Tìm hiểu về Unit Test Coverage và Best practices

MSSV – HỌ TÊN:
22127104 – Phạm Thị Mỹ Hạnh
22127408 – Kha Vĩnh Thuận

1. Các loại Code coverage quan trọng

Unit test coverage là một thước đo để đánh giá mức độ mà mã nguồn được kiểm thử bởi các bài kiểm tra đơn vị (unit tests). Dưới đây là các loại coverage quan trọng:

- Line coverage (Độ bao phủ dòng):
  - Đo lường tỷ lệ phần trăm các dòng code được thực thi trong quá trình chạy unit test.
  - Ví dụ: Nếu một hàm có 10 dòng và unit test thực thi 8 dòng, thì Line Coverage là 80%.
  - Đảm bảo rằng phần lớn mã nguồn đã được chạy qua ít nhất một lần, nhưng không đảm bảo tất cả logic được kiểm tra.
- Branch Coverage (Độ bao phủ nhánh):
  - Tập trung vào việc kiểm tra tất cả các nhánh logic trong mã nguồn, chẳng hạn như if, else, hoặc switch.
  - Đảm bảo logic phân nhánh được kiểm tra đầy đủ, phát hiện lỗi trong các điều kiện.
  - Ví dụ: Nếu một hàm có câu lệnh if (age > 18) với hai nhánh (true và false), thì cần ít nhất hai test case để đạt 100% Branch Coverage.
- Function Coverage (Độ bao phủ hàm):
  - Xác minh rằng mỗi hàm hoặc phương thức trong mã nguồn được gọi ít nhất một lần trong quá trình kiểm thử.
  - Đảm bảo không có hàm nào bị bỏ sót, nhưng không kiểm tra sâu vào logic bên trong.
  - Ví dụ: Nếu ứng dụng có 5 hàm (addStudent, deleteStudent, v.v.), tất cả phải được gọi trong unit test để đạt 100% Function Coverage.
- Path Coverage (Độ bao phủ đường đi):
  - Đảm bảo tất cả các đường đi logic có thể xảy ra trong mã nguồn đều được kiểm tra.
  - Đây là mức coverage kỹ lưỡng nhất, nhưng phức tạp và thường không khả thi với mã nguồn lớn.

    - Ví dụ: Với hai điều kiện if (age > 18 && grade > 5), có 4 đường đi logic (true-true, true-false, false-true, false-false). Path Coverage yêu cầu kiểm tra tất cả 4 trường hợp.

2. **Mức coverage tối thiểu:** trong ngành công nghiệp phần mềm, mức độ code coverage phù hợp phụ thuộc vào loại ứng dụng, độ quan trọng của hệ thống, và tiêu chuẩn trong ngành. Dưới đây là một số tham chiếu từ industry:

- 70-80% Line Coverage: Được coi là mức coverage tốt, đảm bảo rằng phần lớn code được kiểm thử mà không tốn quá nhiều tài nguyên. Tập trung vào việc kiểm thử các logic quan trọng.
- 100% Coverage: Đây là mức coverage lý tưởng, nhưng không phải lúc nào cũng khả thi hoặc cần thiết, vì nó tốn nhiều thời gian và tài nguyên. Trong các ngành mà lỗi có thể gây hậu quả nghiêm trọng như tài chính hoặc y tế thì cần mức coverage cao gần như 100%.

3. **Best practices khi viết unit test**

- Viết unit test dễ bảo trì, không phụ thuộc vào database hoặc persistent storage:
    - Unit test nên tập trung vào logic của code và không phụ thuộc vào cơ sở dữ liệu hoặc các hệ thống lưu trữ khác. Sử dụng mock hoặc stub để thay thế cho các phụ thuộc này khi cần thiết.
- Kiểm thử đầy đủ Happy case và Edge case:
    - Happy case: là trường hợp lí tưởng, đảm bảo các chức năng hoạt động đúng với bộ dữ liệu bình thường(đúng).
        - Ví dụ: Tìm kiếm sinh viên với mã số sinh viên và tên khoa(studentId=”1”, faculty=”Khoa công nghệ thông tin”)
    - Edge case: là trường hợp biên, kiểm tra với dữ liệu không hợp lệ hoặc bất thường và đảm bảo trả về thông báo lỗi cho người dùng.
        - Ví dụ: Tìm kiếm sinh viên với mã số sinh viên rỗng hay tên khoa không nằm trong các khoa hiện đang có.
- Không viết test trùng lặp hoặc quá phụ thuộc vào implementation details:
    - Viết các test case rõ ràng, ngắn gọn. Không viết nhiều test kiểm tra cùng một logic theo cách khác nhau, gây lãng phí.
    - Tránh việc test quá chi tiết vào cách thức triển khai cụ thể của code, thay vào đó tập trung vào đầu vào và đầu ra mong đợi.
    - Ví dụ: Với hàm calculateAverageGrade(), chỉ cần kiểm tra kết quả đầu ra đúng, không cần quan tâm nó dùng phương thức gì để kiểm tra.
- Tự động hóa: Sử dụng các công cụ để chạy test tự động và tích hợp CI/CD pipeline như Jenkins để kiểm tra mã nguồn thường xuyên.

4. **Đánh giá và cải tiến thiết kế**

- Phụ thuộc quá nhiều vào database: các phương thức phụ thuộc trực tiếp vào MongoDB thông qua studentModel, facultyModel,… và các truy vấn Mongoose như findOne, find, populate, exec nên gây khó khăn khi test vì cần mock các truy vấn như populate, exec để giả lập theo các câu lệnh trong database.