**Project Overview: Savvy - Save**

Savvy - Save is a personal finance tracker app designed to help users manage their financial data. The app offers a variety of features like transaction management, tracking stats, setting Key Performance Indicators (KPIs), and editing profiles. It includes a clean and modern design, with an intuitive user interface and robust backend, making it suitable for everyday finance management.

**Technology Stack:**

* **Design Pattern**: MVC (Model-View-Controller)
* **Backend**: **Supabase** (for authentication, database, and storage)
* **Logic**: **Python**
* **UI**: **HTML**, **CSS** (with **Bootstrap** for responsiveness)
* **API**: **FastAPI** or **REST API**
* **Bridge**: **JavaScript**
* **Version Control**: **GitHub**

**Features:**

1. **Login Page**:
   * **Fields**: Email or Username, Password
   * **Buttons**: Login, Face ID Login (using pre-trained model **FaceNet**)

**Description**: Users can login using either their email or username and password. Face ID login will use **FaceNet**, a pre-trained facial recognition model, to authenticate users based on their facial features.

1. **Register Page**:
   * **Fields**: Username, Email, Password, Image Upload (or Camera Capture for profile image)
   * **Button**: Sign-up
   * **Functionality**: User is able to sign up by providing necessary details. Upon successful registration, the user's image will be uploaded to a Supabase bucket named **userimage**, and the URL of the image will be saved to the database.
2. **Left Navigation Bar**:
   * **App Title**: "Savvy - Save"
   * **Buttons**: Page Navigation (Dashboard, Transaction, Stats, Profile), Sign-out Button

**Description**: A consistent left navigation bar will provide access to different sections of the app. The user can easily navigate between pages like Dashboard, Transaction, Stats, and Profile.

1. **Dashboard Page**:
   * **Design**:
     + Display a **credit card-like** div showing the user's balance and card information.
     + **Line Chart**: Show total income and total expense by month.
     + **Transaction Table**: Display transaction details with columns: Transaction ID, Date, Category, Transaction Type, Amount, Note.
     + **Pie Chart**: Show total expense distribution by category.
     + **Stats Divs**: Display total income and expenses, with comparison against the last month.
     + **Filters**: Year filter for the charts and stats.
     + **Export Button**: Option to export data (CSV, PDF, etc.).
2. **Transaction Page**:
   * **Fields**:
     + Add, Delete, Filter, Edit, and Find transactions.
     + Add, Delete, Filter, Edit, and Find KPIs.
   * **Transaction Table**: Similar to the one in the dashboard but with more detailed options.

**Description**: Users can easily manage their transactions with options to add, delete, edit, or filter based on different parameters. Similarly, KPIs can be tracked and managed here.

1. **Stats Page**:
   * **Stats**:
     + Total Income, Total Expense, Total Savings
     + Category Unique Count
   * **Charts**:
     + **Line Chart**: Track savings/balance vs. goal by month.
     + **Bar Chart**: Show total savings/balance vs. goal.
   * **Table**: Category-wise savings and goals comparison.
2. **Profile Page**:
   * **User Information**: Image, Username, Email (displayed as asterisks for security), Card Number, Description.
   * **Edit Button**: Allow users to update their profile details.

**Key Design Considerations:**

* **Modern UI**: Use **Bootstrap** for responsive design, making sure the layout is user-friendly on both desktop and mobile.
* **User Experience (UX)**: Clear separation of content, with navigation menus and actions easily accessible.
* **Image Storage**: When a user uploads an image, it will be saved to a Supabase bucket called **userimage**, and only the URL will be stored in the database.
* **FaceID**: Use **FaceNet**, a pre-trained facial recognition model for face authentication. This adds an extra layer of security for users who prefer to log in via Face ID.
* **OOP (Object-Oriented Programming)**: The code should follow object-oriented principles for better maintainability and scalability.

**Backend - FastAPI & Supabase Integration:**

1. **Authentication**:
   * Use **Supabase Authentication** for managing user sign-ins and registrations.
   * **FaceNet Integration** for Face ID login: You'll need to pre-train the FaceNet model on a set of images to perform facial recognition for user authentication.
2. **Data Storage**:
   * **Supabase**: Store user data (email, username, transactions, etc.) in Supabase’s PostgreSQL database.
   * **File Storage**: User profile images are stored in **Supabase Storage** (bucket: userimage).
3. **API Routes** (FastAPI):
   * **POST** /register: Registers a new user.
   * **POST** /login: Logs in using either email/password or FaceID.
   * **GET** /transactions: Retrieves the list of transactions.
   * **POST** /transaction: Adds a new transaction.
   * **GET** /stats: Fetches the stats (income, expense, savings).
   * **PUT** /profile: Allows users to edit their profile.
4. **Database Structure (Supabase)**:
   * **Users Table**: Stores user credentials, profile image URL, etc.
   * **Transactions Table**: Stores transaction records (ID, date, category, transaction type, amount, note).
   * **Stats Table**: Stores stats data (income, expense, savings).

**Frontend - HTML, CSS (Bootstrap), JavaScript:**

1. **Login and Register Pages**:
   * Use **HTML** for structure, **CSS (Bootstrap)** for styling, and **JavaScript** for handling form submissions (e.g., AJAX requests to FastAPI backend).
2. **Dashboard Page**:
   * Use **Bootstrap** components for grid layout, card design, and charts (with **Chart.js** or **Plotly**).
3. **Transaction and Stats Pages**:
   * Tables: Use **Bootstrap’s table components**.
   * Charts: Use **Chart.js** for generating line and pie charts.

C:/Users\huynq/AppData/Local/Programs/Python/Python310/python.exe -m venv venv

venv\Scripts\activate

pip install fastapi uvicorn supabase fastapi[all]

pip freeze > requirements.txt

pip list

uvicorn main:app --reload