redux Toolkit, Axios, Recharts
Backend	Java 17, Spring Boot 3, Spring Security 6, Spring Data JPA, PostgreSQL, JWT, BCrypt
Database	PostgreSQL
DevOps	Docker, Nginx, Railway (Backend Hosting), Vercel (Frontend Hosting)
Tools	Git, GitHub, Postman, Maven

2. System Architecture

The application follows a decoupled, client-server architecture.

- **Frontend (Client):** A React Single-Page Application (SPA) that runs in the user's browser. It is responsible for the UI and user experience. It communicates with the backend via REST API calls.
- **Backend (Server):** A Spring Boot application that exposes a set of RESTful API endpoints. It handles all business logic, data persistence, and user authentication.
- **Database:** A PostgreSQL database stores all user data, including credentials, transactions, and budgets.

3. Backend Details

The backend is built with Spring Boot and organized into a modular structure.

3.1. Project Structure

The project is structured by features (modules) to promote separation of concerns.

```
src/main/java/com/tracker/finance/
├── core/                   # Cross-cutting concerns
│   ├── config/             # Spring configurations (Security, CORS, OpenAPI)
│   ├── exception/          # Global exception handling
│   └── security/           # JWT logic, UserDetailsService
└── modules/                # Feature modules
    ├── auth/               # Authentication and User Profile
    ├── transaction/        # Transaction management
    ├── budget/             # Budget management
    ├── recurringexpense/   # Recurring Expense management
    └── analytics/          # Data aggregation for charts and forecasts
```

3.2. Database Schema

The database consists of four main tables linked by user ID.

- **`users`**: Stores user credentials.
- **`transactions`**: Stores all income and expense records.
- **`budgets`**: Stores monthly budget limits per category.
- **`recurring_expenses`**: Stores recurring payment information.

3.3. API Endpoints

All endpoints are prefixed with `/api`. Authentication is required for all endpoints except `/auth/register` and `/auth/login`.

Endpoint	Method	Auth	Description
`/auth/register`	`POST`	✗	Registers a new user.
`/auth/login`	`POST`	✗	Authenticates a user and returns a JWT.
`/auth/me`	`GET`	✓	Gets the profile of the currently logged-in user.
`/auth/profile`	`PUT`	✓	Updates the current user's profile.
`/auth/profile`	`DELETE`	✓	Deletes the current user's account.
`/transactions`	`GET`	✓	Retrieves all transactions for the user.
`/transactions`	`POST`	✓	Adds a new transaction.
`/transactions/{id}`	`PUT`	✓	Updates an existing transaction.
`/transactions/{id}`	`DELETE`	✓	Deletes a transaction.
`/budgets`	`GET`	✓	Retrieves all budgets for the user.
`/budgets`	`POST`	✓	Adds a new budget for a specific month.
`/recurring-expenses`	`GET`	✓	Retrieves all recurring expenses for the user.
`/recurring-expenses`	`POST`	✓	Adds a new recurring expense.
`/recurring-expenses/{id}`	`DELETE`	✓	Deletes a recurring expense.
`/analytics/monthly-summary?month=YYYY-MM`	`GET`	✓	Gets a financial summary for the specified month.
`/analytics/forecast`	`GET`	✓	Gets a forecast of expenses for the next month.

3.4. Security

- **Authentication:** Handled by Spring Security using a custom JWT filter (`JwtAuthFilter`). A JWT is generated upon successful login and must be included as a `Bearer` token in the `Authorization` header for all protected requests.
- **Password Hashing:** Passwords are hashed using `BCryptPasswordEncoder` before being stored in the database.
- **CORS:** Cross-Origin Resource Sharing is enabled in `WebConfig` and is configurable via environment variables to only allow requests from the deployed frontend.
- **Exception Handling:** A `GlobalExceptionHandler` provides consistent, structured JSON error responses for different exception types.

4. Frontend Details

The frontend is a modern React SPA built with Vite and TypeScript.

4.1. Project Structure

The project follows a feature-sliced design.

```
src/
├── api/              # Central Axios instance configuration
├── components/       # Global, reusable UI components (e.g., Layout, Header)
├── features/         # Feature-based modules
│   ├── auth/         # Auth pages, components, and logic
│   ├── transactions/  # Transaction pages and components
│   └── ...
├── hooks/            # Custom hooks (e.g., useAppDispatch)
├── routes/           # Application routing configuration
├── store/            # Redux Toolkit store and slices
└── utils/            # Utility functions and validation schemas
```

4.2. State Management (Redux Toolkit)

Global application state is managed using Redux Toolkit.

- **Store:** The single source of truth, configured in `src/store/index.ts`.
- **Slices:** State logic is co-located by feature. Each slice (`authSlice`, `transactionSlice`, etc.) defines its initial state, reducers, and async thunks for interacting with the API.
- **Async Thunks:** Used for all asynchronous operations (API calls). They handle the pending, fulfilled, and rejected states of a request, allowing the UI to display loading indicators and error messages.

4.3. Routing

Client-side routing is handled by `react-router-dom`.

- **Public Routes:** Login and Register pages are accessible to everyone.
- **Private Routes:** All other pages (Dashboard, Transactions, etc.) are protected by the `PrivateRoute` component, which checks for a valid user session in the Redux store. If no user is authenticated, it redirects to the login page.

Future Enhancements

Feature	Description
AI Categorization	Use OpenAI to suggest categories
Notification Engine	Alert users on overspending or recurring bills
Export to Excel/PDF	Generate downloadable reports
Multi-Currency Support	Convert using real-time exchange rates
OAuth2 Google Login	Easier signup/login
Offline Mode (PWA)	Cache recent data with service workers