Project Title:

Expense Manager Using Python (CSV & File Handling)

Abstract :

The Expense Manager is a lightweight Python application that enables users to record, view, and calculate daily expenses using CSV file storage and TXT backups. The system offers a simple menu-driven interface (CLI) and a web interface (Flask) that allow users to add expenses with a date, category, amount, and description. All entries are persisted to a CSV file for ease of analysis and to a TXT backup file for redundancy. This project demonstrates practical use of file handling, CSV manipulation, and basic web development with Flask, providing a reliable tool for personal financial tracking.

Introduction :

Effective personal finance tracking helps individuals control spending and save for goals. Manual tracking methods are error-prone and inconvenient. This project implements an Expense Manager using Python to automate recording and reporting of daily expenses. The program uses CSV for structured, tabular data storage, enabling easy viewing and importing into spreadsheets. A TXT backup log is maintained to ensure transaction history is preserved, preventing accidental loss. A simple Flask web interface provides a more user-friendly front end so the application can run locally in a browser.

Key goals

- Provide an easy way to record daily expenses.
- Store expenses in CSV for portability and analysis.
- Maintain a simple TXT backup log for safety.
- Provide both CLI and web UI for flexibility.

Proposed System :

The proposed system is a **web-based Expense Manager application** designed to help users efficiently manage their income and expenses in a centralized platform. It provides a secure login system that ensures only authenticated users can access their personal financial data. The system allows users to add income, record expenses with predefined categories, and maintain a complete history of transactions. It also features a modern dashboard that visually represents financial insights such as monthly spending trends and category-wise distribution using interactive charts. The application uses CSV files for lightweight data storage while ensuring faster access and easy portability. A backup log is generated automatically for every transaction, improving reliability. The system aims to provide a simple yet powerful interface that helps users track, analyze, and improve their financial habits.

Detailed Explanation of All Modules

1. Menu / Controller Module

- Responsible for presenting choices to the user (Add, View, Total, Exit).
- Validates user input and routes commands to appropriate modules.
- In the web version, routes (/, /add, /view) map to templates and handlers.

2. Input Module

- Accepts expense details: date, category, amount, description (optional).
- Validates fields (amount numeric, date format).
- Prepares a row for storage: [date, category, amount, description].

3. CSV Storage Module

- Handles writing expense rows to expenses.csv.
- On first run, creates the file with header row: Date,Category,Amount,Description.
- Uses Python's csv module to append rows safely (handles quoting, newline issues).
- Provides read function to load all rows for display or calculation.

4. Backup Logger Module (TXT)

- Appends a human-readable log entry to backup.txt for every new expense:

- [2025-11-30] Added Expense -> Food : 120 - Lunch at canteen
- Acts as an audit trail and minimal undo/history record.

5. Display / View Module

- CLI: reads CSV and prints each record (or formatted table).
- Web: reads CSV and passes rows to template for tabular rendering.
- Includes optional filters: by date, by category (can be added).

6. Calculation Module

- Reads the Amount column from CSV and computes totals (daily, monthly, grand total).
- Returns totals as numeric values (float) to display.

7. Persistence / Initialization Module

- On start-up, ensures CSV file exists and has headers.
- Handles any required migrations or data checks.

8. Flask Web Module (for web version)

- app.py sets up routes:
    - / — homepage (links)
    - /add — form to add expense (GET/POST)
    - /view — lists all expenses and shows total
- Uses templates/ folder with index.html, add.html, view.html.
- Uses CSV module for storage (same CSV as CLI version).

Architecture:

1. Client–Server Architecture

The system follows a browser-based client–server architecture where the **client (web browser)** sends requests via HTTP and the **Flask server** processes the logic and responds with HTML pages or JSON data.

---

2. MVC-Inspired Layered Structure

The application uses a simplified **Model–View–Controller pattern**:

- **Model:** CSV-based storage (users.csv, transactions.csv)
- **View:** HTML templates (Jinja2)
- **Controller:** Flask route functions handling logic

---

3. Modular Backend Components

The backend is divided into logical modules:

- **Authentication module** (Register/Login/Logout)
- **Transaction module** (Add/Edit/Delete expenses & income)
- **Analytics module** (Charts & summary computation)
- **Export module** (CSV exporting)

---

4. CSV-Based Data Persistence Layer

Instead of a database, the system uses CSV files for:

- **User accounts**
- **Transactions**
  This lightweight storage allows easy reading, writing, and portability.

---

5. Secure Password Handling

All passwords are stored using:

- **SHA-256 hashing via Werkzeug**
  This ensures sensitive data is not stored in plain text.

---

6. Session Management for Logged-In Users

Flask session cookies are used to:

- Maintain user login state
- Identify which user's data to load
- Prevent unauthorized access

---

7. REST-Like API Endpoints for Charts

Analytics page fetches data from:

- /api/monthly
- /api/categories
  These return JSON results for Chart.js graphs.

---

8. Frontend Built with Bootstrap + Chart.js

The view layer contains:

- Responsive UI using Bootstrap
- Interactive Line & Pie charts using Chart.js
- Reusable base.html layout template

---

9. Role-Based Data Isolation

Every request is tied to a particular user via session.
Users can access:

- **Only their own transactions**
- **Only their own analytics**
  Ensuring privacy and isolated data processing.

10. Modular Template Architecture

The HTML templates use:

- Jinja2 inheritance (base.html)
- Reusable UI blocks
- Shared navigation & design
  This separates logic from presentation and improves maintainability.

11. Reusable Utility Functions Layer

The system includes a set of reusable helper functions for tasks such as reading/writing CSV data, validating inputs, calculating totals, and formatting analytics. This reduces code duplication and increases maintainability.

Architecture :

1. Client–Server Architecture for handling browser requests and server responses.
2. MVC-inspired structured design separating model, view, and controller layers.
3. Modular backend components for authentication, transactions, analytics, and user sessions.
4. CSV-based data storage to maintain lightweight and portable datasets.
5. Secure authentication using hashed passwords for user privacy.
6. Session management to track logged-in users and isolate user-specific data.
7. REST-like API endpoints for analytics and chart data processing.
8. Bootstrap-based responsive UI ensuring compatibility across devices.

9. Chart.js integration for visual representation of financial analytics.

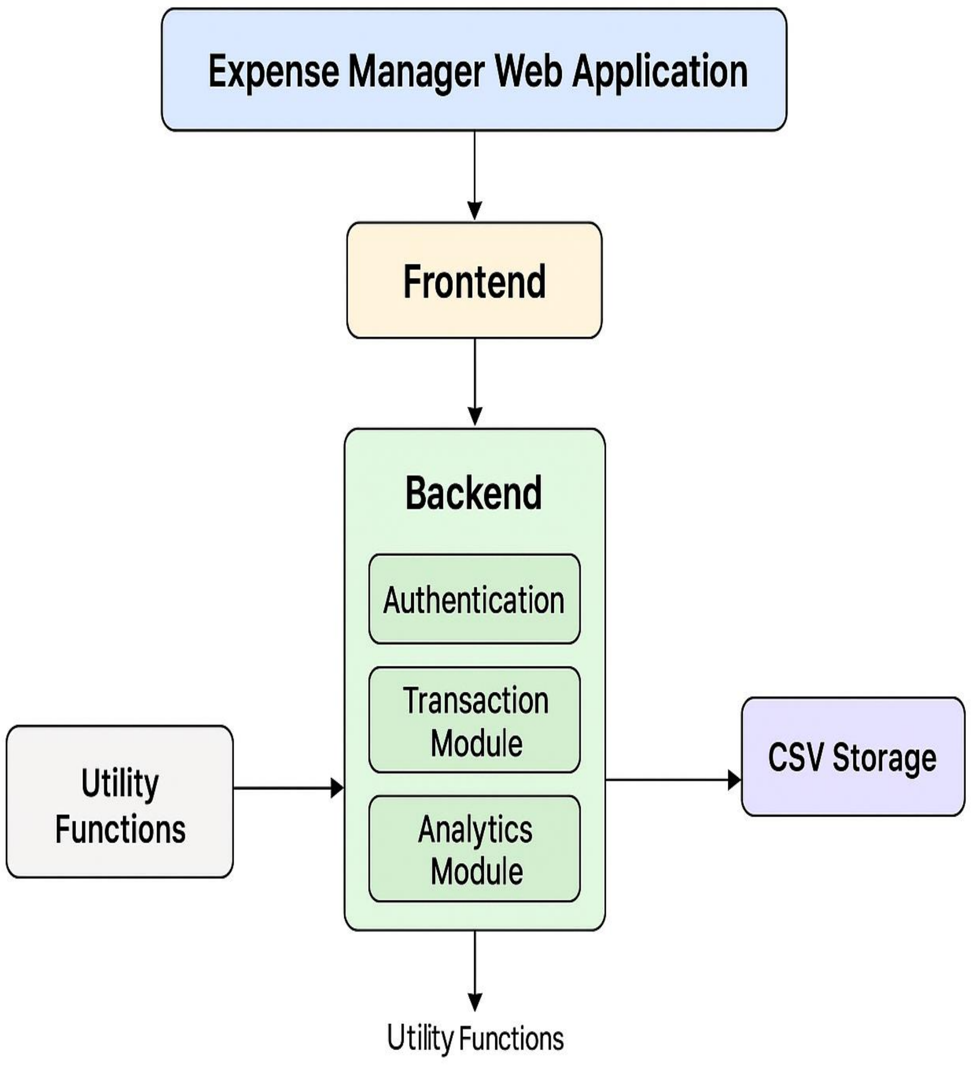
10. Modular template architecture using Jinja2 template inheritance.

11. Reusable utility functions to avoid code duplication.

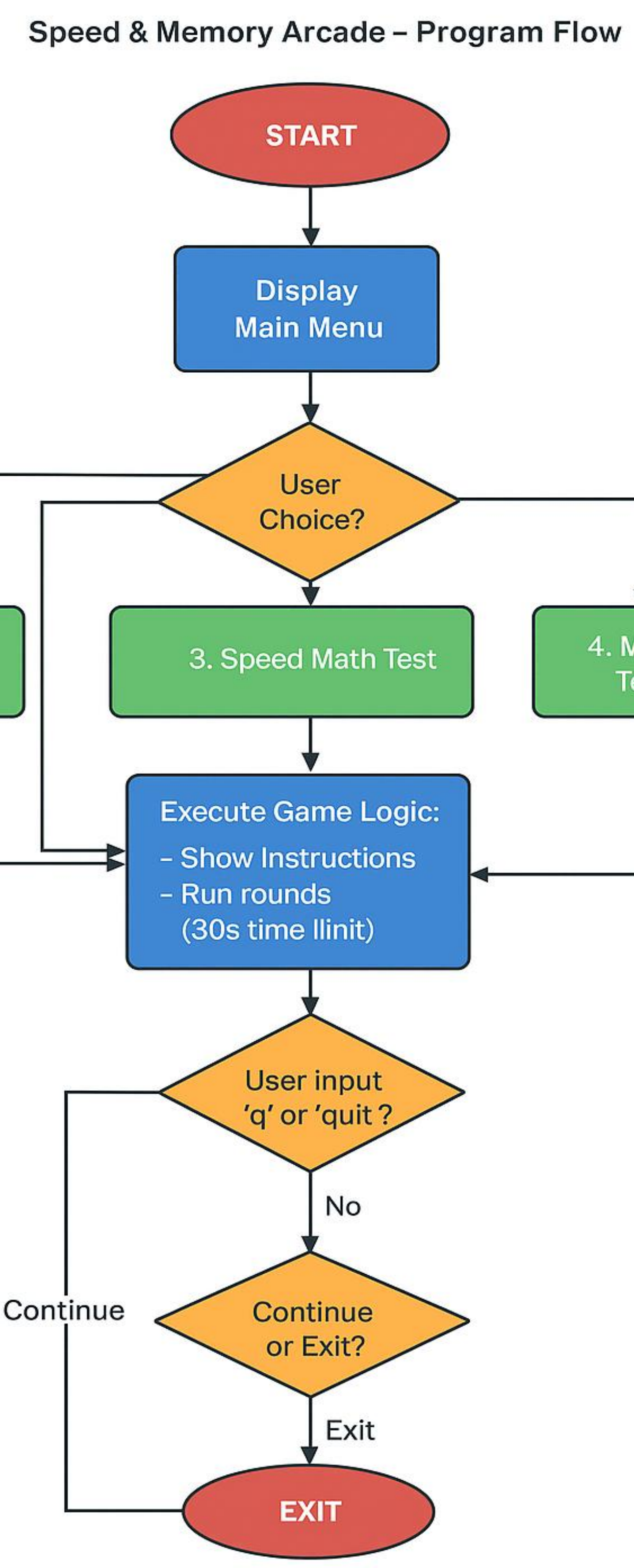
12. Responsive frontend architecture for smooth mobile/desktop adaptation.

13. Analytics computation layer for monthly, category-wise, and summary calculations

Architecture Diagram:



Flowchart :


Speed & Memory Arcade – Program Flow
START
Display Main Menu
User Choice?
1. Follow the Command
3. Speed Math Test
4. Memory Tester
Execute Game Logic:
– Show Instructions
– Run rounds (30s time llinit)
User input 'q' or 'quit ?
No
Continue
Continue or Exit?
Exit
EXIT

Python Code:

```
import csv

import os

import uuid

from datetime import datetime

from flask import Flask, render_template, request, redirect, url_for, session, flash, jsonify,
send_file

from werkzeug.security import generate_password_hash, check_password_hash

from io import StringIO


app = Flask(__name__)

app.secret_key = "replace_with_a_random_secret_key"


USERS_FILE = "users.csv"

TX_FILE = "transactions.csv"

CATEGORIES = ["Food","Social","Transport","Apparel","Education","Gift","Other"]


# Ensure files exist

if not os.path.exists(USERS_FILE):

    with open(USERS_FILE, "w", newline="") as f:

        writer = csv.writer(f)

        writer.writerow(["username","password_hash"])


if not os.path.exists(TX_FILE):

    with open(TX_FILE, "w", newline="") as f:

        writer = csv.writer(f)

        writer.writerow(["id","username","type","date","category","amount","description"])


# Utility functions
```

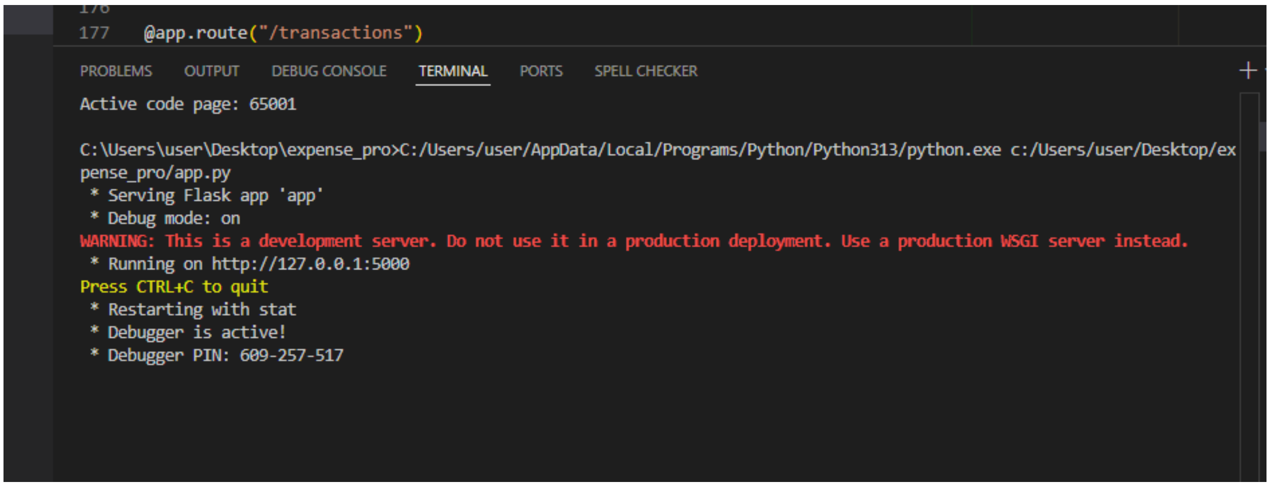
```
def user_exists(username):
    with open(USERS_FILE, newline="") as f:
        reader = csv.DictReader(f)
        for r in reader:
            if r["username"] == username:
                return True
    return False


def create_user(username, password):
    h = generate_password_hash(password)
    with open(USERS_FILE, "a", newline="") as f:
        writer = csv.writer(f)
        writer.writerow([username, h])


def validate_user(username, password):
    with open(USERS_FILE, newline="") as f:
        reader = csv.DictReader(f)
        for r in reader:
            if r["username"] == username and check_password_hash(r["password_hash"],
password):
                return True
    return False


def add_transaction(username, ttype, date, category, amount, description):
    tid = str(uuid.uuid4())
    with open(TX_FILE, "a", newline="") as f:
        writer = csv.writer(f)
        writer.writerow([tid, username, ttype, date, category, f"{float(amount):.2f}",
description])
    return tid
```

```
def read_transactions(username=None):
    rows = []
    with open(TX_FILE, newline="") as f:
        reader = csv.DictReader(f)
        for r in reader:
            if username is None or r["username"] == username:
                rows.append(r)
    return rows

def write_all_transactions(rows):
    with open(TX_FILE, "w", newline="") as f:
        fieldnames = ["id","username","type","date","category","amount","description"]
        writer = csv.DictWriter(f, fieldnames=fieldnames)
        writer.writeheader()
        for r in rows:
            writer.writerow(r)

def get_user_summary(username):
    txs = read_transactions(username)
    income = sum(float(t["amount"]) for t in txs if t["type"]=="income")
    expense = sum(float(t["amount"]) for t in txs if t["type"]=="expense")
    return {"income": income, "expense": expense, "balance": income-expense}

def monthly_aggregation(username):
    txs = read_transactions(username)
    months = {}
    for t in txs:
        d = t["date"]
        if not d: continue
        key = d[:7]  # YYYY-MM
        months.setdefault(key, {"income":0.0,"expense":0.0})
        if t["type"]=="income":
            months[key]["income"] += float(t["amount"])
        else:
            months[key]["expense"] += float(t["amount"])
    # sort by month
    items = sorted(months.items())
    labels = [k for k,_ in items]
    inc = [v["income"] for _,v in items]
    exp = [v["expense"] for _,v in items]
    return {"labels": labels, "income": inc, "expense": exp}

def category_breakdown(username):
    txs = read_transactions(username)
    cats = {}
    for t in txs:
        if t["type"]=="expense":
            cats.setdefault(t["category"],0.0)
            cats[t["category"]] += float(t["amount"])
    labels = list(cats.keys())
    values = [cats[k] for k in labels]
    return {"labels": labels, "values": values}

# Routes
@app.route("/")
def home():
    if "user" in session:
        return redirect(url_for("dashboard"))
    return render_template("login.html")

@app.route("/register", methods=["GET","POST"])
def register():
    if request.method=="POST":
        u = request.form.get("username").strip()
        p = request.form.get("password").strip()
        if not u or not p:
            flash("Provide username and password","error")
            return redirect(url_for("register"))
        if user_exists(u):
            flash("Username already exists","error")
            return redirect(url_for("register"))
        create_user(u,p)
        flash("Account created. Please login.","success")
        return redirect(url_for("home"))
    return render_template("register.html")

@app.route("/login", methods=["POST"])
def login():
    u = request.form.get("username").strip()
    p = request.form.get("password").strip()
    if validate_user(u,p):
        session["user"] = u
        return redirect(url_for("dashboard"))
    flash("Invalid credentials","error")
    return redirect(url_for("home"))
```

```
@app.route("/logout")
def logout():
    session.clear()
    return redirect(url_for("home"))

@app.route("/dashboard")
def dashboard():
    if "user" not in session:
        return redirect(url_for("home"))
    username = session["user"]
    summary = get_user_summary(username)
    recent = sorted(read_transactions(username), key=lambda x: x["date"], reverse=True)[:6]
    return render_template("dashboard.html", username=username, summary=summary,
recent=recent)

@app.route("/add", methods=["GET","POST"])
def add():
    if "user" not in session:
        return redirect(url_for("home"))
    if request.method=="POST":
        ttype = request.form.get("type")
        date = request.form.get("date") or datetime.now().strftime("%Y-%m-%d")
        category = request.form.get("category") or "Other"
        amount = request.form.get("amount") or "0"
        description = request.form.get("description") or ""
        try:
            float(amount)
        except:
            flash("Invalid amount", "error")
            return redirect(url_for("add"))
        add_transaction(session["user"], ttype, date, category, amount, description)
        flash("Transaction added", "success")
        return redirect(url_for("dashboard"))
    return render_template("add_transaction.html", categories=CATEGORIES)

@app.route("/transactions")
def transactions():
    if "user" not in session:
        return redirect(url_for("home"))
    rows = sorted(read_transactions(session["user"]), key=lambda x: x["date"], reverse=True)
    return render_template("view_transactions.html", rows=rows)

@app.route("/delete/<tid>", methods=["POST"])
def delete_transaction(tid):
    if "user" not in session:
        return redirect(url_for("home"))
    rows = read_transactions()
    rows = [r for r in rows if r["id"]!=tid or r["username"]!=session["user"]]
    write_all_transactions(rows)
    flash("Deleted", "success")
    return redirect(url_for("transactions"))

@app.route("/edit/<tid>", methods=["GET","POST"])
def edit_transaction(tid):
    if "user" not in session:
        return redirect(url_for("home"))
    rows = read_transactions()
    target = None
    for r in rows:
        if r["id"]==tid and r["username"]==session["user"]:
            target = r
            break
    if not target:
        flash("Not found","error")
        return redirect(url_for("transactions"))
    if request.method=="POST":
        target["date"] = request.form.get("date") or target["date"]
        target["category"] = request.form.get("category") or target["category"]
        target["amount"] = f"{float(request.form.get('amount')):.2f}"
        target["description"] = request.form.get("description") or ""
        write_all_transactions(rows)
        flash("Updated","success")
        return redirect(url_for("transactions"))
    return render_template("add_transaction.html", edit=target, categories=CATEGORIES)

@app.route("/analytics")
def analytics():
    if "user" not in session:
        return redirect(url_for("home"))
    return render_template("analytics.html")

@app.route("/api/monthly")
def api_monthly():
    if "user" not in session:
        return jsonify({"error":"unauth"}), 401
    data = monthly_aggregation(session["user"])
    return jsonify(data)
```

```
@app.route("/api/categories")
def api_categories():
    if "user" not in session:
        return jsonify({"error":"unauth"}), 401
    data = category_breakdown(session["user"])
    return jsonify(data)

@app.route("/profile")
def profile():
    if "user" not in session:
        return redirect(url_for("home"))
    return render_template("profile.html", username=session["user"])

@app.route("/export")
def export_csv():
    if "user" not in session:
        return redirect(url_for("home"))
    rows = read_transactions(session["user"])
    si = StringIO()
    writer = csv.writer(si)
    writer.writerow(["id","date","type","category","amount","description"])
    for r in rows:
        writer.writerow([r["id"], r["date"], r["type"], r["category"], r["amount"],
r["description"]])
    mem = StringIO(si.getvalue())
    mem.seek(0)
    return send_file(
        mem,
        mimetype='text/csv',
        as_attachment=True,
        download_name=f"{session['user']}_transactions.csv"
    )


if __name__ == "__main__":
    app.run(debug=True)
```

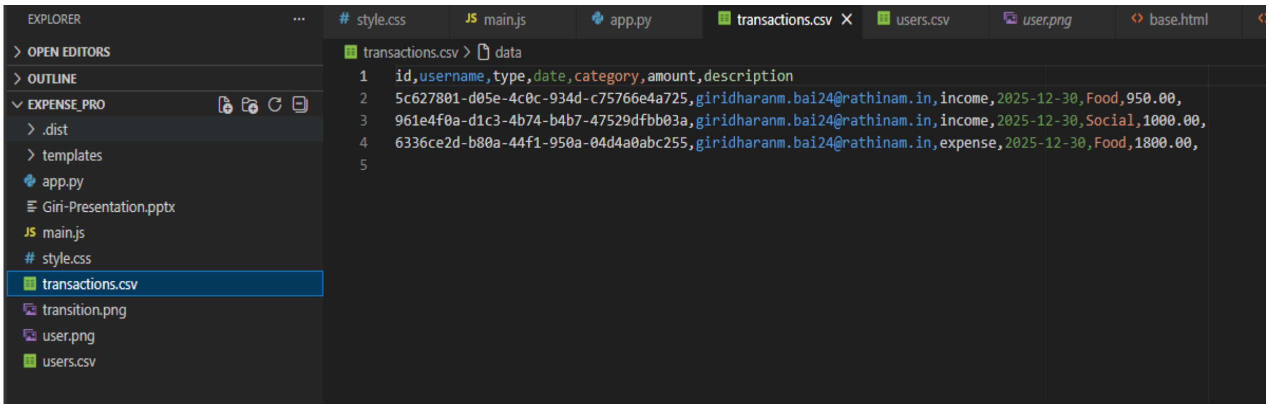
Output:



```
@app.route("/transactions")
```

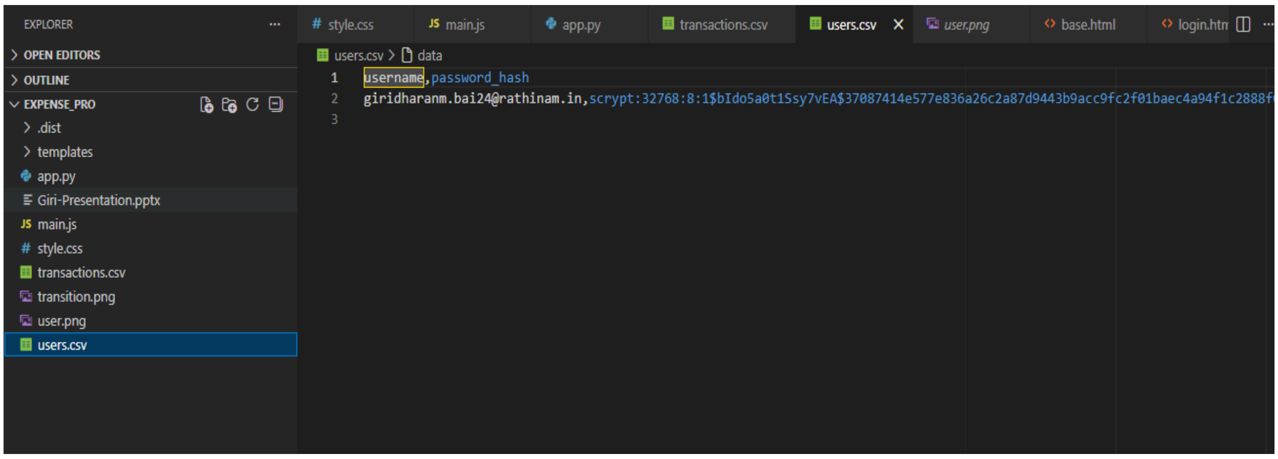
```
Active code page: 65001

C:\Users\user\Desktop\expense_pro>C:/Users/user/AppData/Local/Programs/Python/Python313/python.exe c:/Users/user/Desktop/ex
pense_pro/app.py
 * Serving Flask app 'app'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 609-257-517
```

transactions.csv:



```
id,username,type,date,category,amount,description
5c627801-d05e-4c0c-934d-c75766e4a725,giridharanm.bai24@rathinam.in,income,2025-12-30,Food,950.00,
961e4f0a-d1c3-4b74-b4b7-47529dfbb03a,giridharanm.bai24@rathinam.in,income,2025-12-30,Social,1000.00,
6336ce2d-b80a-44f1-950a-04d4a0abc255,giridharanm.bai24@rathinam.in,expense,2025-12-30,Food,1800.00,
```

User.CSV:



```
username,password_hash
giridharanm.bai24@rathinam.in,scrypt:32768:8:1$bIdo5a0t1Ssy7vEA$37087414e577e836a26c2a87d9443b9acc9fc2f01baec4a94f1c2888f
```

Benefits of the Expense Manager System :

1. **Easy Expense Tracking**
   Users can quickly add, edit, and view daily expenses in one place.
2. **Improves Financial Awareness**
   Helps users understand where their money is being spent.
3. **Time-Saving**
   Automates calculations and categorization, reducing manual work.
4. **Budget Management**
   Users can set budget limits and track spending against those limits.
5. **User-Friendly Interface**
   Simple design makes it easy for anyone to use, even beginners.
6. **Data Stored Safely (CSV / File Storage)**
   Expense data is securely stored in files for later access.
7. **Better Decision Making**
   Analytics module helps understand spending trends and patterns.
8. **Portable & Lightweight**
   Runs using simple Python without heavy databases.
9. **Search & Filter Features**
   Users can quickly find specific transactions based on date/category.
10. **Scalable System**
    New features can be easily added (charts, login, cloud storage, etc.)

Conclusion:

The Expense Manager System provides a simple, efficient, and reliable solution for managing personal finances. By allowing users to record income, categorize expenses, and analyze monthly spending patterns, the system helps individuals develop better financial discipline. Its user-friendly interface, modular architecture, and secure file-based storage make it accessible for beginners while still being powerful enough for practical use. Overall, the system successfully achieves its objective of offering a smart, organized, and scalable platform for tracking and understanding daily financial activities. It also lays a strong foundation for future enhancements such as cloud integration, advanced analytics, mobile support, and real-time notifications.