Báo Cáo Cải Tiến - Phiên Bản 3.0

1. **Nhược Điểm Của Phiên Bản Cũ (v2.0)**
   - **Cấu trúc chưa rõ ràng**: Frontend và Backend được gộp chung, logic quan trọng được viết trực tiếp trong phần view, làm cho mã nguồn trở nên khó bảo trì.
   - **Lưu trữ dữ liệu không tối ưu**: Dữ liệu được lưu trên local storage của người dùng, sử dụng ngôn ngữ Javacript dữ liệu sẽ không đảm bảo tính an toàn và nhất quán (không có ràng buộc kiểu dữ liệu rõ ràng…).
   - **Thiếu tổ chức trong mã nguồn**: Không chia tách rõ ràng các thành phần, khiến việc kiểm thử trở nên khó khăn do các logic được viết lẫn lộn, không có sự phân biệt giữa giao diện, xử lý nghiệp vụ và truy vấn dữ liệu.
2. **Cải Tiến Trong Phiên Bản v3.0**

   Phiên bản 3.0 đã có nhiều thay đổi quan trọng nhằm cải thiện cấu trúc và khả năng mở rộng của hệ thống:

   2.1. Tách biệt Frontend và Backend
   - **Frontend:**
     - Chịu trách nhiệm hiển thị dữ liệu, xử lý giao diện và nhận tương tác từ người dùng.
     - Không chứa logic xử lý, chỉ gửi API về Backend để lấy hoặc cập nhật dữ liệu.
     - Sử dụng **Next.js** kết hợp với **Shadcn UI** để tạo giao diện hiện đại và dễ sử dụng.
     - **Redux** được tích hợp để quản lý state và gọi API hiệu quả hơn.
     - Giao diện được thiết kế trực quan, dễ mở rộng và bảo trì.
   - **Backend:**
     - Chứa toàn bộ logic xử lý nghiệp vụ và thao tác dữ liệu.
     - Dữ liệu được lưu trên **PostgreSQL** chạy trên **Docker**, đảm bảo tính ổn định và an toàn.
     - Áp dụng kiến trúc **Clean Architecture**, chia thành các lớp **Model → Usecase → Repository → Handlers**, giúp phân tách rõ ràng vai trò của từng thành phần và dễ dàng kiểm thử.

   2.2. Kiến trúc Clean Architecture trong Backend
   - **Handlers (Giao tiếp với API - Controller)**: Xử lý request từ client, gọi các usecase phù hợp.
   - **Usecase (Xử lý nghiệp vụ)**: Thực hiện các logic nghiệp vụ như tạo, cập nhật, xóa dữ liệu.
   - **Repository (Tầng truy xuất dữ liệu)**: Giao tiếp với database, thực hiện các truy vấn cần thiết.
   - **Model (Định nghĩa kiểu dữ liệu)**: Chứa các định nghĩa cấu trúc dữ liệu của hệ thống và không phụ thuộc và thư viện bên ngoài.
   - **Mocks (Giả lập dữ liệu để kiểm thử)**: Hỗ trợ testing bằng cách mô phỏng dữ liệu và chức năng của repository, usecase hoặc handlers.
   - **Ưu điểm của Clean Architecture:**
     - **Dễ bảo trì**: Mỗi thành phần có một vai trò rõ ràng, dễ mở rộng.
     - **Dễ kiểm thử**: Có thể viết unit test riêng cho từng phần usecase, repository và handlers.
     - **Tái sử dụng cao**: Các module có thể dễ dàng tái sử dụng cho các dự án khác.
3. **Kiểm Thử Và Đánh Giá**

   Việc kiểm thử được chia thành **Test Unit** và **Test Coverage**.

   3.1. Các module được kiểm thử chính
   - Các thành phần quan trọng được kiểm thử bao gồm:
     - Status (Trạng thái sinh viên)
     - Program (Chương trình đào tạo)
     - Faculty (Khoa)

    - Student (Sinh viên)
- Mỗi module đều được kiểm thử trên 3 lớp chính:
    - Usecase (Nghiệp vụ)
    - Repository (Dữ liệu)
    - Handlers (Giao tiếp API)
- Lưu ý: vì repository được kết nối với cơ sở dữ liệu thực tên phần test sẽ sử dụng mock repository thay thế để giả sử trả về kiểu dữ liệu và thông báo lỗi.

3.2. Kết quả Test Coverage

Kết quả kiểm thử được thống kê như sau (chi tiết xem trong **coverage.html**):

Module	Thành phần	Coverage (%)
Faculty	Handlers	**100%**
Faculty	Repository Mock	**100%**
Faculty	Usecase	**89.5%**
Program	Handlers	**100%**
Program	Repository Mock	**100%**
Program	Usecase	**88.2%**
Status	Handlers	**100%**
Status	Repository Mock	**100%**
Status	Usecase	**89.5%**
Student	Handlers	**96.3%**
Student	Repository Mock	**100%**
Student	Usecase	**85.3%**

Nhận xét:

- Hầu hết các thành phần quan trọng đều đạt mức **test coverage cao**, đặc biệt là handlers và repository mock.
- Một số usecase chưa đạt mức 90%, có thể cần bổ sung thêm test case để đảm bảo độ bao phủ toàn diện hơn.
- Tổng thể, việc kiểm thử trong phiên bản 3.0 đã giúp hệ thống **đáng tin cậy hơn**, dễ bảo trì và mở rộng trong tương lai.