**Project Title: Personal Budget Tracker**

## **Introduction**

The Personal Budget Tracker is a web-based application designed to help users efficiently manage their financial transactions, track expenses, and set budget goals. The system provides an intuitive interface for users to monitor their income and expenditures, ensuring better financial planning.

## **Core Features**

### **1. User Authentication**

* Secure user registration, login, and logout functionality.
* Personalized data for each user to ensure privacy and customization.

### **2. Account/Wallet Management**

* Users can create and manage multiple accounts or wallets (e.g., Checking, Savings, Credit Cards).
* Each account includes:
  + Name
  + Type
  + Current balance

### **3. Transaction Tracking**

* Users can record income and expense transactions.
* Each transaction includes:
  + Date
  + Amount
  + Description
  + Category (e.g., Groceries, Rent, Salary)
  + Account/Wallet
  + Transaction type (Income or Expense)

### **4. Expense Categories**

* Predefined categories such as Housing, Groceries, and Travel.
* Option to add custom categories.

### **5. Date Filtering**

* Filter income and expense data by specific date ranges (e.g., Monthly, Weekly).

### **6. Budget Goals**

* Set monthly or category-specific budget limits.

### **7. Dashboard**

* A summary view displaying:
  + Total Income
  + Total Expenses
  + Remaining Balance
  + Spending Trends

## **Modular Django Apps**

### **1. Users App**

* Manages user accounts, authentication, and profiles.

### **2. Wallets App**

* Handles user wallets/accounts (e.g., Checking, Savings).

### **3. Transactions App**

* Manages income and expense transactions.

### **4. Budgets App**

* Manages user budgets for different categories.

### **5. Dashboard App**

* Provides analytics and visual representations of financial data.

## **Core Models and Relationships**

### **Users App**

#### **Custom User Model (AbstractUser)**

* Extends Django’s AbstractUser.
* Fields:
  + Standard authentication fields (username, email, password).
* Relationships:
  + One-to-Many with Transactions and Wallets.
* Constraints:
  + Email field must be unique.

### **Wallets App**

#### **Wallet Model**

* Fields:
  + name: Name of the wallet (e.g., "Checking").
  + balance: Current balance.
  + type: Type of wallet (Checking, Savings, Credit).
  + created\_at: Timestamp of creation.
  + user: ForeignKey linked to User.
* Relationships:
  + One-to-Many with Transactions.
* Constraints:
  + Each wallet belongs to a single user.
  + Balance must be greater than or equal to 0 for non-credit wallets.

### **Transactions App**

#### **Transaction Model**

* Fields:
  + amount: Amount of the transaction.
  + type: Income or Expense.
  + category: ForeignKey linked to Category.
  + wallet: ForeignKey linked to Wallet.
  + date: Transaction date.
  + description: Optional transaction details.
  + user: ForeignKey linked to User.
* Relationships:
  + Many-to-One with Category, Wallet, and User.
* Constraints:
  + Amount must be positive.
  + Each transaction must be linked to a valid wallet and category.

#### **Category Model**

* Fields:
  + name: Category name (e.g., Food, Rent).
  + user: ForeignKey linked to User for custom categories.
* Relationships:
  + Many-to-One with Transactions.
* Constraints:
  + Category name must be unique per user.

### **Budgets App**

#### **Budget Model**

* Fields:
  + amount: Budget amount.
  + category: ForeignKey linked to Category.
  + wallet: ForeignKey linked to Wallet (optional).
  + month: Budget month.
  + user: ForeignKey linked to User.
* Relationships:
  + Many-to-One with Category, Wallet, and User.
* Constraints:
  + Amount must be positive.
  + No duplicate budgets for the same category in the same month.

### **Notification Model (Optional)**

* Fields:
  + message: Notification content.
  + timestamp: Time of notification.
  + user: ForeignKey linked to User.

## **API Endpoints**

### **Users API**

* POST /api/users/register/ - Register a new user.
* POST /api/users/login/ - Log in an existing user.
* GET /api/users/profile/ - Fetch user profile details.

### **Wallets API**

* GET /api/wallets/ - List all wallets for the user.
* POST /api/wallets/ - Create a new wallet.
* GET /api/wallets/{id}/ - Fetch wallet details.
* PUT /api/wallets/{id}/ - Update wallet details.
* DELETE /api/wallets/{id}/ - Delete a wallet (if no transactions exist).

### **Transactions API**

* GET /api/transactions/ - List all transactions.
* POST /api/transactions/ - Add a transaction.
* PUT /api/transactions/{id}/ - Update a transaction.
* DELETE /api/transactions/{id}/ - Delete a transaction.
* GET /api/transactions/?wallet\_id={id} - Filter transactions by wallet.

### **Budgets API**

* GET /api/budgets/ - List all budgets.
* POST /api/budgets/ - Add a new budget.
* PUT /api/budgets/{id}/ - Update a budget.
* DELETE /api/budgets/{id}/ - Delete a budget.

### **Dashboard API**

* GET /api/dashboard/summary/ - Fetch total income, expenses, and balance.
* GET /api/dashboard/summary/?wallet\_id={id} - Fetch summary for a specific wallet.
* GET /api/dashboard/categories/ - Spending breakdown by category.

## **Working Plan**

1. **Design ER Diagram**
2. **Set Up Django Project and Environment**
3. **Develop the Users App with Authentication**
4. **Implement the Wallets App**
5. **Develop the Transactions App**
6. **Integrate Budget Tracking Features**
7. **Build Dashboards and Visualizations for User Insights**

This structured approach ensures a scalable, modular, and efficient Personal Budget Tracker, allowing users to manage their finances effectively.