**1. Personal Finance Management App**

**Description:** A web application designed to help users track and manage their personal finances. The app includes features for budgeting, expense tracking, and financial goal setting.

**Key Features:**

* **Backend**: Developed using Django for user management, financial data storage, and budget calculations. Implemented RESTful APIs with Django REST Framework for frontend-backend communication.
* **Frontend**: Built with React.js to provide a dynamic and responsive user interface. Used Redux for state management and React Hooks for managing component states and side effects.
* **Database**: Utilized PostgreSQL for structured data storage and MongoDB for storing unstructured financial data and user preferences.
* **Deployment**: Deployed the application on AWS using EC2 for the server and S3 for storing user-uploaded documents.
* **CI/CD**: Set up a CI/CD pipeline with GitHub Actions for automated testing and deployment.

**Technologies:** Python, Django, React.js, Redux, PostgreSQL, MongoDB, AWS (EC2, S3), GitHub Actions

**Achievements:**

* Implemented real-time financial tracking and alerts using WebSockets.
* Achieved 98% code coverage with unit and integration tests using PyTest and Unittest.
* Integrated payment gateways for financial transactions.

==================================================================================================

**Costing** :

* **Development Tools:** Free
* **Hosting (AWS EC2 and S3):** Free tier available for 12 months, costs depend on usage after that.
* **CI/CD (GitHub Actions):** Free for public repositories, limited free usage for private repositories.
* **Payment Gateways:** Free to integrate, transaction fees apply.
* **Optional External Services:** Varies based on service.

==================================================================================================

**Requirements**

**Functional Requirements:**

1. **User Authentication:**
   * Sign up
   * Login
   * Password Reset
   * Profile Management
2. **Expense Tracking:**
   * Add/Edit/Delete expenses
   * Categorize expenses (e.g., food, transportation, entertainment)
   * View expense history
3. **Income Management:**
   * Add/Edit/Delete income sources
   * View income history
4. **Budget Management:**
   * Create/Edit/Delete budgets
   * Set budget limits by category
   * Track budget usage
5. **Financial Goal Setting:**
   * Set financial goals (e.g., saving for a vacation)
   * Track progress towards goals
6. **Real-time Financial Tracking and Alerts:**
   * Real-time updates on expense and income entries
   * Alerts for budget limits and financial goals
7. **Payment Gateway Integration:**
   * Integrate with payment gateways like Stripe or PayPal for financial transactions
   * Secure handling of payment data

**Non-Functional Requirements:**

1. **Security:**
   * HTTPS for all communications
   * Secure storage of user data (encrypted passwords, secure payment data handling)
   * Role-based access control
2. **Performance:**
   * Fast response times for API calls
   * Efficient database queries
3. **Scalability:**
   * Design to handle increasing number of users and data
   * Use scalable cloud services (AWS)
4. **Usability:**
   * User-friendly and intuitive UI/UX
   * Mobile-friendly responsive design
5. **Maintainability:**
   * Well-documented code
   * Modular architecture for easy updates and maintenance

**User Stories :**

1. User Authentication:

* Sign Up: As a user, I want to create a new account so that I can access the app.
* Login: As a user, I want to log in to my account so that I can access my personal finance data.
* Password Reset: As a user, I want to reset my password if I forget it so that I can regain access to my account.
* Profile Management: As a user, I want to update my profile information so that my account is up-to-date.

1. Expense Tracking:

* Add Expense: As a user, I want to add a new expense so that I can keep track of my spending.
* Edit Expense: As a user, I want to edit an existing expense so that I can correct any mistakes.
* Delete Expense: As a user, I want to delete an expense so that I can remove incorrect entries.
* Categorize Expense: As a user, I want to categorize my expenses so that I can see how much I spend in each category.
* View Expense History: As a user, I want to view my past expenses so that I can analyze my spending habits.

1. Income Management:

* Add Income Source: As a user, I want to add a new income source so that I can track my earnings.
* Edit Income Source: As a user, I want to edit an existing income source so that I can correct any mistakes.
* Delete Income Source: As a user, I want to delete an income source so that I can remove incorrect entries.
* View Income History: As a user, I want to view my past income so that I can analyze my earnings.

1. Budget Management:

* Create Budget: As a user, I want to create a new budget so that I can plan my spending.
* Edit Budget: As a user, I want to edit an existing budget so that I can adjust my spending plan.
* Delete Budget: As a user, I want to delete a budget so that I can remove it if it's no longer relevant.
* Set Budget Limits: As a user, I want to set limits for my budget categories so that I can control my spending.
* Track Budget Usage: As a user, I want to see how much of my budget I have used so that I can manage my spending.

1. Financial Goal Setting:

* Set Financial Goal: As a user, I want to set a financial goal so that I can save for future expenses.
* Track Progress: As a user, I want to track my progress towards my financial goals so that I can stay motivated.

1. Real-time Financial Tracking and Alerts:

* Real-time Updates: As a user, I want to receive real-time updates on my expense and income entries so that I can stay informed.
* Budget Alerts: As a user, I want to receive alerts when I am close to my budget limits so that I can avoid overspending.
* Goal Alerts: As a user, I want to receive alerts when I am close to achieving my financial goals so that I can stay motivated.

1. Payment Gateway Integration:

* Secure Transactions: As a user, I want to make secure financial transactions so that my payment data is protected.
* Transaction History: As a user, I want to view my past transactions so that I can keep track of my payments.

**Detailed High-Level Design**

1. Architecture Overview
   * Frontend: React.js + Redux
   * Backend: Django + Django REST Framework
   * Databases: MySQL for all structured data (user data, financial records)
   * Storage: AWS S3 for user-uploaded documents
2. Backend Design
   * Services:
     + Authentication Service: Handles user sign-up, login, and profile management.
     + Expense Management Service: Manages expense CRUD operations and categorization.
     + Income Management Service: Manages income source CRUD operations.
     + Budget Management Service: Handles budget creation, updates, and tracking.
     + Financial Goal Service: Manages financial goals and progress tracking.
     + Notification Service: Sends real-time alerts using Django Channels.
     + Payment Integration Service: Integrates with payment gateways (Stripe/PayPal).
     + Document Management Service: Manages storage and retrieval of user-uploaded documents using AWS S3.
3. Frontend Design
   * Components:
     + Authentication Module: UI for sign-up, login, and profile management.
     + Dashboard Module: Displays an overview of financial status, expenses, income, and budgets.
     + Expense Module: UI for adding, editing, deleting, and viewing expenses.
     + Income Module: UI for adding, editing, deleting, and viewing income sources.
     + Budget Module: UI for creating, updating, deleting, and tracking budgets.
     + Goals Module: UI for setting and tracking financial goals.
     + Notification Module: Displays real-time alerts.
4. Database Design (PostgreSQL)
   * User Table: Stores user information (id, username, email, password\_hash, etc.)
   * Expense Table: Stores expense records (id, user\_id, category, amount, date, description, etc.)
   * Income Table: Stores income records (id, user\_id, source, amount, date, description, etc.)
   * Budget Table: Stores budget information (id, user\_id, category, limit, period, etc.)
   * Goal Table: Stores financial goals (id, user\_id, target\_amount, saved\_amount, description, deadline, etc.)
5. Storage Design (AWS S3)
   * Store all user-uploaded documents.
   * Use S3 buckets with appropriate access controls and encryption for security.
6. Integration and Communication
   * API Communication: The frontend communicates with the backend via RESTful APIs.
   * WebSockets: Real-time updates are pushed from the backend to the frontend using WebSockets.
   * Payment Gateways: Backend integrates with external payment gateways for secure transactions.
7. Deployment Architecture
   * AWS Services:
     + EC2 Instances: Hosts the Django application.
     + S3 Buckets: Stores user-uploaded documents.
     + RDS: Manages PostgreSQL database.
     + CloudFront: Distributes the React application for fast access.

**Sequence Diagram: User Login**

Actors:

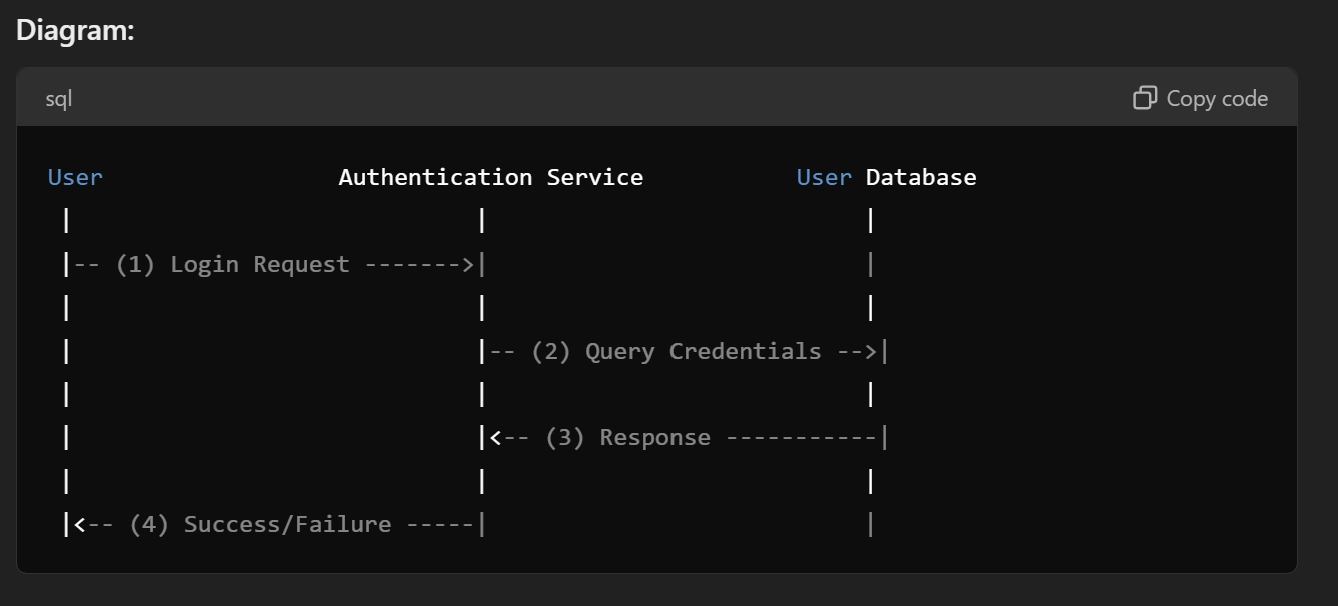
* User: The person trying to log in.
* Authentication Service: The backend service handling authentication.
* User Database: The database storing user credentials.

Steps:

1. User sends login credentials to Authentication Service.
2. Authentication Service queries User Database to verify credentials.
3. User Database responds with user information or failure status.
4. Authentication Service sends a success or failure response back to User.

Explanation:

1. Login Request: User sends login credentials (username and password) to the Authentication Service.
2. Query Credentials: Authentication Service queries the User Database to validate the credentials.
3. Response: User Database returns user information or a failure status.
4. Success/Failure: Authentication Service sends a success or failure message back to the User.

****