Technical Task

Backend Development with Express, TypeScript, and PostgreSQL

Objective

Create a backend API for managing users and their wallets. The API should be built using the following technologies:

- **Framework:** Express.js
- **Language:** TypeScript
- **Database:** PostgreSQL

The task involves implementing authentication (sign-in/sign-out) and CRUD operations for managing wallets.

Requirements

1. **Database Schema**

Users Table:

- *id*: Primary Key (auto-increment or UUID)
- *email*: String (unique, required)
- *password*: String (hashed, required)

Wallets Table:

- *id*: Primary Key (auto-increment or UUID)
- *user_id*: Foreign Key (references the Users table)
- *tag*: String (optional, for labeling wallets)
- *chain*: String (required, identifies the blockchain, e.g., Ethereum, Bitcoin)
- *address*: String (required, unique)

You can add additional tables such as Chains if needed to store metadata for blockchain chains.

2. **API Endpoints**

Authentication

POST */signin*: Sign in a user.

- Input: { email: string, password: string }
- Output: Success or error response (e.g., JWT token for authentication).

POST */signout*: Sign out a user.

- Input: Authorization token (JWT).
- Output: Success or error response.

Wallets CRUD

GET */wallets*: Retrieve all wallets for the authenticated user.

- Input: Authorization token (JWT).
- Output: List of wallets.

POST */wallets*: Create a new wallet for the authenticated user.

- Input: { tag?: string, chain: string, address: string }
- Output: Success or error response.

GET */wallets/:id*: Retrieve details of a specific wallet by id.

- Input: Wallet ID and authorization token.
- Output: Wallet details.

PUT */wallets/:id*: Update a wallet for the authenticated user.

- Input: { tag?: string, chain: string, address: string }
- Output: Success or error response.

DELETE /wallets/:id: Delete a wallet by id.

- Input: Wallet ID and authorization token.
- Output: Success or error response.

3. **Authentication & Security**

- Use JWT (JSON Web Token) for session management and user authentication.
- Passwords should be securely hashed using hashing library.
- Ensure that only authenticated users can manage their own wallets.

4. **Error Handling**

- Implement proper error handling and validation for all API inputs (e.g., invalid email format, missing required fields, etc.).
- Return appropriate HTTP status codes (e.g., 200 for success, 400 for bad requests, 401 for unauthorized).

5. **Documentation**
   - Provide clear and concise documentation for the API endpoints (can be as simple as a README file).
   - Include instructions on how to set up the project locally (e.g., environment variables, database migrations).

Deliverables

1. A GitHub repository with the full backend codebase.
2. Documentation for setting up and running the API.
3. Postman or equivalent collection for testing the API. (optional)

Consider using AI tools to increase productivity and streamline development