BudgetApp

App Requirements

The various Minimum Viable Product features includes the following:

**User Management Includes:**

* *Signup***:**
  + Users can create an account with email + password.
  + Passwords stored securely (hashed, not plain text).
* *Login***:**
  + Users log in with email + password.
  + System returns a secure token (JWT) to keep them logged in.

**Transactions**

* *Add income/expense:*
  + Users enter: amount, type (income/expense), category, date, and optional note.
* *View transactions:*
  + Users see a list of their past income/expenses.
  + Transactions should be linked to the user account (each user sees only their own data).

**Categories**

* Foods & Drinks :

1. Fast Food
2. Restaurant, fast-food
3. Groceries

* Shopping:

The following are the categories involved :-

* Drug-store, chemist
* Free time
* Stationery, tools
* Gifts, joy
* Electronics, accessories
* Pets, animals
* Home, garden

1. Toilertries
2. Kitchen

* Kids
* Health and beauty
* Jewels, accessories
* Men's

1. Fragrances
2. Footwear
3. Clothes

* Housing

1. Energy nd Utilities

* Transport
* Vehicle
* Life & Entertainment

1. TV, Streaming
2. Activeness sport nd fitness
3. Holiday nd trips

* Communication and PC

1. Internet
2. Airtime
3. Bundles

* Financial Expenses

1. Charges & fees
2. Loans & intrests

* Investments

1. Trade
2. MMF
3. Savings

* Income
* Others

***Assign category:*** Every transaction must belong to one category.

**Monthly Summary**

* **Auto-summary:**
  + Show total income, total expenses, and balance for the month.
  + Breakdown by category (e.g., Food = 25% of expenses).

**Export Data**

* **CSV export:** Users can download their transaction history as a spreadsheet.
* **PDF export:** Optionally generate a simple PDF report of the monthly summary.

Tools

* **Python** (for backend with Flask or FastAPI)

Download: [python.org](https://www.python.org/downloads/)

* **Flutter SDK** (for frontend mobile app)

Download: flutter.dev

* **VS Code** (recommended editor for both Python + Flutter)

Extensions: Flutter, Python

* Android Studio
* **Git & GitHub account** (for version control)

Install Git: [git-scm.com](https://git-scm.com/downloads)

* **Postman** (for testing APIs easily).

Project Structure

**budget\_app\_backend/**

│── venv/ # Python virtual environment

│── app.py # Main Flask entry point

│── models.py # Database models (User, Transaction, Category)

│── routes.py # API routes (signup, login, transactions)

│── database.py # DB setup (SQLAlchemy)

│── requirements.txt # Python dependencies

│── README.md # Documentation for backend

**budget\_app\_frontend/**

│── lib/

│ ├── main.dart # Entry point

│ ├── pages/

│ │ ├── login\_page.dart # Login UI

│ │ ├── signup\_page.dart # Signup UI

│ │ ├── dashboard\_page.dart # Transaction list

│ │ └── add\_txn\_page.dart # Add income/expense

│ ├── services/

│ │ └── api\_service.dart # Connects Flutter to Flask API

│ ├── models/

│ │ └── transaction.dart # Transaction data model

│ └── state/

│ └── app\_state.dart # State management (Riverpod/Provider)

│── pubspec.yaml # Flutter dependencies

│── README.md # Documentation for frontend

🏗 Backend Development Tasks – Beginner Notes

## 1️⃣ Create Models with SQLAlchemy (models.py)

**Models** are like **blueprints for your database tables**. Each model defines the structure of a table.

* **User model**

class User(db.Model):

id = db.Column(db.Integer, primary\_key=True)

email = db.Column(db.String(120), unique=True, nullable=False)

password\_hash = db.Column(db.String(128), nullable=False)

* + id → unique identifier for each user
  + email → user’s email, must be unique
  + password\_hash → encrypted password
* **Category model**

class Category(db.Model):

id = db.Column(db.Integer, primary\_key=True)

name = db.Column(db.String(50), nullable=False)

* + Categories like “Food”, “Salary”, “Entertainment”
* **Transaction model**

class Transaction(db.Model):

id = db.Column(db.Integer, primary\_key=True)

user\_id = db.Column(db.Integer, db.ForeignKey('user.id'), nullable=False)

amount = db.Column(db.Float, nullable=False)

type = db.Column(db.String(10), nullable=False) # income or expense

category\_id = db.Column(db.Integer, db.ForeignKey('category.id'))

date = db.Column(db.Date, nullable=False)

note = db.Column(db.String(200))

* + Links **transactions** to users and categories
  + Stores amount, type (income/expense), date, and optional note

✅ **Tip:** Think of models as the **skeleton of your database**. You define what each table should store here.

## 2️⃣ Set up Database Connection (database.py)

We need a **connection** so our app can talk to the database.

from flask\_sqlalchemy import SQLAlchemy

db = SQLAlchemy()

Then in your main app:

from flask import Flask

from database import db

app = Flask(\_\_name\_\_)

app.config['SQLALCHEMY\_DATABASE\_URI'] = 'sqlite:///budget.db'

db.init\_app(app)

* SQLALCHEMY\_DATABASE\_URI → tells SQLAlchemy where the database is
* db.init\_app(app) → connects SQLAlchemy to your Flask app

✅ **Tip:** For beginners, **SQLite** is easiest because it’s just a single file.

## 3⃣ Apply Migrations or Auto-Create SQLite Database

* **Migrations** → track changes to database tables over time
* **For beginners:** You can auto-create tables quickly:

with app.app\_context():

db.create\_all()

* This reads your models and creates tables in budget.db automatically.

✅ **Tip:** Later, when your app grows, learn **Flask-Migrate** for proper migrations.

## ****Visual Map****

┌───────────────────┐

│ app.py │

│ │

│ - Creates Flask │

│ app instance │

│ - Configures DB │

│ - Registers routes│

└─────────┬─────────┘

│

│ imports db & models

▼

┌───────────────────┐

│ database.py │

│ │

│ - Creates SQLAlchemy │

│ instance (db) │

│ - Handles DB connection │

└─────────┬─────────┘

│

│ used by models

▼

┌───────────────────┐

│ models.py │

│ │

│ - Defines tables │

│ (User, Category,│

│ Transaction) │

│ - Uses db from │

│ database.py │

└───────────────────┘

### **Step-by-Step Explanation**

1. **app.py**
   * Main file that runs your Flask app
   * Imports db from database.py and models from models.py
   * Configures the database URI and initializes SQLAlchemy
2. **database.py**
   * Only job: create the SQLAlchemy **database instance (db)**
   * This instance is shared across the app and models
3. **models.py**
   * Defines **all database tables** using SQLAlchemy
   * Uses the db object from database.py to define columns, relationships, etc.

### ✅ **Analogy for beginners:**

Think of it like building a house:

* database.py → provides the **bricks and cement** (the database connection)
* models.py → draws the **blueprints** for each room (tables)
* app.py → is the **construction manager** that puts everything together and runs the house (app)

📌 Concept 1: **Model**

* A **Model** is a Python class that represents a table in the database.
* Example:

class Category(db.Model):

id = db.Column(db.Integer, primary\_key=True)

name = db.Column(db.String(100), nullable=False)

parent\_id = db.Column(db.Integer, db.ForeignKey('category.id'))

* Here, Category is the **model**, and it defines how the **categories table** looks.

📌 Concept 2: **Parent–Child Relationship (Self-Referencing Foreign Key)**

* A **Foreign Key** connects one row in a table to another row in the same or different table.
* In this case, a category can be a **parent** or a **child (subcategory)**.
* Example: "Foods & Drinks" (parent) → "Fast Food" (child).
* This makes your categories **hierarchical**.

📌 Concept 3: **Seeding (Database Seeding)**

* **Seeding** means adding initial data into the database automatically.
* We wrote seed\_categories.py to insert all categories and subcategories.
* This is common in new apps (to preload data like roles, admin user, categories, etc.).

📌 Concept 4: **App Context (Flask App Context)**

* Flask needs to know which app is running before using the database.
* We use:

with app.app\_context():

# database queries go here

* Without **App Context**, you’ll get an error: "Working outside of application context".

📌 Concept 5: **Querying the Database (SQLAlchemy Query)**

* To fetch data:

categories = Category.query.all()

.all() → gets all records

.first() → gets the first record

.filter\_by(name="Foods & Drinks").first() → gets a specific record

📌 Concept 6: **Primary Key (id)**

* Every row in the database needs a **unique identifier**.
* In our model, id = db.Column(db.Integer, primary\_key=True) means id is the **primary key**.

📌 Concept 7: **Foreign Key (parent\_id)**

* Connects one record to another.
* parent\_id links a subcategory to its parent.
* Example: "Fast Food" has parent\_id = 1, meaning its parent is "Foods & Drinks".

## How to verify

That your categories actually exist in the database. Let’s do it

📌 Step 1: Open Flask Shell

From your backend folder (where app.py is), run in terminal:

python

This opens the Python shell.  
(If you prefer, you can also use **Flask shell** if configured, but for beginners plain Python is fine.)

📌 Step 2: Import App and Database

Inside the shell:

from app import app

from database import db

from models import Category

📌 Step 3: Use App Context

Flask needs an **app context** to talk to the database. Enter:

with app.app\_context():

categories = Category.query.all()

for c in categories:

print(c.id, c.name, "-> Parent:", c.parent\_id)

📌 Step 4: Expected Output

You’ll see something like:

1 Foods & Drinks -> Parent: None

2 Fast Food -> Parent: 1

3 Restaurant, fast-food -> Parent: 1

4 Groceries -> Parent: 1

5 Shopping -> Parent: None

6 Drug-store, chemist -> Parent: 5

7 Free time -> Parent: 5

...

* Parent: None = main category
* Parent: [id] = subcategory belonging to that parent

📌 Step 5: Exit Shell

When you’re done, type:

exit()

✅ That’s it! Now you’ve **seeded** your categories and **confirmed** they exist.