**Backend Documentation for Smart Spending System**

This document provides a comprehensive explanation of the backend architecture, functionality, and implementation of the **Smart Spending** system. The backend is responsible for processing user requests, storing data, managing AI models, and ensuring secure and efficient operation of the system.

**1. Overview**

The backend is the server-side component of the **Smart Spending** system. It performs the following tasks:

1. Handles user authentication and authorization.
2. Processes and stores data for spending analysis, budgeting, and savings.
3. Runs AI algorithms for personalized recommendations.
4. Communicates with external APIs (e.g., Mpesa, banking systems).
5. Provides a RESTful API for frontend interactions.

**2. Functional Requirements**

**1. User Management**

* **Signup**: Create user profiles with unique credentials.
* **Login**: Authenticate users and generate secure tokens (e.g., JWT).
* **Profile Management**: Allow users to update personal information.

**2. Spending Analysis**

* Record expenses with details like:
  + Product name.
  + Payment method (Mpesa or bank).
  + Date and time.
  + Amount.
* Categorize expenses into “necessary” and “unnecessary” using an AI model.
* Generate spending reports based on recorded transactions.

**3. Budgeting and Savings**

* Store user-defined budgets and savings goals.
* Track progress and send reminders for savings deadlines.
* Provide alerts when expenses exceed set budgets.

**4. Income Tracking**

* Record user income from multiple sources.
* Analyze income patterns and identify inconsistencies.

**5. Investment Recommendations**

* Store and process data on user preferences and financial profiles.
* Use AI to suggest investment options based on:
  + User’s risk tolerance.
  + Income level.
  + Spending habits.

**6. Integration with Mpesa and Banks**

* Facilitate payment logging through Mpesa or bank APIs.
* Record transactions automatically in the system for analysis.

**7. AI Model Management**

* Process data for training and updating the AI model.
* Generate insights and recommendations for:
  + Budgeting.
  + Expense reduction.
  + Investment opportunities.

**3. Architecture**

**1. API Layer**

* RESTful API endpoints to communicate with the frontend.
* Endpoints categorized into:
  + User Management (signup, login, profile updates).
  + Transaction Management (log expenses, view reports).
  + Budgeting and Savings (set goals, track progress).
  + AI Insights (fetch recommendations, spending analysis).

**2. Database**

* **Type**: Relational database (e.g., PostgreSQL) or NoSQL database (e.g., MongoDB).
* **Tables/Collections**:
  + **Users**: Stores user details (name, email, password hash, etc.).
  + **Transactions**: Logs expense and income records.
  + **Budgets**: Stores user-defined budgets.
  + **Savings Goals**: Tracks savings progress.
  + **AI Insights**: Stores analyzed data and recommendations.

**3. AI Integration**

* **Model Training**: Uses historical data to train a machine learning model for:
  + Expense categorization.
  + Budget predictions.
  + Investment suggestions.
* **Real-time Processing**: Applies the trained model to live user data for insights.

**4. External API Integration**

* **Mpesa API**:
  + Processes payments.
  + Logs transactions automatically.
* **Bank APIs**:
  + Retrieves transaction history.
  + Facilitates secure fund transfers.
* **SMS/Email APIs**:
  + Sends alerts and reminders to users.

**4. Core Components and Implementation**

**1. User Authentication and Authorization**

* Use **JWT (JSON Web Tokens)** for secure authentication.
* Passwords stored using secure hashing (e.g., bcrypt).
* Role-based access control (RBAC) for admin and regular users.

**2. Transaction Management**

* Expense Logging:
  + Endpoint: /api/transactions/log
  + Accepts: Product name, amount, payment method, date.
  + Saves transaction to the database and triggers AI processing.
* Income Logging:
  + Endpoint: /api/income/log
  + Accepts: Source name, amount, date.
  + Updates user’s financial profile.

**3. Spending Analysis**

* Categorization:
  + Endpoint: /api/transactions/analyze
  + Uses the AI model to classify transactions into necessary and unnecessary.
* Insights:
  + Endpoint: /api/insights/spending
  + Returns visual data (e.g., JSON for frontend charts).

**4. Budgeting and Savings**

* Budget Management:
  + Endpoint: /api/budgets/set
  + Accepts income and category allocations.
  + Stores budget in the database and calculates savings goals.
* Savings Tracking:
  + Endpoint: /api/savings/progress
  + Returns current savings status and reminders.

**5. Investment Suggestions**

* Recommendations:
  + Endpoint: /api/investments/recommend
  + Uses AI to suggest options and provide educational material.

**6. AI Model Management**

* Training:
  + Processes historical transaction data for improved accuracy.
* Deployment:
  + Hosted on the backend server or a cloud-based ML service.
* Real-time Processing:
  + Processes user data dynamically for instant recommendations.

**7. Data Security**

* Encryption:
  + All sensitive data (e.g., transactions, personal details) is encrypted.
* Secure Communication:
  + Use HTTPS for all API interactions.
* Compliance:
  + Adhere to GDPR, CCPA, or applicable local data protection laws.

**5. Backend Workflow**

**1. User Signup/Login**

1. User submits credentials.
2. Backend validates and stores the data securely.
3. A token is generated for session management.

**2. Log Expense/Income**

1. User submits transaction data.
2. Backend stores the data and triggers AI analysis.
3. Processed insights are stored and returned to the frontend.

**3. Generate Budget and Savings Plan**

1. User sets income and savings goals.
2. Backend calculates budget allocations.
3. Savings progress is tracked and reminders are scheduled.

**4. AI Recommendations**

1. User requests spending or investment insights.
2. Backend retrieves user data and applies AI processing.
3. Recommendations are returned in JSON format.

**5. Payment Logging**

1. User selects a payment method (Mpesa or bank).
2. Backend interacts with the respective API.
3. Transaction details are stored and analyzed.

**6. Technology Stack**

**Backend Framework**

* **Django** (Python) or **Node.js** (JavaScript) for API development.

**Database**

* Relational: PostgreSQL.
* NoSQL: MongoDB for dynamic data.

**AI/ML**

* Frameworks: TensorFlow or PyTorch for model development.
* Deployment: Flask/Django REST API for AI endpoints.

**APIs**

* Mpesa: Safaricom’s Mpesa API.
* Banking: Open banking APIs (e.g., Plaid).
* SMS/Email: Twilio or SendGrid.

**Hosting**

* **Server**: AWS EC2, Google Cloud, or Azure.
* **Storage**: AWS RDS, Firebase, or MongoDB Atlas.

**7. Challenges and Solutions**

**Challenge 1: Data Privacy**

* Solution: End-to-end encryption, compliance with data laws.

**Challenge 2: AI Accuracy**

* Solution: Regular model training and feedback integration.

**Challenge 3: API Integration**

* Solution: Test APIs rigorously and handle errors gracefully.

**8. Deliverables**

1. Fully functional API with all endpoints.
2. Secure database for storing user and transaction data.
3. Integrated AI model for insights and recommendations.
4. Real-time payment logging with Mpesa and banks.