LAB 5 – THIẾT KẾ KIẾN TRÚC HỆ THỐNG

1. Thông tin nhóm

- **Tên nhóm:** Nhóm 1
- Danh sách thành viên

Họ và tên	**MSSV**	**Vai trò trong Lab 2**
Lê Đăng Khoa	PY00180	Mô tả tổng quan kiến trúc và lý do chọn.
Trần Văn Minh	PY00202	Phân tích các tầng/thành phần
Nguyễn Ngọc Văn	PY00295	Nhận xét kiến trúc.
Thái Quốc Việt	PY00294	Vẽ sơ đồ và giải thích tóm tắt

2. Tên kiến trúc hệ thống

2.1. Mô tả tổng quan

Ứng dụng Quản lý chi tiêu cá nhân được thiết kế theo mô hình kiến trúc hệ thống MVC (Model – View – Controller). MVC (Model – View – Controller) là một mô hình kiến trúc phần mềm dùng để tách biệt các phần chức năng trong một ứng dụng thành ba tầng chính:

- Model (M)
- View (V)
- Controller (C)

2.2. Lý do lựa chọn MVC cho ứng dụng Quản lý chi tiêu cá nhân

a. Tách biệt rõ ràng giữa giao diện và logic

Ứng dụng quản lý chi tiêu thường có nhiều màn hình (thêm khoản chi, xem thống kê, báo cáo biểu đồ...). Nhờ MVC, giao diện (View) tách riêng với phần xử lý dữ liệu (Model) và điều hướng (Controller). Điều này giúp:

- Dễ thay đổi giao diện mà không ảnh hưởng đến logic.

- Dễ mở rộng tính năng mà không phá vỡ cấu trúc tổng thể.

b. Dê bảo trì và mở rông

Khi cần thêm tính năng (ví dụ: thêm bộ lọc chi tiêu theo danh mục hoặc xuất báo cáo ra Excel), chỉ cần thêm Model và Controller liên quan mà không phải viết lại toàn bộ ứng dụng.

c. Tăng tính tái sử dụng và giảm trùng lặp

Model có thể được tái sử dụng cho nhiều View khác nhau (ví dụ: cùng dữ liệu chi tiêu nhưng có thể hiển thị ở dạng bảng hoặc biểu đồ).

d. Dễ dàng kiểm thử (Testing)

Có thể test riêng từng phaần:

- Test Model để kiểm tra logic tính toán.
- Test Controller để đảm bảo luồng xử lý đúng.
- Test View về khả năng hiển thị.

e. Phù hợp với ứng dụng có dữ liệu thay đổi liên tục

Quản lý chi tiêu thường xuyên thêm/xóa/sửa dữ liệu. MVC giúp cập nhật View tự động khi dữ liệu thay đổi (nếu áp dụng thêm cơ chế Observer hoặc Binding).

3. Phân tích các thành phần

3.1. Tên các thành phần

- **Model:** Đại diện cho miền dữ liệu và các quy tắc nghiệp vụ của ứng dụng.
- **View:** Chịu trách nhiệm hiển thị thông tin, giao tiếp trực tiếp với người dùng
- **Controller:** Đóng vai trò trung gian điều phối tiếp nhận yêu cầu từ người dùng và gọi các xử lý thích hợp trong **Model** trước khi trả kết quả cho **View.**

3.2. Vai trò của các thành phần

- **Vai trò của Model:** Model là nơi quản lý dữ liệu cốt lõi của ứng dụng, bao gồm các thực thể như User (người dùng), Category (danh mục thu/chi), Account (tài khoản), Transaction (giao dịch thu chi) và Budget (ngân sách). Tầng này chịu trách nhiệm định nghĩa cấu trúc dữ liệu, ràng buộc tính hợp lệ, thực hiện truy xuất và thao tác dữ liệu thông qua các lớp

DAO/Repository, đồng thời có thể tích hợp các nghiệp vụ quan trọng như tính tổng thu – chi, kiểm tra ngân sách vượt giới hạn, hay cập nhật số dư tài khoản.

- **Vai trò của View:** View chịu trách nhiệm trình bày dữ liệu một cách trực quan và dễ hiểu, thông qua các biểu mẫu nhập liệu, bảng thống kê, biểu đồ báo cáo thu – chi hay giao diện tổng quan (dashboard). Tầng này không chứa logic nghiệp vụ mà chỉ nhận dữ liệu đã được Controller chuẩn bị (ViewModel hoặc DTO) và hiển thị dưới dạng giao diện web hoặc trả về cấu trúc JSON cho ứng dụng di động.
- **Vai trò của Controller:** Controller là "cầu nối" giữa người dùng và hệ thống. Nó tiếp nhận các yêu cầu (HTTP request) từ giao diện, thực hiện kiểm tra dữ liệu đầu vào, gọi các phương thức xử lý nghiệp vụ trong Model, sau đó chọn View hoặc định dạng dữ liệu trả về (HTML hoặc JSON). Controller cũng đảm nhận vai trò xác thực, phân quyền truy cập và xử lý lỗi cơ bản trước khi gửi kết quả cho người dùng.

3.3. Mối quan hệ giữa các thành phần

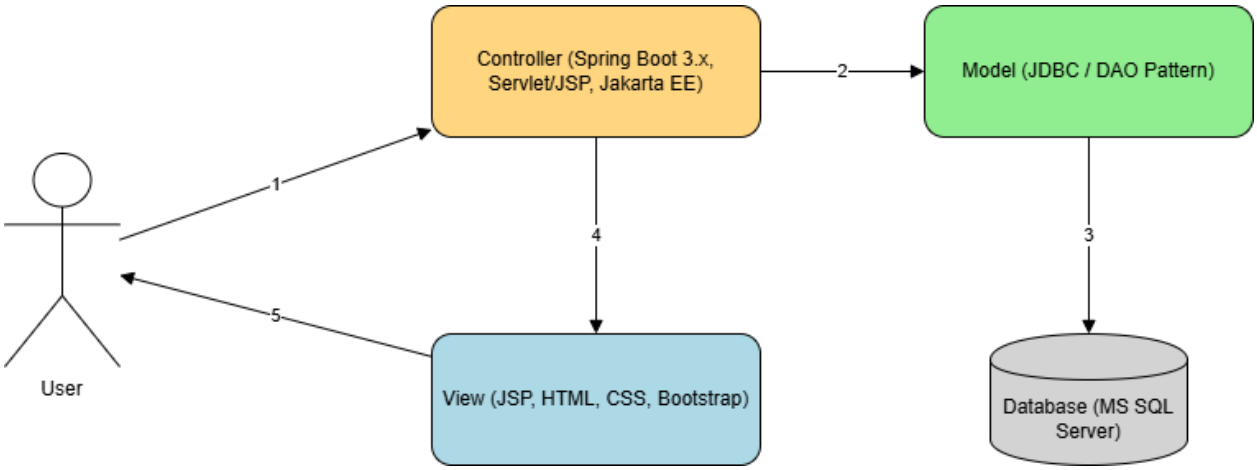
- **Model – View:** Model không giao tiếp trực tiếp với View. Thay vào đó, dữ liệu từ Model được truyền tới View thông qua Controller. Controller sẽ chuyển đổi dữ liệu thô (Entity) thành các đối tượng hiển thị phù hợp (ViewModel/DTO), giúp tách biệt hoàn toàn logic nghiệp vụ và phần giao diện.
- **Model – Controller:** Controller là thành phần chủ động gọi Model. Nó yêu cầu Model thực hiện các thao tác như thêm mới giao dịch, cập nhật ngân sách, truy vấn báo cáo thống kê hoặc xác thực thông tin người dùng. Model trả về kết quả xử lý cho Controller dưới dạng đối tượng hoặc dữ liệu đã tính toán. Sự tách biệt này giúp Controller không phụ thuộc vào cách dữ liệu được lưu trữ hay xử lý bên trong Model.
- **View – Controller:** View và Controller có mối quan hệ hai chiều. Khi người dùng thao tác (gửi form, click liên kết, lọc dữ liệu), View gửi yêu cầu về Controller. Ngược lại, Controller sau khi xử lý nghiệp vụ sẽ lựa chọn View phù hợp và cung cấp dữ liệu cần hiển thị. Điều này đảm bảo luồng thông tin luôn đi qua Controller, tránh việc View thao tác trực tiếp lên dữ liệu của Model.

Tóm lại

Việc áp dụng mô hình **MVC** trong hệ thống quản lý chi tiêu cá nhân giúp tách biệt rõ ràng **giao diện – nghiệp vụ – dữ liệu**, làm cho ứng dụng dễ bảo trì, dễ mở rộng, đồng thời có thể tái sử dụng cùng một nền tảng Model và Controller cho cả **web app** lẫn **ứng dụng di động** thông qua các API.

4. Sơ đồ kiến trúc

4.1. Hình vẽ sơ đồ



Hình 1. Sơ đồ MVC trong Ứng dụng Quản lý chi tiêu cá nhân

4.2. Giải thích tóm tắt

Bước 1 — Người dùng gửi yêu cầu (Request) Người dùng thao tác trên giao diện (View), ví dụ: Thêm một khoản chi tiêu mới. Xem báo cáo chi tiêu theo danh mục hoặc theo tháng. View sẽ gửi yêu cầu này đến Controller.

Bước 2 — Controller xử lý yêu cầu Controller nhận dữ liệu từ View, kiểm tra tính hợp lệ (validation) và ra quyết định: Nếu hợp lệ, Controller gọi Model để truy xuất hoặc cập nhật dữ liệu trong Database. Ví dụ: khi người dùng thêm chi tiêu, Controller sẽ gửi thông tin này cho Model để lưu vào bảng Expenses.

Bước 3 — Model làm việc với Database Model thực hiện các thao tác nghiệp vụ: Ghi dữ liệu mới vào cơ sở dữ liệu. Tính toán, lọc dữ liệu theo yêu cầu (ví dụ: tổng chi tiêu tháng 9). Trả kết quả đã xử lý về cho Controller.

Bước 4 — Controller gửi dữ liệu cho View Controller nhận kết quả từ Model (ví dụ: danh sách chi tiêu, tổng số tiền đã chi) rồi chuẩn bị dữ liệu theo định dạng mà View có thể hiển thị (ViewModel).

Bước 5 — View hiển thị kết quả cho người dùng View cập nhật giao diện theo dữ liệu mà Controller cung cấp, ví dụ: Hiển thị thông báo “Thêm chi tiêu thành công”. Vẽ biểu đồ thống kê chi tiêu theo danh mục.

5. Nhận xét

5.1. Điểm mạnh

Việc áp dụng mô hình MVC vào ứng dụng Quản lý chi tiêu cá nhân đã chứng minh hiệu quả rõ rệt trong quá trình phát triển:

- **Kiến trúc gọn gàng, dễ quản lý:** mã nguồn được phân chia hợp lý, dễ tìm kiếm và chỉnh sửa khi bổ sung chức năng.
- **Hỗ trợ làm việc nhóm:** nhóm có thể phân chia công việc theo từng tầng, giảm chồng chéo và xung đột khi lập trình.
- **Dễ dàng mở rộng:** sau khi xây dựng nền tảng ban đầu, việc thêm chức năng mới (như thống kê nâng cao, xuất báo cáo) không gây ảnh hưởng lớn đến hệ thống.
- **Thuận lợi cho bảo trì:** phát hiện và sửa lỗi nhanh hơn do biết chính xác tầng nào chịu trách nhiệm.

5.2. Hạn chế

- **Cấu trúc ban đầu phức tạp:** khi khởi tạo dự án phải thiết kế rõ ràng các lớp Model, Controller, View; mất thời gian hơn so với kiểu viết trực tiếp.
- **Đòi hỏi tuân thủ nguyên tắc:** nếu thành viên mới chưa quen mô hình, dễ viết sai (như đưa logic vào View hoặc Controller quá nhiều).
- **Chi phí học tập ban đầu:** với sinh viên hoặc người mới, cần thêm thời gian để hiểu cách hoạt động của MVC và tổ chức luồng dữ liệu hợp lý.
- **Luồng xử lý qua nhiều bước:** dữ liệu phải đi qua Controller trước khi đến View, đôi khi làm tăng độ phức tạp cho các chức năng nhỏ.