Project Design Phase
Solution Architecture

Date	16 April 2025
Team ID	SWTID1743315733
Project Name	Personal Finance Tracker
Maximum Marks	4 Marks

Solution Architecture:

The solution architecture for the Personal Finance Tracker is designed to bridge the gap between the user's need for simplified financial management and a robust, scalable technology solution. It aims to provide an intuitive, secure, and reliable platform for users to track income, expenses, and budgets effectively.

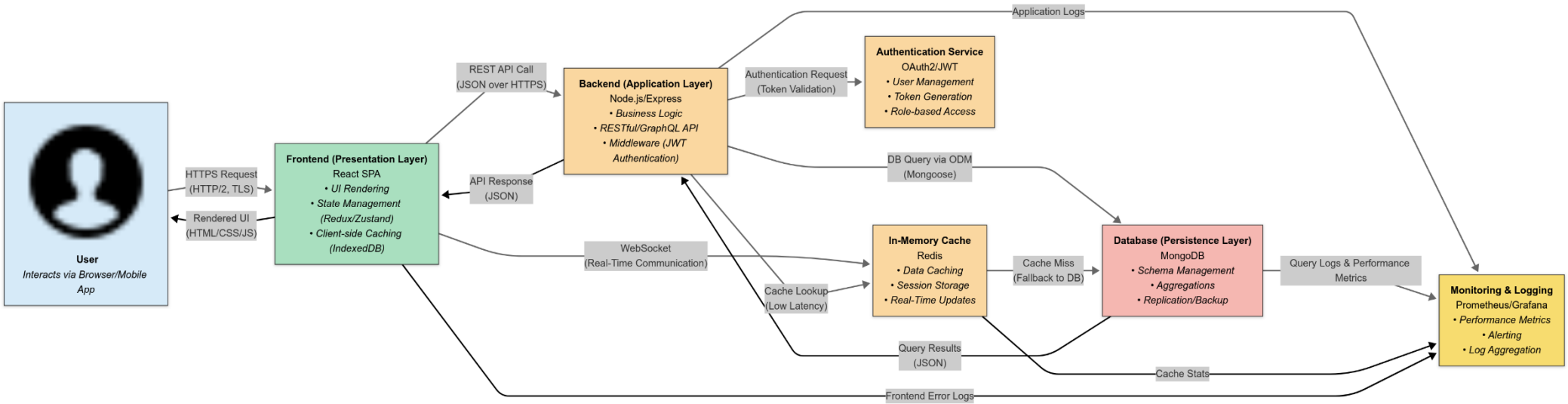
The architecture follows a standard **3-Tier Web Application** model, separating concerns into distinct layers:

1. **Presentation Tier (Frontend):**
    - **Description:** This tier is responsible for the User Interface (UI) and User Experience (UX). It's what the user directly interacts with in their web browser.
    - **Components:** A Single Page Application (SPA) built using React.js. It includes components for user registration/login, data entry forms (income, expense, budget), dashboards, transaction lists, charts, and profile management.
    - **Behavior:** Renders the UI, captures user input, performs client-side validation, communicates with the Backend API via RESTful calls (HTTPS) to fetch and send data, and displays information and visualizations (e.g., charts using Chart.js) received from the backend.
    - **Technology:** React.js, HTML5, CSS3, JavaScript (ES6+), Axios (for API calls), Chart.js.
    - **Problem Addressed:** Provides an easy-to-use, interactive, and visually appealing interface, solving the problem of complex or tedious manual tracking methods and enabling clear visualization of financial data.
2. **Logic Tier (Backend API):**
    - **Description:** This tier contains the core business logic, handles data processing, manages user authentication, and serves as the intermediary between the frontend and the database.
    - **Components:** A RESTful API built using Node.js and the Express.js framework. It includes endpoints for user authentication (registration, login using JWT), CRUD (Create, Read, Update, Delete) operations for income, expenses, and budgets, data aggregation for

dashboard/reports, and user profile updates. Mongoose ODM is used for modeling application data and interacting with the database.

- **Behavior:** Receives requests from the frontend, validates data, performs business logic (e.g., calculating balances, checking budget limits), authenticates/authorizes users using JWT, interacts with the database to store or retrieve data, and sends responses (data payloads or status messages) back to the frontend.
- **Technology:** Node.js, Express.js, MongoDB, Mongoose, JSON Web Tokens (JWT), bcrypt (for password hashing).
- **Problem Addressed:** Enforces business rules, ensures data integrity, provides secure access control, and centralizes the application's core functionality.

3. **Data Tier (Database):**
    - **Description:** This tier is responsible for the persistent storage of all application data.
    - **Components:** A NoSQL database, specifically MongoDB. It stores collections for users (including hashed passwords), income transactions, expense transactions, and budget configurations, all linked to specific users.
    - **Behavior:** Stores data provided by the backend, retrieves data based on queries from the backend, ensures data persistence, and supports indexing for efficient data retrieval.
    - **Technology:** MongoDB (potentially hosted on MongoDB Atlas for cloud deployment).
    - **Problem Addressed:** Reliably stores user financial data, allowing users to access their historical information and enabling the application to perform calculations and generate reports based on persisted data.+


User
Interacts via Browser/Mobile App
HTTPS Request (HTTP/2, TLS)
Rendered UI (HTML/CSS/JS)
Frontend (Presentation Layer)
React SPA
• UI Rendering
• State Management (Redux/Zustand)
• Client-side Caching (IndexedDB)
REST API Call (JSON over HTTPS)
API Response (JSON)
Backend (Application Layer)
Node.js/Express
• Business Logic
• RESTful/GraphQL API
• Middleware (JWT Authentication)
Authentication Request (Token Validation)
Authentication Service
OAuth2/JWT
• User Management
• Token Generation
• Role-based Access
Application Logs
DB Query via ODM (Mongoose)
WebSocket (Real-Time Communication)
Cache Lookup (Low Latency)
In-Memory Cache
Redis
• Data Caching
• Session Storage
• Real-Time Updates
Cache Miss (Fallback to DB)
Database (Persistence Layer)
MongoDB
• Schema Management
• Aggregations
• Replication/Backup
Query Logs & Performance Metrics
Query Results (JSON)
Cache Stats
Frontend Error Logs
Monitoring & Logging
Prometheus/Grafana
• Performance Metrics
• Alerting
• Log Aggregation