funds.io documentation

1. Project Overview

The Personal Finance Tracker (PFT) is a full-stack web application designed to help users manage their personal finances by tracking accounts, transactions, financial goals, and notifications. It includes an admin panel for user management and supports real-time updates via WebSockets. The application emphasizes user experience with features like toast notifications, loading spinners, and secure authentication.

Purpose

* Enable users to create and manage financial accounts, log transactions, and set savings goals.
* Provide real-time notifications for budget violations and goal progress.
* Allow administrators to manage user accounts.
* Ensure a seamless user experience with responsive feedback and secure authentication.

Key Features

* User Authentication: Register, login, and manage profiles (email, name, password).
* Account Management: Create and track financial accounts with budgets.
* Transaction Tracking: Log income and expense transactions, categorized for analysis.
* Financial Goals: Set savings goals with deadlines and spending constraints.
* Notifications: Real-time alerts for budget overruns and goal milestones.
* Admin Panel: View and delete user accounts (admin-only).
* Real-Time Updates: WebSocket-based notifications for transactions and alerts.

2. System Architecture

Technology Stack

* Frontend:
  + React: JavaScript library for building the user interface.
  + TypeScript: Adds static typing for better code reliability.
  + React Router: Handles client-side routing.
  + Axios: Manages HTTP requests with a global 401 interceptor for token expiration.
  + react-toastify: Displays toast notifications for success/error messages.
  + react-spinners: Shows loading indicators during API calls.
  + Chart.js / react-chartjs-2: Renders financial charts (Pie, Bar, Line).
  + Socket.IO Client: Receives real-time updates from the backend.
* Backend:
  + Node.js / Express: Server framework for handling API requests.
  + MongoDB / Mongoose: NoSQL database for storing user, account, transaction, goal, and notification data.
  + Socket.IO: Enables real-time notifications.
  + JWT (JSON Web Tokens): Secures authentication with role-based access (user/admin).
  + bcryptjs: Hashes passwords for secure storage.
* Environment:
  + dotenv: Manages environment variables (e.g., MONGO\_URI, JWT\_SECRET).
  + CORS: Enables cross-origin requests between frontend and backend.

File Structure

Backend (backend/)

backend/

├── models/

│ ├── userModel.js

│ ├── accountModel.js

│ ├── transactionModel.js

│ ├── goalModel.js

│ ├── notificationModel.js

├── routes/

│ ├── userRoutes.js

│ ├── accountRoutes.js

│ ├── transactionRoutes.js

│ ├── goalRoutes.js

│ ├── notificationRoutes.js

├── middleware/

│ ├── authMiddleware.js

├── socket.js

├── server.js

├── package.json

Frontend (frontend/)

frontend/

├── src/

│ ├── api/

│ │ ├── auth.ts

│ │ ├── finance.ts

│ │ ├── goals.ts

│ │ ├── notifications.ts

│ │ ├── users.ts

│ │ ├── axios.ts

│ ├── components/

│ │ ├── Header.tsx

│ │ ├── LoginForm.tsx

│ │ ├── RegisterForm.tsx

│ │ ├── ToastContainer.tsx

│ ├── context/

│ │ ├── AuthContext.tsx

│ ├── pages/

│ │ ├── Home.tsx

│ │ ├── Login.tsx

│ │ ├── Register.tsx

│ │ ├── Tracker.tsx

│ │ ├── Summary.tsx

│ │ ├── Dashboard.tsx

│ │ ├── Notifications.tsx

│ │ ├── UserData.tsx

│ │ ├── Users.tsx

│ │ ├── About.tsx

│ ├── index.css

│ ├── main.tsx

│ ├── App.tsx

├── package.json

Data Models

* User:
  + Fields: \_id, email (unique), name, password (hashed), role (user or admin), createdAt.
  + Pre-save hook: Hashes password using bcryptjs.
* Account:
  + Fields: \_id, userId, name, budget, createdAt.
* Transaction:
  + Fields: \_id, userId, accountId, type (income or expense), amount, category, createdAt.
* Goal:
  + Fields: \_id, userId, name, targetAmount, deadline, accountId (optional), constraints (array of { type, value, accountId }), createdAt.
* Notification:
  + Fields: \_id, userId, message, type, read, relatedId (optional), createdAt.

3. Setup Instructions

Prerequisites

* Node.js: Version 18.x or higher.
* MongoDB: Local or cloud instance (e.g., MongoDB Atlas).
* npm: Package manager for installing dependencies.

Backend Setup

* Navigate to Backend Directory:

bash

cd backend

* Install Dependencies:

bash

npm install

* Configure Environment Variables:
  + Create a .env file in backend/:
  + MONGO\_URI=mongodb://localhost:27017/funds.io
  + JWT\_SECRET=your\_jwt\_secret

PORT=5000

* + Replace JWT\_SECRET with a secure key.
* Start the Backend:

bash

npm run dev

* + Runs the server on http://localhost:5000 using nodemon.

Frontend Setup

* Navigate to Frontend Directory:

bash

cd frontend

* Install Dependencies:

bash

npm install

* Start the Frontend:

bash

npm run dev

* + Runs the app on http://localhost:5173 (or the port shown in the terminal).

Database Setup

* MongoDB:
  + Ensure MongoDB is running locally or use a cloud instance.
  + Create a database named funds.io (or as specified in MONGO\_URI).
* Set Admin User:
  + After registering a user (e.g., via /register), set their role to admin in MongoDB:

javascript

db.users.updateOne({ email: "test@example.com" }, { $set: { role: "admin" } });

4. Features and Implementation

Authentication

* Endpoints:
  + POST /api/register: Creates a user and returns a JWT.
  + POST /api/login: Authenticates a user and returns a JWT.
  + GET /api/users/me: Retrieves current user’s profile.
  + PUT /api/users/me: Updates current user’s email, name, or password.
  + DELETE /api/users/me: Deletes the current user and their data.
* Frontend:
  + AuthContext.tsx: Manages token and role state, persists token in localStorage, and restores role on page reload by decoding the JWT.
  + Axios interceptor (axios.ts): Handles 401 errors by triggering logout and redirecting to /login.
  + Pages: Login.tsx, Register.tsx, UserData.tsx.

Account and Transaction Management

* Endpoints:
  + GET/POST /api/accounts: List or create accounts.
  + GET/POST /api/transactions: List or create transactions.
* Frontend:
  + Tracker.tsx: Form to create accounts and transactions, displays transaction list.
  + Summary.tsx: Visualizes financial data with Pie, Bar, and Line charts.
  + Dashboard.tsx: Combines account, transaction, goal, and notification summaries.

Financial Goals

* Endpoints:
  + GET/POST /api/goals: List or create goals.
  + GET /api/goals/:id/progress: Calculates goal progress.
* Frontend:
  + Tracker.tsx: Form to create goals with constraints (e.g., min/max spending).
  + Summary.tsx, Dashboard.tsx: Display goal progress.

Notifications

* Endpoints:
  + GET /api/notifications: List notifications.
  + PUT /api/notifications/:id/read: Mark a notification as read.
* Frontend:
  + Notifications.tsx: Displays notifications with “Mark as Read” buttons.
  + Socket.IO: Listens for newNotification events to update in real-time.

Admin Panel

* Endpoints:
  + GET /api/users: Lists all users (admin-only).
  + DELETE /api/users/:id: Deletes a user and their data (admin-only, prevents self-deletion).
* Frontend:
  + Users.tsx: Lists users with delete buttons, accessible only to admins.
  + Header.tsx: Shows “Users” link for admins only.

UX Enhancements

* Toast Notifications (react-toastify):
  + Success messages: e.g., “Profile updated successfully,” “User deleted successfully.”
  + Error messages: e.g., “Failed to load profile,” “Invalid credentials.”
* Loading Spinners (react-spinners):
  + Shown during API calls in Tracker, Summary, Dashboard, Notifications, UserData, Users, Login, Register.
* Real-Time Updates:
  + Socket.IO events (newTransaction, newNotification) ensure immediate UI updates.

5. Testing Instructions

* Environment:
  + Ensure backend (http://localhost:5000) and frontend (http://localhost:5173) are running.
  + MongoDB should be accessible with the correct MONGO\_URI.
* Test Cases:
  + Authentication:
    - Register a user, login, and verify token in localStorage.
    - Update profile in /personal, confirm success toast.
    - Delete own account in /personal, confirm logout and data deletion.
  + Account/Transaction:
    - Create an account and transaction in /tracker, verify in /summary charts.
    - Check Socket.IO updates for new transactions.
  + Goals:
    - Create a goal in /tracker, verify progress in /summary and /dashboard.
  + Notifications:
    - Mark notifications as read in /notifications, confirm toast and Socket.IO updates.
  + Admin Panel:
    - Login as admin (set role: 'admin' in MongoDB).
    - Verify /users lists users and allows deletion (except self).
    - Reload page, confirm admin panel persists.
  + Token Expiration:
    - Set expiresIn: '10s' in userRoutes.js, perform an action after expiry, verify redirect to /login with toast.
* Tools:
  + MongoDB Compass: Inspect database.
  + Browser DevTools: Check localStorage and network requests.
  + Postman: Test API endpoints.

6. Known Issues

* Token Expiration UX: Users may be abruptly logged out after token expiry without prior warning.
* Accessibility: Basic styling lacks full accessibility (e.g., ARIA attributes, keyboard navigation).
* Error Handling: Some edge cases (e.g., network failures) may show generic error messages.

7. Future Improvements

* Enhanced Validation:
  + Add client-side form validation (e.g., regex for email, minimum password length).
  + Display specific error messages (e.g., “Email already in use”).
* Advanced Features:
  + Export/import financial data as CSV.
  + Budget analytics (e.g., spending trends, category breakdowns).
  + Recurring transactions or reminders for goals.
* UX Polish:
  + Add a confirmation modal for critical actions (e.g., account deletion).
  + Implement a global loading overlay for better UX.
  + Toast notifications and loading spinners for better feedback.
  + Enhance accessibility with ARIA labels and focus management.
* Security:
  + Implement refresh tokens for longer sessions.
  + Add rate limiting to prevent brute-force attacks.
* Testing:
  + Add unit tests (e.g., Jest for frontend, Mocha for backend).
  + Implement end-to-end tests with Cypress.

Github

<https://github.com/aynkv/funds.io>