Project Design Phase
Solution Architecture

Date	15 April 2025
Team ID	SWTID1743344524
Project Name	Personal Expense Tracker App
Maximum Marks	4 Marks

Smart Spend is a full-stack web application designed to help users track their income and expenses in real-time. The architecture ensures scalability, security, and a seamless user experience by integrating various components that handle user interactions, data processing, and storage.

Architecture Diagram

While I can't display images directly, here's a textual representation of the architecture:

csharp

CopyEdit

[Client (Browser)]

|

v

[Frontend (React.js)]

|

v

[Backend API (Node.js/Express)]

|

v

[Database (MongoDB)]

Components:

- Client (Browser): The user's interface to interact with the application.
- Frontend (React.js): Handles the user interface, rendering components, and managing client-side routing.
- Backend API (Node.js/Express): Processes client requests, handles business logic, and communicates with the database.
- Database (MongoDB): Stores user data, including income, expenses, categories, and user profiles.

Component Details

- Frontend (React.js):
  - Provides a responsive and intuitive user interface.
  - Implements features like dashboards, charts, and forms for data entry.
  - Communicates with the backend via RESTful APIs.GitHub
- Backend API (Node.js/Express):
  - Handles authentication and authorization using JWT tokens.
  - Validates and processes incoming data from the frontend.
  - Implements business logic for categorizing expenses and generating summaries.
  - Interacts with the MongoDB database to perform CRUD operations.
- Database (MongoDB):
  - Stores collections for users, transactions, categories, and settings.
  - Ensures data integrity and supports indexing for efficient queries.

Data Flow

1. User Interaction:
   - The user accesses the application via a web browser.
   - Interacts with the UI to input income or expense data.
2. Frontend Processing:
   - Captures user input and sends it to the backend API.
   - Displays real-time feedback and updates the UI accordingly.
3. Backend Processing:
   - Receives data from the frontend.
   - Validates and processes the data.
   - Performs necessary computations, such as calculating totals or generating reports.
   - Interacts with the database to store or retrieve data.
4. Database Operations:
   - Stores new transactions or updates existing records.
   - Retrieves data for generating summaries or reports.
5. Response to Frontend:

    - Sends processed data back to the frontend.
    - Frontend updates the UI to reflect the latest data.

Security Considerations

- Authentication:
    - Users must register and log in to access the application.
    - JWT tokens are used to manage sessions securely.
- Data Protection:
    - Sensitive data is encrypted in transit using HTTPS.
    - Input validation and sanitization are implemented to prevent common vulnerabilities.

Scalability and Future Enhancements

- Modular Architecture:
    - The separation of concerns allows for independent scaling of components.
- Potential Enhancements:
    - Integration with third-party financial APIs for automatic transaction imports.
    - Mobile application development using React Native for cross-platform support.
    - Implementation of machine learning algorithms for predictive analytics and budgeting suggestions.