Progress Tracker: Week 37, 2025

**✅ Day 1 – Budget App Journal Checklist**

**📝 Planning**

* Write down **MVP features**:
  + User signup/login
  + Add income/expense
  + Categorize transactions
  + Monthly summary
  + Export CSV/PDF (optional)
* Note **non-MVP features** (bank APIs, AI, recurring transactions, notifications).

**⚙️ Setup Tools**

* Install **Python** (latest version).
* Install **Flutter SDK**.
* Install **VS Code** + extensions (Python, Flutter).
* Install **Git**.
* (Optional) Install **Postman** for testing APIs.
* Create a folder: BudgetAppProject/.

**🏗 Backend Setup (Flask)**

* Create folder budget\_app\_backend/.
* Create and activate a Python virtual environment.
* Install dependencies (flask, flask-restful, flask-jwt-extended, flask-cors, sqlalchemy).
* Create app.py with “Hello World” route.
* Test by running python app.py → confirm in browser.
* Create requirements.txt.

**🎨 Frontend Setup (Flutter)**

* Create folder budget\_app\_frontend/ using flutter create budget\_app\_frontend.
* Run default Flutter app (flutter run) → confirm it works.

**🌐 GitHub Setup**

* Create **GitHub repo**: budget\_app\_backend.
* Create **GitHub repo**: budget\_app\_frontend.
* Initialize Git in backend folder → commit → push.
* Initialize Git in frontend folder → commit → push.

**🗂 Documentation**

* Write a simple **README.md** for backend (how to run Flask app).
* Write a simple **README.md** for frontend (how to run Flutter app).
* Journal entry: note what worked, what was tricky, and what you learned.

👉 By the end of Day 1, you should have:  
✔ A “Hello World” Flask backend running.  
✔ A default Flutter app running.  
✔ Both projects uploaded to GitHub.  
✔ Your journal updated with progress + reflections.

✅ Day 2 – Budget App Journal Checklist

### 🎯 Goal

Set up the **Backend MVP** with authentication and transaction handling.

### 🏗 Backend Development Tasks

* Create **models** with SQLAlchemy:
  + User → id, email, password\_hash
  + Category → id, name
  + Transaction → id, user\_id, amount, type, category\_id, date, note
* Set up **database connection** (database.py)
* Apply **migrations** or auto-create **SQLite database - Simple**

### 🔑 Authentication

* Install & configure Flask-JWT-Extended
* Create **signup endpoint** (POST /auth/signup)
* Create **login endpoint** (POST /auth/login)
* Return **JWT token** on successful login

### 🔄 Transactions API

* Create **GET /transactions** → fetch user’s transactions
* Create **POST /transactions** → add new income/expense
* Ensure **JWT authentication** is required for all endpoints

### 📖 API Documentation

* Install **Flasgger** (or switch to **FastAPI** if preferred)
* Add **Swagger/OpenAPI docs** for:
  + Signup
  + Login
  + Transactions

### 🧪 Testing

* Test **signup & login** with Postman
* Confirm JWT token works for **protected routes**
* Test **creating a transaction** and **fetching transactions**

### 🗂 Documentation & Journal

* Update **backend README.md** with new endpoints
* Journal entry:
  + Endpoints created & tested
  + Bugs or issues encountered

### 👉 By the end of Day 2, you should have:

* ✔ User signup & login working with JWT
* ✔ Ability to add and view transactions (linked to user)
* ✔ API documented (Swagger/OpenAPI)
* ✔ Tested endpoints in Postman

✅ Day 3 – Budget App Journal Checklist

### 🎯 Goal for Day 3

Build the **basic Flutter UI** with mock data (no backend connection yet).

### 🎨 Flutter UI Tasks

* Create **Login Page**
  + Email + password text fields
  + Login button
  + Navigation to Signup page
* Create **Signup Page**
  + Email + password text fields
  + Signup button
  + Navigation back to Login page
* Create **Dashboard Page**
  + Display list of transactions (use mock static data)
  + Show balance summary at the top
  + Floating Action Button (➕) to add new transaction
* Create **Add Transaction Page**
  + Form with fields: amount, type (income/expense), category, date, note
  + Save button (for now, just print input to console)

### 📦 Flutter Setup & Dependencies

* Add **Riverpod** (or Provider) for state management
* Add **dio** for API calls (will use tomorrow when connecting to backend)
* Add **flutter\_secure\_storage** for JWT tokens (setup only, use later)

### 🧪 Testing

* Run the app in emulator/device
* Navigate between login → signup → dashboard → add transaction
* Confirm forms work and print data to console

### 🗂 Documentation & Journal

* Update frontend **README.md** with instructions to run Flutter app
* Journal entry: note which pages were created and any design challenges

👉 By the end of Day 3, you should have:  
✔ A working **Flutter UI skeleton** (Login, Signup, Dashboard, Add Transaction).  
✔ Navigation between pages.  
✔ Mock data showing in Dashboard.  
✔ Setup of packages for future API integration.

Ah, got it! You want the **actual checkboxes** (✅ or ⬜) like in your Day 3 checklist, so you can **tick off tasks as you go**. Here’s the Day 4 checklist updated with checkboxes:

✅ Day 4 – Budget App Journal Checklist

### 🎯 Goal for Day 4

Connect the Flutter frontend to the Flask backend and enable real data flow.

### 🎨 Flutter UI + Backend Tasks

* ⬜ **Integrate Signup/Login**
  + ⬜ Create POST requests to /signup and /login endpoints using Dio
  + ⬜ Send email + password
  + ⬜ Save JWT token using flutter\_secure\_storage on success
  + ⬜ Display error messages on failure
* ⬜ **Persist JWT & Attach to Requests**
  + ⬜ Store token securely after login/signup
  + ⬜ Include token in Authorization headers for protected endpoints
* ⬜ **Dashboard Page**
  + ⬜ Fetch user transactions from /transactions endpoint
  + ⬜ Display real data instead of mock transactions
  + ⬜ Update list dynamically when new transactions are added
* ⬜ **Add Transaction Page**
  + ⬜ Send POST request to /transactions endpoint
  + ⬜ Include fields: amount, type, category, date, note
  + ⬜ Update Dashboard after successful addition
* ⬜ **Handle CORS in Flask**
  + ⬜ Install and configure Flask-CORS
  + ⬜ Allow requests from your emulator/device during development

### 📦 Flutter Setup & Dependencies

* ⬜ **Dio** – already installed; use for API calls
* ⬜ **flutter\_secure\_storage** – already installed; use to persist JWT tokens
* ⬜ **Riverpod** – manage state for authentication and transaction lists

### 🧪 Testing

* ⬜ Test signup/login → confirm JWT is saved securely
* ⬜ Test fetching transactions → Dashboard shows real backend data
* ⬜ Test adding new transactions → POST request updates backend and UI
* ⬜ Debug common issues: CORS errors, wrong URLs, missing headers

### 🗂 Documentation & Journal

* ⬜ Update README with instructions on connecting to backend
* ⬜ Journal entry: note which endpoints were integrated, errors faced, and fixes applied
* ⬜ Record any challenges with API requests, JWT persistence, or state updates