

TRƯỜNG ĐẠI HỌC SÀI GÒN
KHOA CÔNG NGHỆ THÔNG TIN



KIỂM THỬ PHẦN MỀM

Bài Tập Lớn - Kiểm Thử Phần Mềm

Ứng dụng Đăng nhập & Quản lý Sản phẩm
(Version 1.0)

GVHD: Từ Lăng Phiêu

Contents

1	Giới thiệu dự án	1
1.1	Tổng quan	1
1.2	Công nghệ sử dụng	1
1.2.1	Frontend	1
1.2.2	Backend	1
1.2.3	Testing	1
1.2.4	DevOps & Tools:	1
2	Phân tích và thiết kế test cases	1
2.1	Login — Phân tích và Test Scenarios	1
2.2	Product — Phân tích và Test Scenarios	6
3	Unit Testing và Test-Driven Development	11
3.1	Login - Unit Tests Frontend và Backend	11
3.1.1	Frontend Unit Tests - Validation Login	11
3.1.2	Backend Unit Tests - Login Service	11
3.2	Product - Unit Tests Frontend và Backend	15
3.2.1	Frontend Unit Tests - Product Validation	15
3.2.2	Backend Unit Tests - Product Service	16
4	Integration Testing	18
4.1	Login - Integration Testing	18
4.1.1	Frontend Component Integration	18
4.1.2	Backend API Integration	20
4.2	Product - Integration Testing	23
4.2.1	Frontend Component Integration	23
4.2.2	Backend API Integration	25
5	Mock Testing	27
5.1	Login - Mock Testing	27
5.1.1	Frontend Mocking	27
5.1.2	Backend Mocking	28
5.2	Product - Mock Testing	32
5.2.1	Frontend Mocking	32
5.2.2	Backend Mocking	35
6	Automation Testing và CI/CD	39
6.1	Login - E2E Automation Testing	39
6.1.1	Setup và Configuration	39
6.1.2	E2E Test Scenarios cho Login	42
6.1.3	CI/CD Integration cho Login Tests	44
6.2	Product - E2E Automation Testing	46
6.2.1	Setup Page Object Model	46
6.2.2	E2E Test Scenarios cho Product	50
6.2.3	CI/CD Integration	54
7	Phản Mở Rộng	55
7.1	Performance Testing	55
8	Test coverage	57
9	Test results	58

1 Giới thiệu dự án

1.1 Tổng quan

- Chức năng Login: Hệ thống đăng nhập với validation đầy đủ
- Chức năng Product: Quản lý sản phẩm (CRUD operations)
- Frontend: React 18+
- Backend: Spring Boot 3.2+
- Testing: Phát triển theo phương pháp TDD

1.2 Công nghệ sử dụng

1.2.1 Frontend

- ReactJS (v18+)
- Axios
- React Testing Library & Jest

1.2.2 Backend

- Java 17
- Spring Boot (v3.2+)
- Spring Data JPA
- Maven

1.2.3 Testing

- JUnit 5
- Mockito
- Cypress
- JaCoCo

1.2.4 DevOps & Tools:

- GitHub Actions
- Git

2 Phân tích và thiết kế test cases

2.1 Login — Phân tích và Test Scenarios

- a) Phân tích đầy đủ các yêu cầu chức năng của tính năng Login:
- Validation rules cho username:
 - Phải từ 3–50 ký tự
 - Không được bỏ trống
 - Không được có ký tự đặc biệt, khoảng trắng
 - Validation rules cho password:
 - Phải từ 6–100 ký tự
 - Phải gồm cả chữ và số
 - Authentication flow

- Bước 1: Người dùng nhập thông tin username và password, nhấn login. React Frontend gửi request POST api/auth/login chứa thông tin này đến Spring Boot Backend
- Bước 2: Backend nhận request, kiểm tra định dạng (Validate rules) của username và password. Nếu sai format thì sẽ trả về lỗi 400 Bad Request ngay lập tức
- Bước 3: AuthService gọi xuống Database (qua Repository) để tìm username. So sánh password của người dùng nhập (sau khi hash) với password hash trong cơ sở dữ liệu
- Bước 4: Nếu khớp thì hệ thống sẽ tạo một chuỗi Access Token (JWT) chứa thông tin user. Nếu không khớp thì hệ thống không sinh ra token và trả về phản hồi thất bại
- Bước 5: Backend trả về status 200 OK kèm theo Token và thông tin User
- Bước 6: Frontend nhận token lưu vào localStorage hoặc cookies và điều hướng người dùng sang trang Dashboard
- Error handling
 - Lỗi validation:
- Yêu cầu chức năng:
 - Validate username theo validation rules
 - Validate password theo validation rules
 - Thông báo lỗi nếu validate không thành công
 - Chuyển user sang trang chính khi thành công

b) Liệt kê và mô tả ít nhất 10 test scenarios cho Login bao gồm:

- Happy path: Đăng nhập thành công
- Negative tests: Username/password rỗng, sai format
- Boundary tests: Độ dài min/max của username/password
- Edge cases: Ký tự đặc biệt, khoảng trắng

STT	Test Scenario	Input	Preconditions	Expected	Mức độ ưu tiên
1	User đăng nhập thành công	Username đúng và mật khẩu đúng	Ứng dụng đang chạy	Hệ thống validate thành công và chuyển người dùng sang trang chủ	Critical
2	User nhập vào username rỗng	Username = “ ” và mật khẩu hợp lệ	Ứng dụng đang chạy	Hệ thống validate thất bại và thông báo lỗi username rỗng	High
3	User nhập vào password rỗng	Username hợp lệ và mật khẩu = “ ”	Ứng dụng đang chạy	Hệ thống validate thất bại và thông báo lỗi password rỗng	High
4	User nhập vào username không tồn tại	Username không tồn tại và mật khẩu hợp lệ	Ứng dụng đang chạy	Hệ thống validate thất bại khi không tìm được username, thông báo lỗi username không tồn tại	Critical
5	User nhập vào username quá ngắn	Username = “a” và mật khẩu hợp lệ	Ứng dụng đang chạy	Hệ thống validate thất bại, thông báo username phải từ 3 ký tự trở lên	Medium
6	User nhập vào username quá dài	Username trên 15 ký tự và mật khẩu hợp lệ	Ứng dụng đang chạy	Hệ thống validate thất bại, thông báo lỗi username quá dài	Medium
7	User nhập vào username với độ dài phù hợp	Username hợp lệ và mật khẩu hợp lệ	Ứng dụng đang chạy	Hệ thống validate chiều dài username. Chiều dài hợp lệ, tiếp tục validate tồn tại. Nếu tồn tại, chuyển sang validate mật khẩu	High
8	User nhập vào username chứa ký tự đặc biệt hoặc khoảng trắng	Username = “% \$” và mật khẩu đúng	Ứng dụng đang chạy	Hệ thống validate thất bại, thông báo username không được chứa khoảng trắng hoặc ký tự đặc biệt	Medium
9	User nhập vào mật khẩu quá ngắn	Username hợp lệ và mật khẩu = “1”	Ứng dụng đang chạy	Hệ thống validate thất bại, thông báo lỗi password quá ngắn	Medium
10	User nhập vào mật khẩu với độ dài phù hợp	Username hợp lệ và mật khẩu hợp lệ	Ứng dụng đang chạy	Hệ thống validate chiều dài mật khẩu. Chiều dài hợp lệ, tiếp tục validate trùng khớp	High
11	User nhập sai mật khẩu	Username đúng và mật khẩu	Ứng dụng đang chạy	Hệ thống validate thất bại khi password không trùng khớp, thông báo sai password	Critical
12	Username / mật khẩu chứa khoảng trắng ở đầu hoặc cuối	Username hợp lệ và mật khẩu hợp lệ	Ứng dụng đang chạy	Hệ thống sẽ lược bỏ (trim) khoảng trắng dư thừa ở đầu và cuối trước khi validate	Low
13	User đăng nhập thất bại 5 lần liên tiếp	Username đúng, mật khẩu sai	Ứng dụng đang chạy	Sau 5 lần sai mật khẩu liên tiếp, thông báo vượt ngưỡng, tạm khóa tài khoản	High

Table 1: Test scenarios cho chức năng đăng nhập

c) Phân loại test scenarios theo mức độ ưu tiên (Critical, High, Medium, Low) và giải thích

STT	Test Scenario	Ưu tiên	Giải thích
1	User đăng nhập thành công	Critical	Tính năng login hỏng hoàn toàn nếu user không thể login với thông tin đúng.
2	User nhập vào username rỗng	High	Lỗi phổ biến; hệ thống phải thông báo lỗi thay vì trả về bad request.
3	User nhập vào password rỗng	High	Lỗi phổ biến; hệ thống phải thông báo lỗi thay vì trả về bad request.
4	User nhập vào username không tồn tại	Critical	Nhận diện username không hợp lệ; thất bại có thể gây lỗi bảo mật / logic xác thực.
5	User nhập vào username quá ngắn	Low	Validation nhỏ; không ảnh hưởng logic chính, đảm bảo thống nhất format.
6	User nhập vào username quá dài	High	Ngăn user gửi input quá dài ảnh hưởng hiệu năng.
7	User nhập username với độ dài phù hợp	High	Bước trung gian quan trọng cho flow xác thực; fail ảnh hưởng truy cập.
8	User nhập username chứa ký tự đặc biệt hoặc khoảng trắng	Critical	Ngăn input độc hại / không hợp lệ; phòng injection.
9	User nhập vào mật khẩu quá ngắn	Low	Validation cơ bản; không ảnh hưởng logic chính.
10	User nhập mật khẩu với độ dài phù hợp	High	Đảm bảo cơ chế kiểm tra độ dài đúng; fail chặn user hợp lệ.
11	User nhập sai mật khẩu	Critical	Sai nhưng vẫn login → lỗ hổng bảo mật nghiêm trọng.
12	Username / mật khẩu có khoảng trắng đầu/cuối	Low	Chủ yếu vấn đề UX; có thể trim.
13	User đăng nhập thất bại 5 lần liên tiếp	High	Nguy hại bảo mật

Table 3: Phân loại mức độ ưu tiên (tô màu) và giải thích cho các test scenario đăng nhập

Test Case ID	TC_Login_001
Test Name	Đăng nhập thành công với credentials hợp lệ
Priority	Critical
Preconditions	<ul style="list-style-type: none"> Tài khoản user tồn tại Ứng dụng đang hoạt động
Test Steps	<ol style="list-style-type: none"> Truy cập trang login Nhập vào username Nhập vào password Bấm nút đăng nhập
Test Data	Username: Password:
Expected Result	Chuyển người dùng về trang chủ
Actual Result	
Status	

Table 5: TC_Login_001 — Đăng nhập thành công

Test Case ID	TC_Login_002
Test Name	Đăng nhập với username không tồn tại
Priority	Critical
Preconditions	<ul style="list-style-type: none"> Tài khoản user không tồn tại Ứng dụng đang hoạt động
Test Steps	<ol style="list-style-type: none"> Truy cập trang login Nhập vào username không tồn tại Nhập vào password Bấm nút đăng nhập
Test Data	Username: Password:
Expected Result	<ul style="list-style-type: none"> Không chuyển người dùng về trang chủ Hiện thị, thông báo lỗi "username không tồn tại" cho người dùng
Actual Result	
Status	

Table 7: TC_Login_002 — Username không tồn tại

Test Case ID	TC_Login_003
Test Name	Đăng nhập với username chứa khoảng trắng / ký tự đặc biệt
Priority	Critical
Preconditions	Ứng dụng đang hoạt động
Test Steps	<ol style="list-style-type: none"> Truy cập trang login Nhập vào username chứa khoảng trắng / ký tự đặc biệt Nhập vào password Bấm nút đăng nhập
Test Data	Username: Password:
Expected Result	<ul style="list-style-type: none"> Không chuyển người dùng về trang chủ Hiện thị, thông báo lỗi "username không đúng định dạng" cho người dùng
Actual Result	
Status	

Table 9: TC_Login_003 — Username chứa khoảng trắng / ký tự đặc biệt

Test Case ID	TC_Login_004
Test Name	Đăng nhập với username đúng, mật khẩu sai
Priority	Critical
Preconditions	<ul style="list-style-type: none"> • Username đúng • Ứng dụng đang hoạt động
Test Steps	<ol style="list-style-type: none"> 1. Truy cập trang login 2. Nhập vào username đúng 3. Nhập vào password sai 4. Bấm nút đăng nhập
Test Data	Username: Password:
Expected Result	<ul style="list-style-type: none"> • Không chuyển người dùng về trang chủ • Hiện thị, thông báo lỗi "sai tên đăng nhập hoặc mật khẩu" cho người dùng
Actual Result	
Status	

Table 11: TC_Login_004 — Username đúng, mật khẩu sai

Test Case ID	TC_Login_005
Test Name	Đăng nhập với username hoặc mật khẩu rỗng
Priority	High
Preconditions	<ul style="list-style-type: none"> • Username hoặc password rỗng • Ứng dụng đang hoạt động
Test Steps	<ol style="list-style-type: none"> 1. Truy cập trang login 2. Nhập vào username (có thể rỗng) 3. Nhập vào password (có thể rỗng) 4. Bấm nút đăng nhập
Test Data	Username: Password:
Expected Result	<ul style="list-style-type: none"> • Không chuyển người dùng về trang chủ • Hiện thị, thông báo lỗi "phải điền tất cả các mục" cho người dùng
Actual Result	
Status	

Table 13: TC_Login_005 — Username hoặc mật khẩu rỗng

2.2 Product — Phân tích và Test Scenarios

a) Phân tích đầy đủ các yêu cầu chức năng của Product CRUD

Create (Thêm sản phẩm mới)

Validation rules:

- Tên sản phẩm không được rỗng
- Tên sản phẩm phải từ 3–100 ký tự
- Giá sản phẩm phải > 0 và $\leq 999,999,999$
- Số lượng sản phẩm phải từ 0 đến 999,999,999
- Mô tả sản phẩm phải từ 0–500 ký tự
- Danh mục sản phẩm phải nằm trong tập tên danh mục sẵn có

Yêu cầu chức năng:

- Validate dữ liệu người dùng nhập vào
- Generate mã sản phẩm tự động
- Lưu thông tin vào cơ sở dữ liệu
- Thông báo khi thêm thành công
- Thông báo, hiển thị lỗi khi thất bại

Read (Xem danh sách / chi tiết sản phẩm)

Yêu cầu chức năng xem danh sách:

- Hiển thị đầy đủ, đúng thông tin tổng quát của mọi sản phẩm
- Cho phép người dùng xem chi tiết của mỗi sản phẩm
- Cho phép người dùng thêm / xóa / sửa sản phẩm

Yêu cầu chức năng xem chi tiết:

- Hiển thị đầy đủ, đúng thông tin chi tiết của mỗi sản phẩm
- Cho phép người dùng thêm / xóa / sửa chi tiết

Update (Cập nhật thông tin sản phẩm)

Validation rules (giống Create):

- Tên sản phẩm không được rỗng
- Tên sản phẩm phải từ 3–100 ký tự
- Giá sản phẩm phải > 0 và $\leq 999,999,999$
- Số lượng sản phẩm phải từ 0 đến 999,999,999
- Mô tả sản phẩm phải từ 0–500 ký tự
- Danh mục sản phẩm phải hợp lệ

Yêu cầu chức năng:

- Hiển thị thông tin hiện tại để người dùng chỉnh sửa
- Validate thông tin được chỉnh sửa
- Chỉ lưu phần thông tin thay đổi
- Lưu vào cơ sở dữ liệu
- Thông báo, cập nhật hiển thị khi thành công
- Thông báo, hiển thị lỗi khi thất bại

Delete (Xóa sản phẩm)

Yêu cầu chức năng:

- Kiểm tra tồn tại trước khi xóa
- Cho phép xóa một sản phẩm
- Xóa hẳn khỏi hệ thống
- Xác nhận trước khi xóa
- Thông báo, cập nhật giao diện khi xóa thành công
- Thông báo, hiển thị lỗi khi thất bại

b) Liệt kê và mô tả ít nhất 10 test scenarios cho Product bao gồm:

- Happy path: CRUD operations thành công
- Negative tests: Dữ liệu không hợp lệ
- Boundary tests: Giá trị min/max (tên, giá, số lượng, mô tả)
- Edge cases: Sản phẩm trùng tên, xóa sản phẩm không tồn tại

STT	Test Scenario	Input	Expected	Mức độ ưu tiên
1	Thêm / Xóa / Sửa / Xem một sản phẩm thành công	Thông tin sản phẩm hợp lệ	Hệ thống thực hiện thành công các CRUD operations, hiện thông báo thành công với mỗi chức năng	Critical
2	Cập nhật lại giao diện sau khi thêm / xóa / sửa	Thông tin sản phẩm hợp lệ	Giao diện danh sách/chi tiết được cập nhật sau thao tác thành công	Critical
3	Thêm sản phẩm với thông tin không hợp lệ	Dữ liệu không hợp lệ	Hệ thống báo lỗi; không tạo mới sản phẩm	High
4	Sửa sản phẩm với thông tin không hợp lệ	Dữ liệu không hợp lệ	Hệ thống báo lỗi; không sửa thông tin sản phẩm	High
5	Để trống tên sản phẩm khi thêm / sửa	Product Name trống, các trường khác hợp lệ	Báo lỗi tên sản phẩm không được rỗng; không thực hiện thêm / sửa	High
6	Tên sản phẩm dưới 3 ký tự khi thêm / sửa	Product Name < 3 ký tự (các trường khác hợp lệ)	Báo lỗi tên phải từ 3–100 ký tự; không thêm / sửa	Medium
7	Tên sản phẩm đúng với 100 ký tự khi thêm / sửa	Product Name = 100 ký tự (hợp lệ)	Validate thành công; thêm / sửa thành công; cập nhật giao diện	High
8	Mô tả quá dài (> 500 ký tự) khi thêm / sửa	Description > 500 ký tự	Báo lỗi mô tả phải ≤ 500 ký tự; không thêm / sửa	Medium
9	Xóa sản phẩm không tồn tại	Tên sản phẩm không tồn tại	Báo lỗi sản phẩm không tồn tại; không xóa	High
10	Sửa sản phẩm nhưng không có thay đổi	Không thay đổi trường nào	Thông báo không có thay đổi; không lưu update	High

Table 15: Phân loại mức độ ưu tiên các test scenario cho chức năng Product

c) Phân loại test scenarios theo mức độ ưu tiên và giải thích:

Test Case ID	TC_Product_001
Test Name	Tạo sản phẩm mới thành công
Priority	Critical
Preconditions	<ul style="list-style-type: none"> Ứng dụng đang hoạt động User đã đăng nhập
Test Steps	<ol style="list-style-type: none"> Truy cập trang login Thực hiện đăng nhập Nhập thông tin sản phẩm Bấm nút tạo sản phẩm
Test Data	Name = Laptop Dell Price = 15000000 Quantity = 10 Category = Electronics Description = '15-inch laptop'
Expected Result	<ul style="list-style-type: none"> Thêm thành công sản phẩm Thông báo thành công Làm mới danh sách
Actual Result	
Status	

Table 17: TC_Product_001 — Tạo sản phẩm mới thành công

Test Case ID	TC_Product_002
Test Name	Sửa sản phẩm mới thành công
Priority	Critical
Preconditions	<ul style="list-style-type: none"> • Ứng dụng đang hoạt động • User đã đăng nhập
Test Steps	<ol style="list-style-type: none"> 1. Truy cập trang login 2. Thực hiện đăng nhập 3. Chọn sửa sản phẩm 4. Sửa giá sản phẩm 5. Bấm nút sửa
Test Data	id = 1 new price = 14000000
Expected Result	<ul style="list-style-type: none"> • Cập nhật giá trị thành công • Thông báo thành công • Làm mới danh sách
Actual Result	
Status	

Table 19: TC_Product_002 — Sửa sản phẩm mới thành công

Test Case ID	TC_Product_003
Test Name	Xóa sản phẩm mới thành công
Priority	Critical
Preconditions	<ul style="list-style-type: none"> • Ứng dụng đang hoạt động • User đã đăng nhập
Test Steps	<ol style="list-style-type: none"> 1. Truy cập trang login 2. Thực hiện đăng nhập 3. Bấm nút xóa 4. Bấm nút xác nhận
Test Data	id = 1
Expected Result	<ul style="list-style-type: none"> • Xóa thành công sản phẩm • Thông báo thành công • Làm mới danh sách
Actual Result	
Status	

Table 21: TC_Product_003 — Xóa sản phẩm mới thành công

Test Case ID	TC_Product_004
Test Name	Xem danh sách, chi tiết sản phẩm
Priority	Critical
Preconditions	<ul style="list-style-type: none"> • Ứng dụng đang hoạt động • User đã đăng nhập
Test Steps	<ol style="list-style-type: none"> 1. Truy cập trang login 2. Thực hiện đăng nhập 3. Chọn sản phẩm id = 1 4. Nhấn nút xem chi tiết
Test Data	id = 1
Expected Result	Thông tin, thông tin chi tiết hiển thị đúng
Actual Result	
Status	

Table 23: TC_Product_004 — Xem danh sách, chi tiết sản phẩm

Test Case ID	TC_Product_005
Test Name	Tạo / Sửa sản phẩm với price <= 0
Priority	Critical
Preconditions	<ul style="list-style-type: none"> • Ứng dụng đang hoạt động • User đã đăng nhập
Test Steps	<ol style="list-style-type: none"> 1. Truy cập trang login 2. Thực hiện đăng nhập 3. Thêm/Sửa sản phẩm với price <= 0 4. Nhấn nút xem chi tiết
Test Data	price = 0
Expected Result	<ul style="list-style-type: none"> • Không thực hiện thêm / sửa • Hiện thị 'Giá phải > 0'
Actual Result	
Status	

Table 25: TC_Product_005 — Tạo / Sửa sản phẩm với price <= 0

3 Unit Testing và Test-Driven Development

3.1 Login - Unit Tests Frontend và Backend

3.1.1 Frontend Unit Tests - Validation Login

```
import { describe, it, expect } from 'vitest'
import { validateUsername, validatePassword } from '../utils/validateLogin.js'

describe('validateUsername', () => {
  it('accepts_valid', () => {
    expect(validateUsername('john_doe')).toBe(true)
  })
  it('rejects_short', () => {
    expect(validateUsername('ab')).not.toBe(true)
  })
  it('rejects_long', () => {
    expect(validateUsername('a'.repeat(51))).not.toBe(true)
  })
  it('rejects_special_characters', () => {
    expect(validateUsername('user@name')).not.toBe(true)
  })
  it('rejects_empty', () => {
    expect(validateUsername('')).not.toBe(true)
  })
})

describe('validatePassword', () => {
  it('accepts_valid', () => {
    expect(validatePassword('abc123')).toBe(true)
  })
  it('rejects_missing_number', () => {
    expect(validatePassword('abcdef')).not.toBe(true)
  })
  it('rejects_missing_letter', () => {
    expect(validatePassword('123456')).not.toBe(true)
  })
  it('rejects_short', () => {
    expect(validatePassword('abc')).not.toBe(true)
  })
  it('rejects_long', () => {
    expect(validatePassword('a'.repeat(101))).not.toBe(true)
  })
  it('rejects_empty', () => {
    expect(validatePassword('')).not.toBe(true)
  })
})
```

3.1.2 Backend Unit Tests - Login Service

a) Test method authenticate() với các scenarios

```
package com.sgu.login;

import java.util.Optional;

import static org.junit.jupiter.api.Assertions.assertNotNull;
import static org.junit.jupiter.api.Assertions.assertThrows;
import static org.junit.jupiter.api.Assertions.assertTrue;
import org.junit.jupiter.api.Test;
import org.mockito.Mockito;

import com.sgu.login.model.User;
```

```

import com.sgu.login.repository.UserRepository;
import com.sgu.login.service.AuthService;

public class AuthServiceTest {

    @Test
    void login_success() {
        var repo = Mockito.mock(UserRepository.class);
        Mockito.when(repo.findByUsername("admin"))
            .thenReturn(Optional.of(new User("admin", "Admin123")));
        var svc = new AuthService(repo);

        String token = svc.authenticate("admin", "Admin123");

        assertNotNull(token);
        assertTrue(token.startsWith("demo-token-"));
    }

    @Test
    void login_wrong_password() {
        var repo = Mockito.mock(UserRepository.class);
        Mockito.when(repo.findByUsername("admin"))
            .thenReturn(Optional.of(new User("admin", "Admin123")));
        var svc = new AuthService(repo);

        assertThrows(RuntimeException.class, () -> svc.authenticate("admin", "x
            "));
    }

    @Test
    void login_user_not_found() {
        var repo = Mockito.mock(UserRepository.class);
        Mockito.when(repo.findByUsername("ghost")).thenReturn(Optional.empty());
        ;
        var svc = new AuthService(repo);

        assertThrows(RuntimeException.class, () -> svc.authenticate("ghost", "
            abc"));
    }

    @Test
    void login_empty_username() {
        var repo = Mockito.mock(UserRepository.class);
        var svc = new AuthService(repo);

        assertThrows(IllegalArgumentException.class, () -> svc.authenticate("",
            "abc"));
    }

    @Test
    void login_short_username() {
        var repo = Mockito.mock(UserRepository.class);
        var svc = new AuthService(repo);
        assertThrows(IllegalArgumentException.class, () -> svc.authenticate("a"
            , "abc"));
    }

    @Test
    void login_long_username() {
        var repo = Mockito.mock(UserRepository.class);
        var svc = new AuthService(repo);
        assertThrows(IllegalArgumentException.class, () -> svc.authenticate("a"
            .repeat(51), "abc"));
    }
}

```

```

    }

    @Test
    void login_invalid_username() {
        var repo = Mockito.mock(UserRepository.class);
        var svc = new AuthService(repo);
        assertThrows(IllegalArgumentException.class, () -> svc.authenticate("
            admin@", "abc"));
    }

    @Test
    void login_empty_password() {
        var repo = Mockito.mock(UserRepository.class);
        var svc = new AuthService(repo);

        assertThrows(IllegalArgumentException.class, () -> svc.authenticate("
            admin", ""));
    }

    @Test
    void login_short_password() {
        var repo = Mockito.mock(UserRepository.class);
        var svc = new AuthService(repo);
        assertThrows(IllegalArgumentException.class, () -> svc.authenticate("
            admin", "a"));
    }

    @Test
    void login_long_password() {
        var repo = Mockito.mock(UserRepository.class);
        var svc = new AuthService(repo);
        assertThrows(IllegalArgumentException.class, () -> svc.authenticate("
            admin", "a".repeat(101)));
    }

    @Test
    void login_invalid_password() {
        var repo = Mockito.mock(UserRepository.class);
        var svc = new AuthService(repo);
        assertThrows(IllegalArgumentException.class, () -> svc.authenticate("
            admin", "admin"));
    }
}

```

b) Test validation methods riêng lẻ

```

package com.sgu.login;

import static org.junit.jupiter.api.Assertions.assertDoesNotThrow;
import static org.junit.jupiter.api.Assertions.assertThrows;
import org.junit.jupiter.api.Test;
import org.mockito.Mockito;

import com.sgu.login.repository.UserRepository;
import com.sgu.login.service.AuthService;

public class ValidateInputTest {

    @Test
    void validateInput_valid_username_and_password(){
        var repo = Mockito.mock(UserRepository.class);
        var svc = new AuthService(repo);

        assertDoesNotThrow(() -> svc.validateInput("admin", "Admin123"));
    }
}

```

```

}

@Test
void validateInput_empty_username(){
    var repo = Mockito.mock(UserRepository.class);
    var svc = new AuthService(repo);

    assertThrows(IllegalArgumentException.class, () -> svc.validateInput("",
        , "Admin123"));
}

@Test
void validateInput_short_username() {
    var repo = Mockito.mock(UserRepository.class);
    var svc = new AuthService(repo);

    assertThrows(IllegalArgumentException.class, () -> svc.validateInput("
        ab", "Admin123"));
}

@Test
void validateInput_long_username() {
    var repo = Mockito.mock(UserRepository.class);
    var svc = new AuthService(repo);

    assertThrows(IllegalArgumentException.class, () -> svc.validateInput("a
        ".repeat(51), "Admin123"));
}

@Test
void validateInput_invalid_username_characters() {
    var repo = Mockito.mock(UserRepository.class);
    var svc = new AuthService(repo);

    assertThrows(IllegalArgumentException.class, () -> svc.validateInput("
        admin@123", "Admin123"));
}

@Test
void validateInput_empty_password() {
    var repo = Mockito.mock(UserRepository.class);
    var svc = new AuthService(repo);

    assertThrows(IllegalArgumentException.class, () -> svc.validateInput("
        admin", ""));
}

@Test
void validateInput_short_password() {
    var repo = Mockito.mock(UserRepository.class);
    var svc = new AuthService(repo);

    assertThrows(IllegalArgumentException.class, () -> svc.validateInput("
        admin", "abc"));
}

@Test
void validateInput_long_password() {
    var repo = Mockito.mock(UserRepository.class);
    var svc = new AuthService(repo);

    assertThrows(IllegalArgumentException.class, () -> svc.validateInput("
        admin", "a".repeat(101)));
}

```

```

    }

    @Test
    void validateInput_password_without_letters() {
        var repo = Mockito.mock(UserRepository.class);
        var svc = new AuthService(repo);

        assertThrows(IllegalArgumentException.class, () -> svc.validateInput("
            admin", "123456"));
    }

    @Test
    void validateInput_password_without_numbers() {
        var repo = Mockito.mock(UserRepository.class);
        var svc = new AuthService(repo);

        assertThrows(IllegalArgumentException.class, () -> svc.validateInput("
            admin", "abcdef"));
    }

    @Test
    void validateInput_null_username() {
        var repo = Mockito.mock(UserRepository.class);
        var svc = new AuthService(repo);

        assertThrows(IllegalArgumentException.class, () -> svc.validateInput(
            null, "Admin123"));
    }

    @Test
    void validateInput_null_password() {
        var repo = Mockito.mock(UserRepository.class);
        var svc = new AuthService(repo);

        assertThrows(IllegalArgumentException.class, () -> svc.validateInput("
            admin", null));
    }
}

```

3.2 Product - Unit Tests Frontend và Backend

3.2.1 Frontend Unit Tests - Product Validation

a) Unit tests cho validateProduct()

```

import { describe, it, expect } from 'vitest'
import { validateProduct } from '../utils/validateProduct.js'

describe('validateProduct', () => {
    it('valid product', () => {
        const p = {name: 'Ball', price: 1000, quantity: 1, description: '', category: 'SPORT'}
        expect(validateProduct(p)).toBe(true)
    })
    it('rejects missing name', () => {
        const p = {name: '', price: 1000, quantity: 1, description: '', category: 'SPORT'}
        expect(validateProduct(p)).not.toBe(true)
    })
    it('rejects negative quantity', () => {
        const p = {name: 'Ball', price: 1000, quantity: -1, description: '', category: 'SPORT'}
        expect(validateProduct(p)).not.toBe(true)
    })
})

```

```

it('rejects_price_greater_than_1_billion', () => {
  const p = {name:'Ball', price: 1000000001, quantity:1, description:'',
    category:'SPORT'}
  expect(validateProduct(p)).not.toBe(true)
})
it('rejects_price_less_than_1', () => {
  const p = {name:'Ball', price: 0, quantity:1, description:'', category:'SPORT'}
  expect(validateProduct(p)).not.toBe(true)
})
it('rejects_too_long_description', () => {
  const p = {name:'Ball', price: 1000, quantity:1, description:'a'.repeat(501), category:'SPORT'}
  expect(validateProduct(p)).not.toBe(true)
})
it('rejects_invalid_category', () => {
  const p = {name:'Ball', price: 1000, quantity:1, description:'', category:'UNKNOWN'}
  expect(validateProduct(p)).not.toBe(true)
})
})

```

b) Tests cho Product form component

```

import { render, screen, fireEvent, waitFor } from '@testing-library/react'
import { describe, it, expect, vi } from 'vitest'
import { MemoryRouter } from 'react-router-dom'
import ProductForm from '../components/ProductForm.jsx'
import * as svc from '../services/productService.js'

describe('ProductForm', () => {
  it('create_success', async () => {
    vi.spyOn(svc, 'createProduct').mockResolvedValue({ id: 1 })
    render(
      <MemoryRouter>
        <ProductForm token="t" />
      </MemoryRouter>
    )

    // Fill out the form
    fireEvent.change(screen.getByLabelText(/name/i), { target: { value: 'New Product' } })
    fireEvent.change(screen.getByLabelText(/price/i), { target: { value: '150' } })
    fireEvent.change(screen.getByLabelText(/quantity/i), { target: { value: '10' } })
    fireEvent.change(screen.getByLabelText(/description/i), { target: { value: 'A great new product' } })
    fireEvent.change(screen.getByLabelText(/category/i), { target: { value: 'ELECTRONICS' } })

    fireEvent.submit(screen.getByLabelText('product-form'))

    // Đọc và xác minh thông báo thành công xuất hiện
    const successMessage = await screen.findByRole('status')
    expect(successMessage).toHaveTextContent('Created product id=1')
  })
})

```

3.2.2 Backend Unit Tests - Product Service

a) Test CRUD operations

```

package com.sgu.login;

import com.sgu.login.model.Product;
import com.sgu.login.repository.ProductRepository;
import com.sgu.login.service.ProductService;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import org.mockito.ArgumentCaptor;
import org.mockito.Mockito;

import java.util.Arrays;
import java.util.List;
import java.util.NoSuchElementException;
import java.util.Optional;

import static org.junit.jupiter.api.Assertions.*;

public class ProductServiceTest {

    private ProductRepository repo;
    private ProductService svc;

    @BeforeEach
    void setup() {
        repo = Mockito.mock(ProductRepository.class);
        svc = new ProductService(repo);
    }

    @Test
    void create_saves() {
        var p = new Product("Ball", 1000, 1, "", "SPORT");
        Mockito.when(repo.save(Mockito.any())).thenAnswer(inv -> inv.
            getArgument(0));

        var saved = svc.create(p);

        ArgumentCaptor<Product> cap = ArgumentCaptor.forClass(Product.class);
        Mockito.verify(repo).save(cap.capture());

        assertEquals("Ball", cap.getValue().getName());
        assertEquals("Ball", saved.getName());
    }

    @Test
    void get_all_returns_list() {
        var list = Arrays.asList(
            new Product("Ball", 1000, 1, "", "SPORT"),
            new Product("Shoe", 5000, 2, "", "SPORT")
        );
        Mockito.when(repo.findAll()).thenReturn(list);

        List<Product> result = svc.getAll();
        assertEquals(2, result.size());
    }

    @Test
    void update_existing_product() {
        var p = new Product("Ball", 1000, 1, "", "SPORT");
        Mockito.when(repo.findById(1L)).thenReturn(Optional.of(p));

        var update = new Product("Ball", 2000, 3, "", "SPORT");
        svc.update(1L, update);
    }

```

```

        Mockito.verify(repo).findById(1L);
    }

    @Test
    void delete_existing_product() {
        var p = new Product("Ball", 1000, 1, "", "SPORT");
        Mockito.when(repo.findById(1L)).thenReturn(Optional.of(p));

        svc.delete(1L);
        Mockito.verify(repo).delete(p);
    }

    @Test
    void delete_not_found_throws() {
        Mockito.when(repo.findById(99L)).thenReturn(Optional.empty());
        assertThrows(NoSuchElementException.class, () -> svc.delete(99L));
    }

    @Test
    void update_not_found_throws() {
        var p = new Product("Ball", 2000, 3, "", "SPORT");
        Mockito.when(repo.findById(1L)).thenReturn(Optional.empty());
        assertThrows(NoSuchElementException.class, () -> svc.update(1L, p));
    }

    @Test
    void get_by_id_returns_product() {
        var p = new Product("Ball", 1000, 1, "", "SPORT");
        Mockito.when(repo.findById(1L)).thenReturn(Optional.of(p));

        Product result = svc.getById(1L);
        assertEquals("Ball", result.getName());
    }

    @Test
    void get_by_id_not_found_throws() {
        Mockito.when(repo.findById(1L)).thenReturn(Optional.empty());
        assertThrows(NoSuchElementException.class, () -> svc.getById(1L));
    }
}

```

4 Integration Testing

4.1 Login - Integration Testing

4.1.1 Frontend Component Integration

a) Test rendering và user interactions

```

vi.mock('../services/authService.js', () => ({
  login: vi.fn(),
}))

describe('Login_Integration', () => {
  afterEach(() => {
    vi.clearAllMocks()
  })

  test('Success_flow: Render -> Nhập email -> Submit -> Gọi API đúng tham số -> Thành công', async () => {
    // Setup
    login.mockResolvedValueOnce('demo-token-admin')
    const onSuccess = vi.fn()

```

```
// 2. ọBc Router
render(
  <MemoryRouter>
    <LoginForm onSuccess={onSuccess} />
  </MemoryRouter>
)

// --- YÊU ẦCU A: Test Rendering & Interactions ---
// ểKim tra input có ểhin ịth không
const usernameInput = screen.getByLabelText(/username/i) // ặHoc getByRole
('textbox')
const passwordInput = screen.getByLabelText(/password/i)
const submitBtn = screen.getByRole('button', { name: /login/i })

expect(usernameInput).toBeInTheDocument()
expect(passwordInput).toBeInTheDocument()

// ắGi ậlp ườngi dùng ậnhp ệliu (Interaction)
fireEvent.change(usernameInput, { target: { value: 'admin' } })
fireEvent.change(passwordInput, { target: { value: 'password123' } })
```

b) Test form submission và API calls

```
fireEvent.click(submitBtn)

await waitFor(() => {
  expect(login).toHaveBeenCalledTimes(1)
  // QUAN ỚTRNG: ểKim tra xem API có được ọgi ớvi đúng ữđ ệliu ừv ậnhp không
  expect(login).toHaveBeenCalledWith({
    username: 'admin',
    password: 'password123'
  })
})
```

c) Test error handling và success messages

```
expect(onSuccess).toHaveBeenCalledWith('demo-token-admin')
})
})

test('Error flow: ậNhậpệliu->Submit->APIốli->ểHinịththông báo', async
() => {
  login.mockRejectedValueOnce(new Error('Invalid credentials'))
  const onSuccess = vi.fn()

  render(
    <MemoryRouter>
      <LoginForm onSuccess={onSuccess} />
    </MemoryRouter>
  )

  // ắVn ậphi ậnhp ệliu đểđ ỏm ắbo quy trình đúng
  fireEvent.change(screen.getByLabelText(/username/i), { target: { value: '
    wrong' } })
  fireEvent.change(screen.getByLabelText(/password/i), { target: { value: '
    wrong123' } })

  fireEvent.click(screen.getByRole('button', { name: /login/i }))

  await waitFor(() => {
    expect(onSuccess).not.toHaveBeenCalled()
  })
  // --- YÊU ẦCU C: Error Handling ---
  const alert = screen.getByRole('alert')
  expect(alert.textContent).toMatch(/invalid credentials/i)
```

```
})
```

4.1.2 Backend API Integration

a) Test POST /api/auth/login endpoint

```
package com.sgu.login.controller;

import org.junit.jupiter.api.DisplayName;
import org.junit.jupiter.api.Test;
import static org.mockito.ArgumentMatchers.anyString;
import org.mockito.Mockito;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.autoconfigure.web.servlet.WebMvcTest;
import org.springframework.boot.test.mock.mockito.MockBean;
import org.springframework.http.MediaType;
import org.springframework.test.web.servlet.MockMvc;
import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.post;
import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.jsonPath;
import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.status;

import com.fasterxml.jackson.databind.ObjectMapper;
import com.sgu.login.dto.LoginRequest;
import com.sgu.login.service.AuthService;

@WebMvcTest(AuthController.class)
@DisplayName("AuthControllerIntegrationTests")
public class AuthControllerIntegrationTest {

    @Autowired
    private MockMvc mockMvc;

    @MockBean
    private AuthService authService;

    @Autowired
    private ObjectMapper objectMapper;

    @Test
    @DisplayName("POST /api/auth/login - Success")
    void testLoginSuccess() throws Exception {
        LoginRequest request = new LoginRequest("admin", "Admin123");

        // Giả lập AuthService trả về token adng õchui
        Mockito.when(authService.authenticate(anyString(), anyString()))
            .thenReturn("fake-token-123");

        mockMvc.perform(post("/api/auth/login")
            .contentType(MediaType.APPLICATION_JSON)
            .content(objectMapper.writeValueAsString(request)))
            .andExpect(status().isOk())
            .andExpect(jsonPath("$.success").value(true))
            .andExpect(jsonPath("$.token").value("fake-token-123"));
    }

    @Test
    @DisplayName("POST /api/auth/login - Validation failed")
    void testLoginValidationFail() throws Exception {
        // Username quá ngắn
        LoginRequest badRequest = new LoginRequest("a", "123");
    }
}
```

```

        mockMvc.perform(post("/api/auth/login")
            .contentType(MediaType.APPLICATION_JSON)
            .content(objectMapper.writeValueAsString(badRequest))
        )
        .andExpect(status().isBadRequest());
    }
}

```

b) Test response structure và status codes

```

@Test
@DisplayName("POST /api/auth/login - Success")
void testLoginSuccess() throws Exception {
    LoginRequest request = new LoginRequest("admin", "Admin123");

    // Giả lập AuthService trả về token adng õchui
    Mockito.when(authService.authenticate(anyString(), anyString()))
        .thenReturn("fake-token-123");

    mockMvc.perform(post("/api/auth/login")
        .contentType(MediaType.APPLICATION_JSON)
        .content(objectMapper.writeValueAsString(request)))
        .andExpect(status().isOk()) // ← Status code 200
        .andExpect(jsonPath("$.success").value(true)) // ← Response structure
        .andExpect(jsonPath("$.token").value("fake-token-123")); // ← Response field
}

@Test
@DisplayName("POST /api/auth/login - Validation failed")
void testLoginValidationFail() throws Exception {
    // Username quá ngắn
    LoginRequest badRequest = new LoginRequest("a", "123");

    mockMvc.perform(post("/api/auth/login")
        .contentType(MediaType.APPLICATION_JSON)
        .content(objectMapper.writeValueAsString(badRequest)))
        .andExpect(status().isBadRequest()); // ← Status code 400
}

@Test
@DisplayName("GET /api/products - should return list")
void testGetAllProducts() throws Exception {
    List<Product> products = List.of(
        new Product("Laptop", 15000000, 10, "High-end", "ELECTRONIC"),
        new Product("Ball", 100000, 5, "Football", "SPORT")
    );
    Mockito.when(service.getAll()).thenReturn(products);

    mockMvc.perform(get("/api/products"))
        .andExpect(status().isOk()) // ← Status 200
        .andExpect(jsonPath("$", hasSize(2))) // ← Array size
        .andExpect(jsonPath("$[0].name").value("Laptop")) // ← Response structure
        .andExpect(jsonPath("$[1].category").value("SPORT"));
}

@Test
@DisplayName("GET /api/products/{id} - should return single product")
void testGetProductById() throws Exception {
    Product product = new Product("Laptop", 15000000, 10, "High-end", "ELECTRONIC");
    product.setId(1L);
}

```

```

Mockito.when(service.getById(1L)).thenReturn(product);

mockMvc.perform(get("/api/products/1"))
    .andExpect(status().isOk()) // ← Status 200
    .andExpect(jsonPath("$.id").value(1)) // ← Response
        fields
    .andExpect(jsonPath("$.name").value("Laptop"))
    .andExpect(jsonPath("$.price").value(15000000))
    .andExpect(jsonPath("$.quantity").value(10))
    .andExpect(jsonPath("$.category").value("ELECTRONIC"));
}

@Test
@DisplayName("POST /api/products → should create new product when token valid")
void testCreateProductSuccess() throws Exception {
    Product req = new Product("Phone", 10000000, 3, "Smartphone", "ELECTRONIC");
    Product saved = new Product("Phone", 10000000, 3, "Smartphone", "ELECTRONIC");
    saved.setId(10L);

    Mockito.when(service.create(any(Product.class))).thenReturn(saved);

    mockMvc.perform(post("/api/products")
        .contentType(MediaType.APPLICATION_JSON)
        .header("Authorization", "Bearer demo-token-123")
        .content(mapper.writeValueAsString(req)))
        .andExpect(status().isOk()) // ← Status 200
        .andExpect(jsonPath("$.id").value(10)) // ← Response
            structure
        .andExpect(jsonPath("$.name").value("Phone"));
}

@Test
@DisplayName("DELETE /api/products/{id} → should delete product")
void testDeleteProductSuccess() throws Exception {
    Mockito.doNothing().when(service).delete(1L);

    mockMvc.perform(delete("/api/products/1"))
        .andExpect(status().isNoContent()); // ← Status 204
}

```

c) Test CORS và headers

```

@Test
@DisplayName("POST /api/products → should return 401 if missing Authorization header")
void testCreateProductUnauthorized() throws Exception {
    Product req = new Product("Phone", 10000000, 3, "Smartphone", "ELECTRONIC");

    mockMvc.perform(post("/api/products")
        .contentType(MediaType.APPLICATION_JSON) // ← Content-Type header
        .content(mapper.writeValueAsString(req)))
        .andExpect(status().isUnauthorized()); // ← 401 when no Authorization header
}

mockMvc.perform(post("/api/products")
    .contentType(MediaType.APPLICATION_JSON) //Content-Type header
    .header("Authorization", "Bearer demo-token-123")
    .content(mapper.writeValueAsString(req)))

```

4.2 Product - Integration Testing

4.2.1 Frontend Component Integration

a) Test ProductList component với API

```
/**
 * ProductList Integration Test
 */
import React from 'react'
import { render, screen, waitFor } from '@testing-library/react'
import { vi } from 'vitest'
import { MemoryRouter } from 'react-router-dom'

vi.mock('../services/productService.js', () => ({
  getProducts: vi.fn(),
}))

import { getProducts } from '../services/productService.js'
import ProductList from '../components/ProductList.jsx'

describe('ProductList Integration', () => {
  afterEach(() => vi.clearAllMocks())

  test('Hành động danh sách sản phẩm từ API', async () => {
    getProducts.mockResolvedValueOnce([
      { id: 1, name: 'Ball' },
      { id: 2, name: 'Shoes' },
    ])

    render(
      <MemoryRouter>
        <ProductList />
      </MemoryRouter>
    )

    await waitFor(() => {
      const items = screen.getAllByTestId('product-item')
      expect(items).toHaveLength(2)
      expect(items[0]).toHaveTextContent('Ball')
    })
  })

  test('Hành động lỗi khi API lỗi', async () => {
    getProducts.mockRejectedValueOnce(new Error('Server error'))
    render(
      <MemoryRouter>
        <ProductList />
      </MemoryRouter>
    )

    await waitFor(() => {
      expect(screen.getByRole('alert').textContent).toMatch(/server error/i)
    })
  })
})
```

b) Test ProductForm component (create/edit)

```
/**
 * ProductList Integration Test
 */
import React from 'react'
import { render, screen, waitFor } from '@testing-library/react'
import { vi } from 'vitest'
```

```

import { MemoryRouter } from 'react-router-dom'

vi.mock('../services/productService.js', () => ({
  getProducts: vi.fn(),
}))

import { getProducts } from '../services/productService.js'
import ProductList from '../components/ProductList.jsx'

describe('ProductList Integration', () => {
  afterEach(() => vi.clearAllMocks())

  test('Hành động danh sách sản phẩm từ API', async () => {
    getProducts.mockResolvedValueOnce([
      { id: 1, name: 'Ball' },
      { id: 2, name: 'Shoes' },
    ])

    render(
      <MemoryRouter>
        <ProductList />
      </MemoryRouter>
    )

    await waitFor(() => {
      const items = screen.getAllByTestId('product-item')
      expect(items).toHaveLength(2)
      expect(items[0]).toHaveTextContent('Ball')
    })
  })

  test('Hành động lỗi khi API lỗi', async () => {
    getProducts.mockRejectedValueOnce(new Error('Server error'))
    render(
      <MemoryRouter>
        <ProductList />
      </MemoryRouter>
    )

    await waitFor(() => {
      expect(screen.getByRole('alert').textContent).toMatch(/server error/i)
    })
  })
})

```

c) Test ProductDetail component

```

/**
 * ProductDetail Integration Test
 */
import React from 'react'
import { render, screen, waitFor } from '@testing-library/react'
import { vi } from 'vitest'

vi.mock('../services/productService.js', () => ({
  getProductById: vi.fn(),
}))

import { getProductById } from '../services/productService.js'
import ProductDetail from '../components/ProductDetail.jsx'

describe('ProductDetail Integration', () => {
  afterEach(() => vi.clearAllMocks())

```

```

test('Hinh ảnh chi tiết sản phẩm khi load thành công', async () => {
  getProductById.mockResolvedValueOnce({
    id: 1,
    name: 'Ball',
    price: 1000,
    quantity: 2,
    category: 'SPORT',
  })

  render(<ProductDetail id={1} />)

  await waitFor(() => {
    expect(screen.getByLabelText('product-detail')).toBeInTheDocument()
    expect(screen.getByText('Ball')).toBeInTheDocument()
    expect(screen.getByTestId('price').textContent).toBe('1000')
  })
})

test('Hinh ảnh lỗi khi API lỗi', async () => {
  getProductById.mockRejectedValueOnce(new Error('Not found'))

  render(<ProductDetail id={99} />)

  await waitFor(() => {
    expect(screen.getByRole('alert').textContent).toMatch(/not found/i)
  })
})
}

```

4.2.2 Backend API Integration

a) Test POST /api/products (Create)

```

// POST /api/products
@Test
@DisplayName("POST /api/products → should create new product when token valid")
void testCreateProductSuccess() throws Exception {
  Product req = new Product("Phone", 10000000, 3, "Smartphone", "ELECTRONIC");
  Product saved = new Product("Phone", 10000000, 3, "Smartphone", "ELECTRONIC");
  saved.setId(10L);

  Mockito.when(service.create(any(Product.class))).thenReturn(saved);

  mockMvc.perform(post("/api/products")
    .contentType(MediaType.APPLICATION_JSON)
    .header("Authorization", "Bearer demo-token-123")
    .content(mapper.writeValueAsString(req)))
    .andExpect(status().isOk())
    .andExpect(jsonPath("$.id").value(10))
    .andExpect(jsonPath("$.name").value("Phone"));
}

// POST /api/products - không có token
@Test
@DisplayName("POST /api/products → should return 401 if missing Authorization header")
void testCreateProductUnauthorized() throws Exception {
  Product req = new Product("Phone", 10000000, 3, "Smartphone", "ELECTRONIC");

  mockMvc.perform(post("/api/products")

```

```

        .contentType(MediaType.APPLICATION_JSON)
        .content(mapper.writeValueAsString(req)))
        .andExpect(status().isUnauthorized());
    }

```

b) Test GET /api/products (Read all)

```

// GET /api/products
@Test
@DisplayName("GET /api/products should return list")
void testGetAllProducts() throws Exception {
    List<Product> products = List.of(
        new Product("Laptop", 15000000, 10, "High-end", "ELECTRONIC"),
        new Product("Ball", 100000, 5, "Football", "SPORT")
    );
    Mockito.when(service.getAll()).thenReturn(products);

    mockMvc.perform(get("/api/products"))
        .andExpect(status().isOk())
        .andExpect(jsonPath("$", hasSize(2)))
        .andExpect(jsonPath("$[0].name").value("Laptop"))
        .andExpect(jsonPath("$[1].category").value("SPORT"));
}

```

c) Test GET /api/products/id (Read one)

```

// GET /api/products/{id}
@Test
@DisplayName("GET /api/products/{id} should return single product")
void testGetProductById() throws Exception {
    Product product = new Product("Laptop", 15000000, 10, "High-end", "ELECTRONIC");
    product.setId(1L);

    Mockito.when(service.getById(1L)).thenReturn(product);

    mockMvc.perform(get("/api/products/1"))
        .andExpect(status().isOk())
        .andExpect(jsonPath("$.id").value(1))
        .andExpect(jsonPath("$.name").value("Laptop"))
        .andExpect(jsonPath("$.price").value(15000000))
        .andExpect(jsonPath("$.quantity").value(10))
        .andExpect(jsonPath("$.category").value("ELECTRONIC"));
}

```

d) Test PUT /api/products/id (Update)

```

@Test
@DisplayName("PUT /api/products/{id} should update product")
void testUpdateProductSuccess() throws Exception {
    Product req = new Product("Updated Laptop", 20000000, 15, "Premium", "ELECTRONIC");
    Product updated = new Product("Updated Laptop", 20000000, 15, "Premium", "ELECTRONIC");
    updated.setId(1L);

    Mockito.when(service.update(any(Long.class), any(Product.class))).thenReturn(updated);

    mockMvc.perform(put("/api/products/1")
        .contentType(MediaType.APPLICATION_JSON)
        .content(mapper.writeValueAsString(req)))
        .andExpect(status().isOk())
        .andExpect(jsonPath("$.id").value(1))

```

```

        .andExpect(jsonPath("$.name").value("Updated Laptop"))
        .andExpect(jsonPath("$.price").value(20000000))
        .andExpect(jsonPath("$.quantity").value(15));
    }

```

e) Test DELETE /api/products/id (Delete)

```

// DELETE /api/products/{id}
@Test
@DisplayName("DELETE /api/products/{id} should delete product")
void testDeleteProductSuccess() throws Exception {
    Mockito.doNothing().when(service).delete(1L);

    mockMvc.perform(delete("/api/products/1"))
        .andExpect(status().isNoContent());
}

```

5 Mock Testing

5.1 Login - Mock Testing

5.1.1 Frontend Mocking

a) Mock authService.loginUser()

```

import { render, screen, fireEvent } from "@testing-library/react"
import * as authService from "@services/authService"
import LoginForm from "@components/LoginForm"

// ảGi ảp toàn ộb module authService
vi.mock("@services/authService", () => ({
    login: vi.fn(),
}))

// ảTo 1 test suite ởvi describe/test
describe("Mock testing for LoginForm", () => {
    test("mock login success & failure", async () => {
        // THÀNH CÔNG
        authService.login.mockResolvedValueOnce({ token: "demo-token" })
        render(<LoginForm />)

        const usernameInput = screen.getByRole("textbox") // input ầu tiên là text
        const passwordInput = document.querySelector('input[type="password"]') //
            tìm input password theo type

        fireEvent.change(usernameInput, { target: { value: "admin" } })
        fireEvent.change(passwordInput, { target: { value: "Admin123" } })

        // Nút ểhin ith ữch "Login", nên ổi selector cho úng
        fireEvent.click(screen.getByRole("button", { name: /login/i }))

        // ờCh ểhin ith thông báo thành công (ảgi ửs có dòng text "Success" ặhoc
            ươtng ựt)
        expect(await screen.findByText(/success/i)).toBeInTheDocument()

        // ểKim tra ốs ảln ọgi mock
        expect(authService.login).toHaveBeenCalledTimes(1)
    })
}

```

b) Test với mocked successful/failed responses

```

// ờCh ểhin ith thông báo thành công (ảgi ửs có dòng text "Success" ặhoc
    ươtng ựt)

```

```

expect(await screen.findByText(/success/i)).toBeInTheDocument()

// ểKim tra ốs ầln ọgi mock
expect(authService.login).toHaveBeenCalledTimes(1)

// ẦTHT ẠBI
authService.login.mockRejectedValueOnce(new Error("Invalid credentials"))
fireEvent.click(screen.getByRole("button", { name: /login/i }))

// ờCh thông báo ỗili
expect(await screen.findByText(/invalid/i)).toBeInTheDocument()

```

c) Verify mock calls

```

// ểKim tra ốs ầln ọgi mock
expect(authService.login).toHaveBeenCalledTimes(1)

// ẦTHT ẠBI
authService.login.mockRejectedValueOnce(new Error("Invalid credentials"))
fireEvent.click(screen.getByRole("button", { name: /login/i }))

// ờCh thông báo ỗili
expect(await screen.findByText(/invalid/i)).toBeInTheDocument()

```

5.1.2 Backend Mocking

a) Mock AuthService với @MockBean

```

package com.sgu.login;

import java.util.Optional;

import static org.junit.jupiter.api.Assertions.assertNotNull;
import static org.junit.jupiter.api.Assertions.assertThrows;
import static org.junit.jupiter.api.Assertions.assertTrue;
import org.junit.jupiter.api.Test;
import org.mockito.Mockito;

import com.sgu.login.model.User;
import com.sgu.login.repository.UserRepository;
import com.sgu.login.service.AuthService;

public class AuthServiceTest {

    @Test
    void login_success() {
        var repo = Mockito.mock(UserRepository.class);
        Mockito.when(repo.findByUsername("admin"))
            .thenReturn(Optional.of(new User("admin", "Admin123")));
        var svc = new AuthService(repo);

        String token = svc.authenticate("admin", "Admin123");

        assertNotNull(token);
        assertTrue(token.startsWith("demo-token-"));
    }

    @Test
    void login_wrong_password() {
        var repo = Mockito.mock(UserRepository.class);
        Mockito.when(repo.findByUsername("admin"))
            .thenReturn(Optional.of(new User("admin", "Admin123")));
        var svc = new AuthService(repo);
    }
}

```

```

        assertThrows(RuntimeException.class, () -> svc.authenticate("admin",
            "x"));
    }

    @Test
    void login_user_not_found() {
        var repo = Mockito.mock(UserRepository.class);
        Mockito.when(repo.findByUsername("ghost")).thenReturn(Optional.empty());
        var svc = new AuthService(repo);

        assertThrows(RuntimeException.class, () -> svc.authenticate("ghost",
            "abc"));
    }

    @Test
    void login_empty_username() {
        var repo = Mockito.mock(UserRepository.class);
        var svc = new AuthService(repo);

        assertThrows(IllegalArgumentException.class, () -> svc.authenticate(
            "", "abc"));
    }

    @Test
    void login_short_username() {
        var repo = Mockito.mock(UserRepository.class);
        var svc = new AuthService(repo);
        assertThrows(IllegalArgumentException.class, () -> svc.authenticate(
            "a", "abc"));
    }

    @Test
    void login_long_username() {
        var repo = Mockito.mock(UserRepository.class);
        var svc = new AuthService(repo);
        assertThrows(IllegalArgumentException.class, () -> svc.authenticate(
            "a".repeat(51), "abc"));
    }

    @Test
    void login_invalid_username() {
        var repo = Mockito.mock(UserRepository.class);
        var svc = new AuthService(repo);
        assertThrows(IllegalArgumentException.class, () -> svc.authenticate(
            "admin@", "abc"));
    }

    @Test
    void login_empty_password() {
        var repo = Mockito.mock(UserRepository.class);
        var svc = new AuthService(repo);

        assertThrows(IllegalArgumentException.class, () -> svc.authenticate(
            "admin", ""));
    }

    @Test
    void login_short_password() {
        var repo = Mockito.mock(UserRepository.class);
        var svc = new AuthService(repo);
    }

```

```

        assertThrows(IllegalArgumentException.class, () -> svc.authenticate("
            admin", "a"));
    }

    @Test
    void login_long_password() {
        var repo = Mockito.mock(UserRepository.class);
        var svc = new AuthService(repo);
        assertThrows(IllegalArgumentException.class, () -> svc.authenticate("
            admin", "a".repeat(101)));
    }

    @Test
    void login_invalid_password() {
        var repo = Mockito.mock(UserRepository.class);
        var svc = new AuthService(repo);
        assertThrows(IllegalArgumentException.class, () -> svc.authenticate("
            admin", "admin"));
    }
}

```

b) Test controller với mocked service

```

package com.sgu.login.controller;

import org.junit.jupiter.api.DisplayName;
import org.junit.jupiter.api.Test;
import static org.mockito.ArgumentMatchers.any;
import static org.mockito.ArgumentMatchers.anyString;
import static org.mockito.ArgumentMatchers.eq;
import static org.mockito.Mockito.times;
import static org.mockito.Mockito.verify;
import static org.mockito.Mockito.when;
import static org.mockito.Mockito.never;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.autoconfigure.web.servlet.WebMvcTest;
import org.springframework.boot.test.mock.mockito.MockBean;
import org.springframework.http.MediaType;
import org.springframework.test.web.servlet.MockMvc;
import static org.springframework.test.web.servlet.request.
    MockMvcRequestBuilders.post;
import static org.springframework.test.web.servlet.result.
    MockMvcResultMatchers.jsonPath;
import static org.springframework.test.web.servlet.result.
    MockMvcResultMatchers.status;

import com.fasterxml.jackson.databind.ObjectMapper;
import com.sgu.login.dto.LoginRequest;
import com.sgu.login.service.AuthService;

@WebMvcTest(AuthController.class)
@DisplayName("AuthController Backend Mocking Tests")
public class AuthControllerMockTest {

    @Autowired
    private MockMvc mockMvc;

    @MockBean
    private AuthService authService;

    @Autowired
    private ObjectMapper objectMapper;

    @Test

```

```

@DisplayName("Mock: Controller ở vi mocked service success")
void testLoginWithMockedServiceSuccess() throws Exception {
    // Arrange: Setup mock behavior
    when(authService.authenticate(anyString(), anyString()))
        .thenReturn("mock-token-12345");

    LoginRequest request = new LoginRequest("testuser", "Pass123");

    // Act & Assert
    mockMvc.perform(post("/api/auth/login")
        .contentType(MediaType.APPLICATION_JSON)
        .content(objectMapper.writeValueAsString(request)))
        .andExpect(status().isOk())
        .andExpect(jsonPath("$.success").value(true))
        .andExpect(jsonPath("$.message").value("Đăng nhập thành công"))
        .andExpect(jsonPath("$.token").value("mock-token-12345"));
}

@Test
@DisplayName("Mock: Controller ở vi different credentials")
void testLoginWithDifferentCredentials() throws Exception {
    // Mock trả về token khác cho user khác
    when(authService.authenticate(eq("admin"), eq("Admin123")))
        .thenReturn("admin-token-xyz");

    LoginRequest request = new LoginRequest("admin", "Admin123");

    mockMvc.perform(post("/api/auth/login")
        .contentType(MediaType.APPLICATION_JSON)
        .content(objectMapper.writeValueAsString(request)))
        .andExpect(status().isOk())
        .andExpect(jsonPath("$.token").value("admin-token-xyz"));
}

@Test
@DisplayName("Mock: Service không được gọi khi validation fails")
void testServiceNotCalledWhenValidationFails() throws Exception {
    // Request không hợp lệ (username quá ngắn)
    LoginRequest invalidRequest = new LoginRequest("ab", "Pass123");

    mockMvc.perform(post("/api/auth/login")
        .contentType(MediaType.APPLICATION_JSON)
        .content(objectMapper.writeValueAsString(
            invalidRequest)))
        .andExpect(status().isBadRequest());

    // Verify rằng service KHÔNG được gọi khi validation fail
    verify(authService, never()).authenticate(anyString(), anyString());
}

```

c) Verify mock interactions

```

@Test
@DisplayName("Verify: Service được gọi đúng 1 lần ở vi tham số chính xác")
void testVerifyServiceCalledOnce() throws Exception {
    when(authService.authenticate(anyString(), anyString()))
        .thenReturn("verify-token");

    LoginRequest request = new LoginRequest("verifyuser", "VerifyPass123");

    mockMvc.perform(post("/api/auth/login")
        .contentType(MediaType.APPLICATION_JSON)

```

```

        .content(objectMapper.writeValueAsString(request)))
        .andExpect(status().isOk());

    // Verify service được gọi đúng 1 lần
    verify(authService, times(1)).authenticate(anyString(), anyString());
}

@Test
@DisplayName("Verify: Service được gọi ở vi phạm số chính xác")
void testVerifyServiceCalledWithCorrectParameters() throws Exception {
    when(authService.authenticate(anyString(), anyString()))
        .thenReturn("param-token");

    LoginRequest request = new LoginRequest("exactuser", "ExactPass123");

    mockMvc.perform(post("/api/auth/login")
        .contentType(MediaType.APPLICATION_JSON)
        .content(objectMapper.writeValueAsString(request)))
        .andExpect(status().isOk());

    // Verify service được gọi ở vi phạm đúng username và password
    verify(authService, times(1)).authenticate(eq("exactuser"), eq("ExactPass123"));
}

@Test
@DisplayName("Verify: Multiple requests call service multiple times")
void testVerifyMultipleServiceCalls() throws Exception {
    when(authService.authenticate(anyString(), anyString()))
        .thenReturn("multi-token-1")
        .thenReturn("multi-token-2");

    LoginRequest request1 = new LoginRequest("user1", "Pass123");
    LoginRequest request2 = new LoginRequest("user2", "Pass456");

    // First request
    mockMvc.perform(post("/api/auth/login")
        .contentType(MediaType.APPLICATION_JSON)
        .content(objectMapper.writeValueAsString(request1)))
        .andExpect(status().isOk())
        .andExpect(jsonPath("$.token").value("multi-token-1"));

    // Second request
    mockMvc.perform(post("/api/auth/login")
        .contentType(MediaType.APPLICATION_JSON)
        .content(objectMapper.writeValueAsString(request2)))
        .andExpect(status().isOk())
        .andExpect(jsonPath("$.token").value("multi-token-2"));

    // Verify service được gọi 2 lần
    verify(authService, times(2)).authenticate(anyString(), anyString());
}
}

```

5.2 Product - Mock Testing

5.2.1 Frontend Mocking

a) Mock CRUD operations

```

import * as productService from '../services/productService';

jest.mock('../services/productService');

```

```

describe('Product_Service_Mock_CRUD_Tests', () => {

  afterEach(() => {
    jest.clearAllMocks();
  });

  // CREATE (ATO ƠMI)
  describe('CREATE_Operations', () => {
    test('Success:_Create_product_thành_công', async () => {
      // Setup data
      const newProductInput = { name: 'Laptop_Gaming', price: 20000000 };
      const mockCreatedProduct = { id: 1, ...newProductInput };

      // Mock implementation (Requirement a)
      productService.createProduct.mockResolvedValue(mockCreatedProduct);

      // Test implementation (oGi hàm)
      const result = await productService.createProduct(newProductInput);

      // Assertions (Requirement b & c)
      expect(result).toEqual(mockCreatedProduct);
      expect(productService.createProduct).toHaveBeenCalledTimes(1);
      expect(productService.createProduct).toHaveBeenCalledWith(
        newProductInput);
    });

    test('Failure:_Create_product_âtht_âbi_do_õli_Validation', async () => {
      const invalidProduct = { name: '' }; // Tên ãrng
      const error = new Error('Validation_Error:_Name_is_required');

      // Mock implementation failure (Requirement b)
      productService.createProduct.mockRejectedValue(error);

      // Assertions
      await expect(productService.createProduct(invalidProduct)).rejects.
        toThrow('Validation_Error');
      expect(productService.createProduct).toHaveBeenCalledTimes(1);
    });
  });
});

// 2. READ ĐQ(C/ẢLY ỮD ẼLIU)
describe('READ_Operations', () => {
  test('Success:_Get_products_with_pagination', async () => {
    const mockResponse = {
      data: [
        { id: 1, name: 'Laptop', price: 15000000 },
        { id: 2, name: 'Mouse', price: 200000 }
      ],
      page: 1,
      total: 100
    };

    // Mock implementation (Requirement a)
    productService.getProducts.mockResolvedValue(mockResponse);

    // Test implementation
    const result = await productService.getProducts({ page: 1, limit:
      10 });

    // Verify (Requirement c)
    expect(result.data).toHaveLength(2);
    expect(productService.getProducts).toHaveBeenCalledWith({ page: 1,
      limit: 10 });
  });
});

```

```

    });

    test('Success: Get product by ID', async () => {
        const mockProduct = { id: 99, name: 'Keyboard', price: 500000 };
        productService.getProductById.mockResolvedValue(mockProduct);

        const result = await productService.getProductById(99);

        expect(result).toEqual(mockProduct);
        expect(productService.getProductById).toHaveBeenCalledWith(99);
    });
    test('Failure: Get product by ID not found', async () => {
        const error = new Error('Product not found');
        productService.getProductById.mockRejectedValue(error);

        await expect(productService.getProductById(999)).rejects.toThrow('Product not found');
        expect(productService.getProductById).toHaveBeenCalledTimes(1);
    });
});

// 3. UPDATE (Cập nhật)
describe('UPDATE Operations', () => {
    test('Success: Update product thành công', async () => {
        const updateData = { price: 18000000 };
        const updatedProduct = { id: 1, name: 'Laptop', ...updateData };

        // Mock (Requirement a)
        productService.updateProduct.mockResolvedValue(updatedProduct);

        // Call
        const result = await productService.updateProduct(1, updateData);

        // Verify (Requirement c)
        expect(result).toEqual(updatedProduct);
        expect(productService.updateProduct).toHaveBeenCalledWith(1, updateData);
    });

    test('Failure: Update thất bại do lỗi server', async () => {
        const error = new Error('Internal Server Error');
        productService.updateProduct.mockRejectedValue(error);

        await expect(productService.updateProduct(1, {})).rejects.toThrow('Internal Server Error');
        expect(productService.updateProduct).toHaveBeenCalledTimes(1);
    });
});

// 4. DELETE (Xóa)
describe('DELETE Operations', () => {
    test('Success: Delete product thành công', async () => {
        // Khi hàm trả về true khi xóa thành công
        productService.deleteProduct.mockResolvedValue(true);

        const result = await productService.deleteProduct(5);

        expect(result).toBe(true);
        expect(productService.deleteProduct).toHaveBeenCalledWith(5);
    });

    test('Failure: Delete thất bại (ID không tồn tại)', async () => {
        // Khi hàm trả về false hoặc throw error khi không xóa được
    });
});

```

```

        productService.deleteProduct.mockResolvedValue(false);

        const result = await productService.deleteProduct(9999);

        expect(result).toBe(false);
        expect(productService.deleteProduct).toHaveBeenCalledTimes(1);
    });
});
});

```

b) Test success và failure scenarios

```

test('a) Tạo sản phẩm thành công → gọi API đúng payload', async () => {
    // Gọi API trả về kết quả thành công
    createProduct.mockResolvedValueOnce({
        id: 1,
        name: 'Ball',
        price: 1000,
        quantity: 1,
        category: 'SPORT',
    })

    // ... submit form ...

    await waitFor(() => {
        expect(createProduct).toHaveBeenCalledTimes(1)
    })
})

test('b) Tạo sản phẩm lỗi → hiển thị lỗi', async () => {
    // Gọi API ném lỗi
    createProduct.mockRejectedValueOnce(new Error('Server error'))

    // ... submit form ...

    await waitFor(() => {
        const alert = screen.getByRole('alert')
        expect(alert.textContent).toMatch(/server error/i)
    })
})

```

c) Verify all mock calls

```

await waitFor(() => {
    expect(createProduct).toHaveBeenCalledTimes(1)

    // Kiểm tra tham số đầu tiên (form)
    const [payload, token] = createProduct.mock.calls[0]
    expect(payload).toMatchObject({
        name: 'Ball',
        price: 1000,
        quantity: 1,
        description: 'A nice ball',
        category: 'SPORT',
    })

    // Kiểm tra tham số thứ hai (token) có thể là undefined
    expect(token === undefined || typeof token === 'string').toBeTruthy()
})

```

5.2.2 Backend Mocking

a) Mock ProductRepository

```

@Mock
private ProductRepository productRepository;

@InjectMocks
private ProductService productService;

private Product testProduct;

@BeforeEach
void setUp() {
    testProduct = new Product("Laptop", 15000000, 10, "High-end_laptop", "
        ELECTRONIC");
    testProduct.setId(1L);
}

```

b) Test service layer với mocked repository

```

@Test
@DisplayName("Mock: Get product by ID - success")
void testGetProductById() {
    when(productRepository.findById(1L))
        .thenReturn(Optional.of(testProduct));

    Product result = productService.getById(1L);

    assertNotNull(result);
    assertEquals("Laptop", result.getName());
    assertEquals(15000000, result.getPrice());
    assertEquals(10, result.getQuantity());

    verify(productRepository, times(1)).findById(1L);
}

@Test
@DisplayName("Mock: Get product by ID - not found")
void testGetProductByIdNotFound() {
    when(productRepository.findById(99L))
        .thenReturn(Optional.empty());

    assertThrows(NoSuchElementException.class, () -> {
        productService.getById(99L);
    });

    verify(productRepository, times(1)).findById(99L);
}

@Test
@DisplayName("Mock: Get all products")
void testGetAllProducts() {
    Product product2 = new Product("Mouse", 500000, 20, "Gaming_mouse", "
        ELECTRONIC");
    product2.setId(2L);

    List<Product> mockProducts = List.of(testProduct, product2);

    when(productRepository.findAll()).thenReturn(mockProducts);

    List<Product> result = productService.getAll();

    assertNotNull(result);
    assertEquals(2, result.size());
    assertEquals("Laptop", result.get(0).getName());
    assertEquals("Mouse", result.get(1).getName());
}

```

```

        verify(productRepository, times(1)).findAll();
    }

    @Test
    @DisplayName("Mock: Create product")
    void testCreateProduct() {
        Product newProduct = new Product("Keyboard", 1000000, 15, "Mechanical keyboard", "ELECTRONIC");
        Product savedProduct = new Product("Keyboard", 1000000, 15, "Mechanical keyboard", "ELECTRONIC");
        savedProduct.setId(3L);

        when(productRepository.save(any(Product.class))).thenReturn(savedProduct);

        Product result = productService.create(newProduct);

        assertNotNull(result);
        assertEquals(3L, result.getId());
        assertEquals("Keyboard", result.getName());
        assertEquals(1000000, result.getPrice());

        verify(productRepository, times(1)).save(any(Product.class));
    }

    @Test
    @DisplayName("Mock: Update product - success")
    void testUpdateProduct() {
        Product updateData = new Product("Updated Laptop", 20000000, 5, "Premium laptop", "ELECTRONIC");
        Product updatedProduct = new Product("Updated Laptop", 20000000, 5, "Premium laptop", "ELECTRONIC");
        updatedProduct.setId(1L);

        when(productRepository.findById(1L)).thenReturn(Optional.of(testProduct));
        when(productRepository.save(any(Product.class))).thenReturn(updatedProduct);

        Product result = productService.update(1L, updateData);

        assertNotNull(result);
        assertEquals(1L, result.getId());
        assertEquals("Updated Laptop", result.getName());
        assertEquals(20000000, result.getPrice());
        assertEquals(5, result.getQuantity());

        verify(productRepository, times(1)).findById(1L);
        verify(productRepository, times(1)).save(any(Product.class));
    }

    @Test
    @DisplayName("Mock: Update product - not found")
    void testUpdateProductNotFound() {
        Product updateData = new Product("Updated Product", 10000000, 5, "Description", "CATEGORY");

        when(productRepository.findById(99L)).thenReturn(Optional.empty());

        assertThrows(NoSuchElementException.class, () -> {
            productService.update(99L, updateData);
        });
    }

```

```

        verify(productRepository, times(1)).findById(99L);
        verify(productRepository, never()).save(any(Product.class));
    }

    @Test
    @DisplayName("Mock: Delete product - success")
    void testDeleteProduct() {
        when(productRepository.findById(1L)).thenReturn(Optional.of(testProduct));

        productService.delete(1L);

        verify(productRepository, times(1)).findById(1L);
        verify(productRepository, times(1)).delete(testProduct);
    }

    @Test
    @DisplayName("Mock: Delete product - not found")
    void testDeleteProductNotFound() {
        when(productRepository.findById(99L)).thenReturn(Optional.empty());

        assertThrows(NoSuchElementException.class, () -> {
            productService.delete(99L);
        });

        verify(productRepository, times(1)).findById(99L);
        verify(productRepository, never()).delete(any(Product.class));
    }
}

```

c) Verify repository interactions

```

@Test
@DisplayName("Verify: Repository findById called with correct parameter")
void testVerifyFindByIdParameter() {
    when(productRepository.findById(anyLong())).thenReturn(Optional.of(testProduct));

    productService.getById(5L);

    verify(productRepository, times(1)).findById(5L);
}

@Test
@DisplayName("Verify: Repository save called when creating product")
void testVerifySaveCalledOnCreate() {
    Product newProduct = new Product("Monitor", 3000000, 8, "4K Monitor", "ELECTRONIC");

    when(productRepository.save(newProduct)).thenReturn(newProduct);

    productService.create(newProduct);

    verify(productRepository, times(1)).save(newProduct);
}

@Test
@DisplayName("Verify: Multiple repository interactions")
void testVerifyMultipleInteractions() {
    when(productRepository.findById(1L)).thenReturn(Optional.of(testProduct));
    when(productRepository.findById(2L)).thenReturn(Optional.of(testProduct));

    productService.getById(1L);
}

```

```

    productService.getById(2L);

    verify(productRepository, times(2)).findById(anyLong());
    verify(productRepository, times(1)).findById(1L);
    verify(productRepository, times(1)).findById(2L);
}

```

6 Automation Testing và CI/CD

6.1 Login - E2E Automation Testing

6.1.1 Setup và Configuration

- Cài đặt Cypress:

- Cài đặt thư viện:

```
Npm install cypress --save-dev
```

- Cấu trúc thư mục sau khi cài đặt:

```

frontend/
  cypress/
    e2e/
      login.cy.js
    fixtures/
      example.json
    support/
      commands.js
      e2e.js
    page-objects/
      LoginPage.js
      ProductPage.js
    screenshots/
  cypress.config.js
  package.json

```

- Scripts trong package.json:

```

{
  "scripts": {
    "cypress:open": "cypress open",
    "cypress:run" : "cypress run"
  }
}

```

- Cấu hình Test Environment:

```

import { defineConfig } from "cypress";

export default defineConfig({
  e2e: {
    // Base URL configuration
    baseUrl: "http://localhost:5173",

    // Viewport settings
    viewportWidth: 1280,
    viewportHeight: 720,

    // Timeouts
    defaultCommandTimeout: 10000,
    requestTimeout: 10000,
    responseTimeout: 10000,
    pageLoadTimeout: 30000,
  },
});

```

```

// Test files location
specPattern: "cypress/e2e/**/*.cy.{js,jsx,ts,tsx}",
supportFile: "cypress/support/e2e.js",

setupNodeEvents(on, config) {
  // implement node event listeners here
},

// Environment variables
env: {
  apiUrl: "http://localhost:8080",
  loginEndpoint: "/api/auth/login",
  productsEndpoint: "/api/products"
},

// Video and screenshot configuration
video: true,
screenshotOnRunFailure: true,
screenshotsFolder: "cypress/screenshots",
videosFolder: "cypress/videos",

// Security and retry settings
chromeWebSecurity: false,
retries: {
  runMode: 2,
  openMode: 0
}
});

```

- Setup Page Object Model (POM):

```

// Page Object Model for Login Page
class LoginPage {
  // Selectors
  get usernameInput() {
    return cy.get('[data-testid="username-input"]')
  }

  get passwordInput() {
    return cy.get('[data-testid="password-input"]')
  }

  get loginButton() {
    return cy.get('[data-testid="login-button"]')
  }

  get loginError() {
    return cy.get('[data-testid="login-error"]')
  }

  // Actions
  visit() {
    cy.visit('/')
  }

  fillUsername(username) {
    this.usernameInput.clear().type(username)
  }

  fillPassword(password) {
    this.passwordInput.clear().type(password)
  }
}

```

```

clearUsername() {
  this.usernameInput.clear()
}

clearPassword() {
  this.passwordInput.clear()
}

clickLogin() {
  this.loginButton.click()
}

login(username, password) {
  this.fillUsername(username)
  this.fillPassword(password)
  this.clickLogin()
}

// Assertions
verifyUsernameVisible() {
  this.usernameInput.should('be.visible')
}

verifyPasswordVisible() {
  this.passwordInput.should('be.visible')
}

verifyLoginButtonVisible() {
  this.loginButton.should('be.visible')
}

verifyErrorVisible() {
  this.loginError.should('be.visible')
}

verifyErrorMessage(message) {
  this.loginError.should('contain.text', message)
}

verifyErrorNotExist() {
  this.loginError.should('not.exist')
}

verifyRedirectToProducts() {
  cy.url().should('include', '/products')
}

// API Intercepts
interceptLoginSuccess() {
  cy.intercept('POST', '**/api/auth/login', {
    statusCode: 200,
    body: { token: 'fake-token-123', user: { name: 'Admin' } },
  }).as('loginRequest')
}

interceptLoginFailure() {
  cy.intercept('POST', '**/api/auth/login', {
    statusCode: 401,
    body: { message: 'Username does not exist' }
  }).as('loginFailed')
}

```

```
waitForLoginRequest() {
  return cy.wait('@loginRequest', { timeout: 10000 })
}

waitForLoginFailure() {
  return cy.wait('@loginFailed', { timeout: 5000 })
}

export default LoginPage
```

6.1.2 E2E Test Scenarios cho Login

a) Test complete login flow

```
// Test Scenario: Đăng nhập thành công với thông tin hợp lệ
it('Đăng nhập thành công với thông tin hợp lệ', () => {
  loginPage.interceptLoginSuccess()
  loginPage.login('admin', 'Admin123')

  loginPage.waitForLoginRequest()
    .its('response.statusCode')
    .should('eq', 200)

  loginPage.verifyRedirectToProducts()
})
```

b) Test validation messages

```
// Test Scenario: Nhập vào username trống
it('Hành động khi username trống', () => {
    loginPage.clearUsername()
    loginPage.fillPassword('Admin123')
    loginPage.clickLogin()

    loginPage.verifyErrorVisible()
    loginPage.verifyErrorMessage('Username is required')
})

// Test Scenario: Nhập vào username quá ngắn
it('Hành động khi username quá ngắn', () => {
    loginPage.login('ad', 'SomePassword123')

    loginPage.verifyErrorVisible()
    loginPage.verifyErrorMessage('Username must be at least 3 characters')
})

// Test Scenario: Nhập vào username quá dài
it('Hành động khi username quá dài', () => {
    loginPage.login(
        'averyveryveryveryveryveryveryveryveryveryveryveryveryveryverylongusername',
        'SomePassword123')

    loginPage.verifyErrorVisible()
    loginPage.verifyErrorMessage('Username must be 50 characters')
})

// Test Scenario: username chứa ký tự đặc biệt không đúng
it('Hành động khi username chứa ký tự đặc biệt không đúng', () => {
    loginPage.login('@invalid user!', 'SomePassword123')

    loginPage.verifyErrorVisible()
    loginPage.verifyErrorMessage('Username contains invalid characters')
})
```

```
// Test Scenario: password quá ngắn
it('ẽHìn_ìth_õlì_khi_password_quá_ắgn', () => {
  loginPage.login('validuser', 'a')

  loginPage.verifyErrorVisible()
  loginPage.verifyErrorMessage('Password_must_be_6_characters')
})

// Test Scenario: Đăng ậnp fail
it('ẽHìn_ìth_õlì_khi_ậnp_sai_thông_tin', () => {
  loginPage.login('wronguser', 'wrongpass')

  loginPage.verifyErrorVisible()
  loginPage.verifyErrorMessage('Password_must_include_both_letters_and_numbers')
})
```

c) Test success/error flows

```
// Test Scenario: ậnp vào username không ồtn ạti
it('ẽHìn_ìth_õlì_khi_username_không_ồtn_ạti', () => {
  loginPage.interceptLoginFailure()
  loginPage.login('nonexistentuser', 'SomePassword123')

  loginPage.waitForLoginFailure()
  loginPage.verifyErrorVisible()
})

// Test Scenario: Username / ậmt ắkhư ứcha ắkhong ắtrng ởđầ u ặhoc ồcui - ựt
động trim và đăng ậnp thành công
it('ựT_động_trim_ắkhong_ắtrng_và_đăng_ậnp_thành_công', () => {
  loginPage.interceptLoginSuccess()
  loginPage.fillUsername('admin')
  loginPage.fillPassword('Admin123')
  loginPage.clickLogin()

  loginPage.waitForLoginRequest()
  .its('response.statusCode')
  .should('eq', 200)

  loginPage.verifyErrorNotExist()
})
```

d) Test UI elements interactions

```
import LoginPage from '../support/page-objects/LoginPage'

describe('Login_E2E_Tests', () => {
  const loginPage = new LoginPage()

  beforeEach(() => {
    loginPage.visit()
  })

  // ---Test UI Elements ---
  it('ẽHìn_ìth_form_đăng_ậnp_đầy_ủ', () => {
    loginPage.verifyUsernameVisible()
    loginPage.verifyPasswordVisible()
    loginPage.verifyLoginButtonVisible()
  })
})
```

e) Cypress UI:

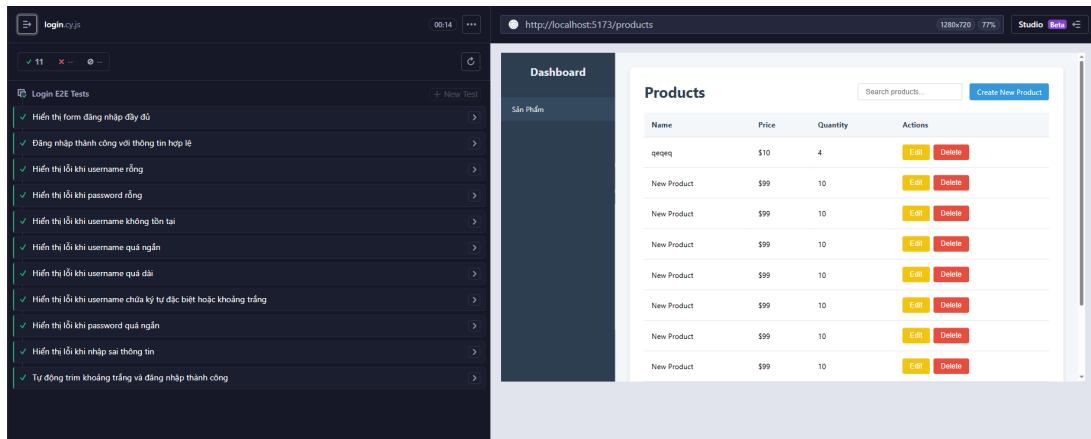


Figure 1: Giao diện kiểm thử Cypress cho Login

6.1.3 CI/CD Integration cho Login Tests

- GitHub Actions workflow:

```
name: Login Tests CI

on:
  push:
    branches: [ main, develop, lam ]
  pull_request:
    branches: [ main, lam ]

jobs:
  test-login:
    runs-on: ubuntu-latest

    steps:
      - name: Checkout repository
        uses: actions/checkout@v4

      - name: Setup Node.js
        uses: actions/setup-node@v4
        with:
          node-version: 20
          cache: 'npm'
          cache-dependency-path: frontend/package-lock.json

      - name: Install dependencies
        run: |
          cd frontend
          npm ci

      - name: Run Unit & Integration Tests (Vitest)
        run: |
          cd frontend
          npm test -- run

      # Build production (Optional)
      - name: Build frontend
        run: |
          cd frontend
          npm run build

      # Run E2E Test (Cypress)
      - name: Run E2E Tests
        uses: cypress-io/github-action@v6
```

```

with:
  working-directory: frontend
  start: npm run dev
  wait-on: 'http://localhost:5173'
  wait-on-timeout: 120
  spec: cypress/e2e/login.cy.js

# Upload Artifacts
- name: Upload Cypress videos and screenshots
  if: failure()
  uses: actions/upload-artifact@v4
  with:
    name: cypress-artifacts
    path: |
      frontend/cypress/videos
      frontend/cypress/screenshots

```

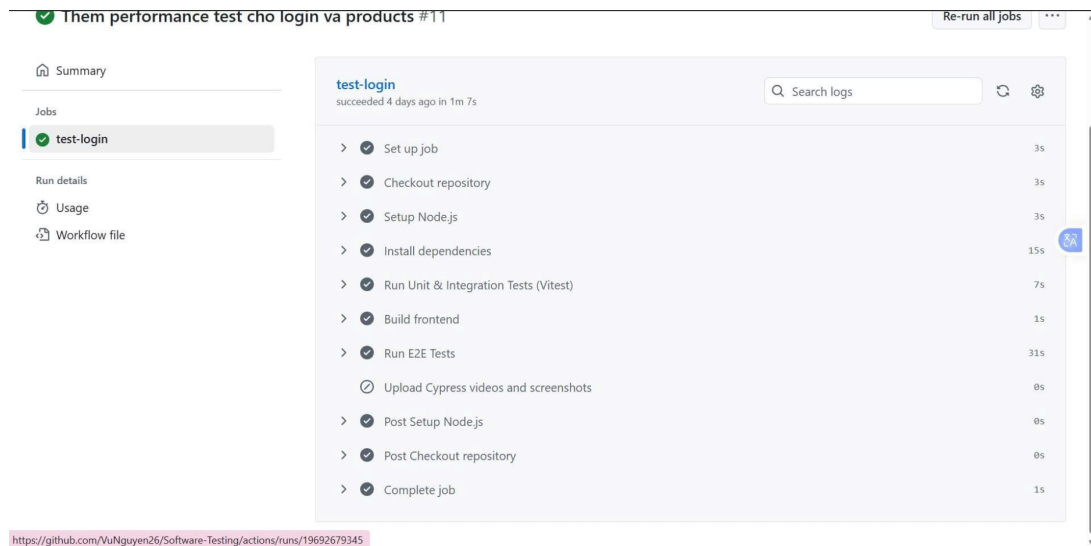
- Run login tests automatically

The screenshot shows the GitHub Actions interface for a workflow named 'login-tests.yml'. On the left, there's a sidebar with 'Actions' and a list of workflow runs. The main area displays a table of 11 workflow runs. Each row includes a status icon (green for success, red for failure), a title, a link to the run, a branch, and a duration. The runs are sorted by time, with the most recent at the top.

Status	Title	Link	Branch	Duration
Success	Them performance test cho login va products	Login Tests CI #11: Commit 4057189 pushed by LamLam44	1am	1m 10s
Success	Fix lai pass cho test case API loi cua login integrat...	Login Tests CI #10: Commit 5618f3d pushed by LamLam44	1am	1m 42s
Failure	them nhanh lam vao CI/CD	Login Tests CI #9: Commit f1cc87f pushed by LamLam44	1am	36s
Success	chore(ci): enable Cypress video & screenshot artif...	Login Tests CI #8: Commit 51c5086 pushed by VuNguyen26	main	59s
Success	fix(e2e): mock login API for CI to always return 200	Login Tests CI #7: Commit 64ae975 pushed by VuNguyen26	main	1m 17s
Failure	fix(e2e): update cy.visit port to 4173 for Vite previ...	Login Tests CI #6: Commit b6902bf pushed by VuNguyen26	main	1m 9s

The screenshot shows the details of a specific workflow run for 'login-tests.yml'. The top section shows the run was triggered via push 4 days ago, by user 'LamLam44', with commit '4057189', and it was successful with a total duration of '1m 10s'. Below this, the 'login-tests.yml' file is shown with the 'on: push' trigger and a single job 'test-login' with a duration of '1m 7s'. At the bottom, there's a 'test-login summary' section with 'Cypress Results'.

Result	Passed	Failed	Pending	Skipped	Duration
Passing	5	0	0	0	3.715s



6.2 Product - E2E Automation Testing

6.2.1 Setup Page Object Model

```
// Page Object Model for Product Page
class ProductPage {
  // Selectors
  get addProductButton() {
    return cy.get('[data-testid="add-product-button"]')
  }

  get productList() {
    return cy.get('[data-testid="product-list"]')
  }

  get productItems() {
    return cy.get('[data-testid="product-item"]')
  }

  get productNameInput() {
    return cy.get('[data-testid="product-name-input"]')
  }

  get productQuantityInput() {
    return cy.get('[data-testid="product-quantity-input"]')
  }

  get productPriceInput() {
    return cy.get('[data-testid="product-price-input"]')
  }

  get productDescriptionInput() {
    return cy.get('[data-testid="product-description-input"]')
  }

  get productCategoryInput() {
    return cy.get('[data-testid="product-category-input"]')
  }

  get saveProductButton() {
    return cy.get('[data-testid="save-product-button"]')
  }
}
```

```

get validationError() {
    return cy.get('[data-testid="validation-error"]')
}

get notification() {
    return cy.get('[data-testid="notification"]')
}

getEditButton() {
    return cy.get('[data-testid="edit-product-button"]')
}

getDeleteButton() {
    return cy.get('[data-testid="delete-product-button"]')
}

// Actions
visit() {
    cy.visit('/')
}

clickAddProduct() {
    this.addProductButton.click()
}

fillProductName(name) {
    this.productNameInput.clear().type(name)
}

fillProductQuantity(quantity) {
    this.productQuantityInput.type(quantity)
}

fillProductPrice(price) {
    this.productPriceInput.type(price)
}

clearProductPrice() {
    this.productPriceInput.clear()
}

fillProductDescription(description) {
    this.productDescriptionInput.clear().type(description)
}

selectProductCategory(category) {
    this.productCategoryInput.select(category)
}

clickSaveProduct() {
    this.saveProductButton.click()
}

clearProductName() {
    this.productNameInput.clear()
}

addProduct(name, quantity, price, description, category) {
    this.fillProductName(name)
    this.fillProductQuantity(quantity)
    this.fillProductPrice(price)
    this.fillProductDescription(description)
    this.selectProductCategory(category)
}

```

```

    this.clickSaveProduct()
  }

  clickEditFirstProduct() {
    this.productItems.first().within(() => {
      this.getEditButton().click()
    })
  }

  clickDeleteFirstProduct() {
    this.productItems.first().within(() => {
      this.getDeleteButton().click()
    })
    cy.on('window:confirm', () => true)
  }

  // Assertions
  verifyProductListVisible() {
    this.productList.should('be.visible')
  }

  verifyProductItemsCount(count) {
    this.productItems.its('length').should('be.gte', count)
  }

  verifyValidationErrorExists() {
    this.validationError.should('exist')
  }

  verifyValidationErrorNotExist() {
    this.validationError.should('not.exist')
  }

  verifyUrlIncludesProductsNew() {
    cy.url().should('include', '/products/new')
  }

  verifyUrlIncludesProducts() {
    cy.url().should('include', '/products')
  }

  verifyProductNameValue(name) {
    this.productNameInput.should('have.value', name)
  }

  // API Intercepts
  interceptGetProducts(products) {
    cy.intercept('GET', '**/api/products', {
      statusCode: 200,
      body: products
    }).as('getProducts')
  }

  interceptCreateProduct(product) {
    cy.intercept('POST', '**/api/products', {
      statusCode: 201,
      body: product
    }).as('createProduct')
  }

  interceptUpdateProduct(productId, product) {
    cy.intercept('PUT', `**/api/products/${productId}`, {
      statusCode: 200,

```

```

        body: product
    }).as('updateProduct')
}

interceptDeleteProduct(productId) {
    cy.intercept('DELETE', `**/api/products/${productId}`, {
        statusCode: 200,
        body: { id: productId }
    }).as('deleteProduct')
}

interceptGetProductById(productId, product) {
    cy.intercept('GET', `**/api/products/${productId}`, {
        statusCode: 200,
        body: product
    }).as('getProduct')
}

interceptLogin() {
    cy.intercept('POST', '**/api/auth/login', {
        statusCode: 200,
        body: { token: 'fake-token-123', user: { name: 'Admin' } }
    }).as('loginRequest')
}

// Wait methods
waitForGetProducts() {
    return cy.wait('@getProducts', { timeout: 5000 })
}

waitForCreateProduct() {
    return cy.wait('@createProduct', { timeout: 5000 })
}

waitForUpdateProduct() {
    return cy.wait('@updateProduct', { timeout: 5000 })
}

waitForDeleteProduct() {
    return cy.wait('@deleteProduct', { timeout: 5000 })
}

waitForGetProduct() {
    return cy.wait('@getProduct', { timeout: 5000 })
}

waitForLogin() {
    return cy.wait('@loginRequest', { timeout: 5000 })
}

// Additional helper methods
clearAndTypeProductName(name) {
    this.productNameInput.clear().type(name)
}

clearAndTypeProductPrice(price) {
    this.productPriceInput.clear().type(price)
}

typeProductPrice(price) {
    this.productPriceInput.type(price)
}

```

```

verifyProductItemsExist() {
  this.productItems.its('length').should('be.gte', 1)
}

verifyNoValidationError() {
  cy.get('body').then($body => {
    if ($body.find('[data-testid="validation-error"]').length > 0) {
      throw new Error('Validation error should not exist')
    }
  })
}

// Search functionality
get searchInput() {
  return cy.get('[data-testid="search-input"]')
}

searchProduct(searchTerm) {
  this.searchInput.clear().type(searchTerm)
}

clearSearch() {
  this.searchInput.clear()
}

verifyProductItemsCountEquals(count) {
  this.productItems.should('have.length', count)
}

verifyProductContainsName(name) {
  this.productItems.should('contain', name)
}

verifyProductDoesNotContainName(name) {
  this.productItems.should('not.contain', name)
}

// Login helper for beforeEach
loginAsAdmin() {
  cy.get('[data-testid="username-input"]').clear().type('admin')
  cy.get('[data-testid="password-input"]').clear().type('Admin123')
  cy.get('[data-testid="login-button"]').click()
  this.waitForLogin()
  cy.get('[data-testid="login-error"]').should('not.exist')
}

export default ProductPage

```

6.2.2 E2E Test Scenarios cho Product

a) Test Create product flow

```

import ProductPage from '../support/page-objects/ProductPage'

describe('Product E2E Tests', () => {
  const productPage = new ProductPage()

  beforeEach(() => {
    productPage.interceptGetProducts([
      { id: 1, name: 'Product_1', quantity: 5, price: 29.99, description: 'First product', category: 'SPORT' },
      { id: 2, name: 'Product_2', quantity: 10, price: 49.99, description: 'Second product', category: 'ELECTRONICS' },
    ])
  })

```

```

    })

    productPage.interceptLogin()

    productPage.visit()
    productPage.loginAsAdmin()
    productPage.waitForGetProducts()
  })
  it('should add a new product', () => {
    productPage.interceptCreateProduct({
      id: 999,
      name: 'New Product',
      quantity: 10,
      price: 99.99,
      description: 'This is a new product.',
      category: 'SPORT'
    })

    productPage.clickAddProduct()
    productPage.verifyUrlIncludesProductsNew()
    productPage.addProduct('New Product', '10', '99.99', 'This is a new product.', 'SPORT')

    productPage.waitForCreateProduct()
    productPage.verifyUrlIncludesProducts()
  })

```

b) Test Read/List products

```

it('should display the product list', () => {
  productPage.verifyProductListVisible()
  productPage.verifyProductItemsExist()
})
// Test Scenario: Thêm sản phẩm mới với thông tin không hợp lệ
it('should show validation errors when adding a product with invalid data',
  () => {
    productPage.clickAddProduct()
    productPage.addProduct('Test Product', '10', '0', 'This is a new product.', 'ELECTRONICS')
    productPage.verifyValidationErrorExists()
  })

// Test Scenario: Tên sản phẩm trống khi thêm
it('should show validation errors when product name is empty', () => {
  productPage.clickAddProduct()
  productPage.clearProductName()
  productPage.typeProductPrice('50')
  productPage.clickSaveProduct()
  productPage.verifyValidationErrorExists()
})

// Test Scenario: Tên sản phẩm quá ngắn
it('should show validation errors when product name is too short', () => {
  productPage.clickAddProduct()
  productPage.addProduct('AB', '10', '50', 'This is a new product.', 'SPORT')
  productPage.verifyValidationErrorExists()
})

// Test Scenario: Tên sản phẩm quá dài
it('should show validation errors when product name is too long', () => {
  productPage.clickAddProduct()
  productPage.addProduct('A'.repeat(101), '10', '50', 'This is a new product.', 'ELECTRONICS')

```

```

    productPage.verifyValidationErrorExists()
  })

  // Test Scenario: Description sản phẩm quá dài
  it('should show validation errors when product description is too long', ()
    => {
    productPage.clickAddProduct()
    productPage.addProduct('Valid Name', '10', '50', 'D'.repeat(501), '
      ELECTRONICS')
    productPage.verifyValidationErrorExists()
  })

  // Test Scenario: Sản phẩm vượt  m   giới
  it('should show validation errors when product price exceeds max limit', ()
    => {
    productPage.clickAddProduct()
    productPage.addProduct('Expensive Product', '5', '999999999', 'This is
      an expensive product.', 'ELECTRONICS')
    productPage.verifyValidationErrorExists()
  })

```

c) Test Update product

```

it('should edit a product', () => {
  productPage.interceptGetProductById(1, { id: 1, name: 'Product 1',
    quantity: 5, price: 29.99, description: 'First product', category
      : 'SPORT' })

  productPage.interceptUpdateProduct(1, { id: 1, name: 'Updated Product
    Name', quantity: 5, price: 29.99, description: 'First product',
      category: 'SPORT' })

  productPage.interceptGetProducts([
    { id: 1, name: 'Updated Product Name', quantity: 5, price: 29.99,
      description: 'First product', category: 'SPORT' }
  ])

  productPage.clickEditFirstProduct()
  productPage.waitForGetProduct()
  productPage.clearAndTypeProductName('Updated Product Name')
  productPage.clickSaveProduct()

  productPage.waitForUpdateProduct()
})

// Test Scenario:  sả sản phẩm  vi thông tin không  p h 
it('should show validation errors when editing a product with invalid data'
  , () => {
  productPage.interceptGetProductById(1, { id: 1, name: 'Product 1',
    quantity: 5, price: 29.99, description: 'First product', category:
      'SPORT' })

  productPage.clickEditFirstProduct()
  productPage.waitForGetProduct()
  productPage.clearAndTypeProductName('Updated Product')
  productPage.clearAndTypeProductPrice('0')
  productPage.clickSaveProduct()
  productPage.verifyValidationErrorExists()
})

// Test Scenario:  sả sản phẩm không thay   i g 
it('should not change product when editing without modifications', () => {
  productPage.interceptGetProductById(1, { id: 1, name: 'Product 1',
    quantity: 5, price: 29.99, description: 'First product', category:

```

```

        'SPORT' })

    productPage.interceptUpdateProduct(1, { id: 1, name: 'Product_1',
        quantity: 5, price: 29.99, description: 'First_product', category:
        'SPORT' })

    productPage.clickEditFirstProduct()
    productPage.waitForGetProduct()
    productPage.verifyProductNameValue('Product_1')
    productPage.clickSaveProduct()
    productPage.waitForUpdateProduct()
    productPage.verifyUrlIncludesProducts()
})

```

d) Test Delete product

```

it('should_delete_a_product', () => {
    cy.intercept('DELETE', '**/api/products/**', {
        statusCode: 200,
        body: {}
    }).as('deleteProduct')

    productPage.interceptGetProducts([
        { id: 2, name: 'Product_2', quantity: 10, price: 49.99, description
        : 'Second_product', category: 'ELECTRONICS' },
    ])

    productPage.clickDeleteFirstProduct()

    cy.wait('@deleteProduct', { timeout: 5000 })
})

```

e) Test Search/Filter functionality

```

// Test Scenario: Tìm kiếm sản phẩm theo tên
it('should_filter_products_when_searching_by_name', () => {
    productPage.searchProduct('Product_1')
    productPage.verifyProductItemsCountEquals(1)
    productPage.verifyProductContainsName('Product_1')
    productPage.verifyProductDoesNotContainName('Product_2')
})

// Test Scenario: Tìm kiếm sản phẩm không tồn tại
it('should_show_no_products_when_searching_for_non-existent_product', () => {
    {
        productPage.searchProduct('NonExistentProduct')
        productPage.verifyProductItemsCountEquals(0)
    }
})

```

f) Cypress UI:

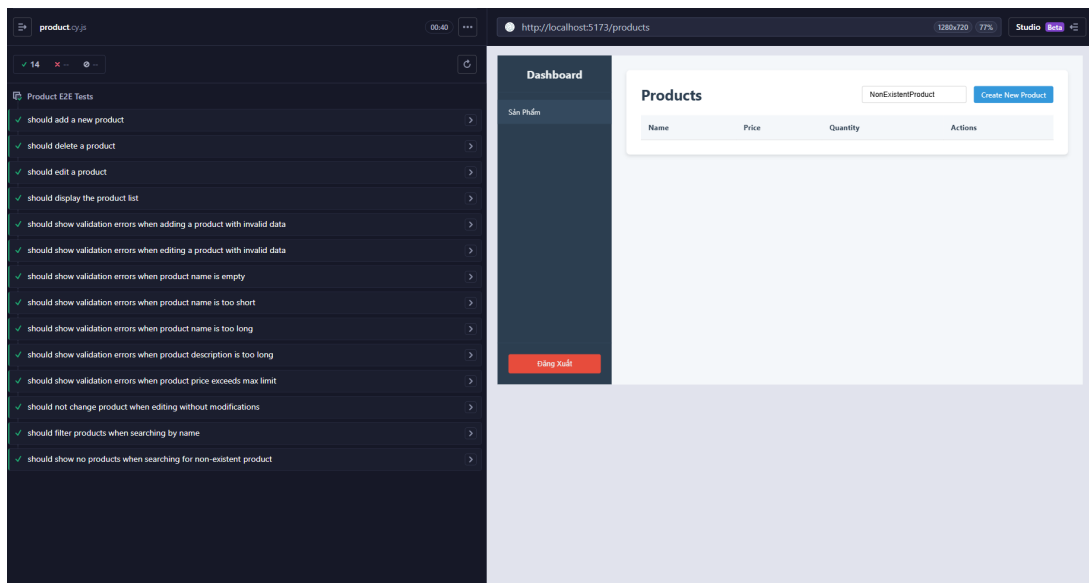


Figure 2: Giao diện kiểm thử Cypress cho Product

6.2.3 CI/CD Integration

- GitHub Actions workflow:

```
name: Product Feature CI/CD

on:
  push:
    branches: [ "main" ]
  pull_request:
    branches: [ "main" ]

jobs:
  test-product-feature:
    runs-on: ubuntu-latest

    steps:
      - name: Checkout code
        uses: actions/checkout@v4

      - name: Set up JDK 17
        uses: actions/setup-java@v4
        with:
          java-version: '17'
          distribution: 'temurin'
          cache: maven

      - name: Set up Node.js 18
        uses: actions/setup-node@v4
        with:
          node-version: '18'
          cache: 'npm'
          cache-dependency-path: frontend/package-lock.json

      - name: Run Backend Product Tests
        run: |
          cd backend
          chmod +x mvnw
          ./mvnw clean test -Dtest=*Product*

      - name: Install Frontend Dependencies
```

```

    run: |
      cd frontend
      npm ci

- name: Run Frontend Unit Tests (Product Only)
  run: |
    cd frontend
    npm test -- Product

- name: Run Cypress Product Tests
  uses: cypress-io/github-action@v6
  with:
    working-directory: frontend
    start: npm run dev
    wait-on: 'http://localhost:5173'
    spec: cypress/e2e/product.cy.js

```

7 Phần Mở Rộng

7.1 Performance Testing

- Setup k6 cho performance testing

```

# Choco (Windows)
choco install k6 -y

# Or Scoop
scoop install k6

```

- performance tests cho Login API

```

import http from 'k6/http';
import { check, sleep } from 'k6';

// ấCu hình Load Test & Stress Test
export const options = {
  stages: [
    // 1. Ramp up: 100 users 30s
    { duration: '30s', target: 100 },

    // 2. Load Test: Keep 100 users in 1 minute
    { duration: '1m', target: 100 },

    // 3. Stress Test: --> 500 --> 1000 users
    { duration: '1m', target: 500 },
    { duration: '1m', target: 1000 },

    // 4. Ramp down (ạH ệnhit): to 0
    { duration: '30s', target: 0 },
  ],

  thresholds: {
    http_req_duration: ['p(95)<500'], // 95%
    http_req_failed: ['rate<0.01'],
  },
};

export default function () {
  // URL API Login backend (uLu ý: if using localhost use local IP or host.
  docker.internal)
  const url = 'http://localhost:8080/api/auth/login';

  const payload = JSON.stringify({

```

```

    username: 'admin',
    password: 'Admin123',
  });

  const params = {
    headers: {
      'Content-Type': 'application/json',
    },
  };

  // send Request
  const res = http.post(url, payload, params);

  //(Check)
  check(res, {
    'status_is_200': (r) => r.status === 200, // Login thành công
    'token_exists': (r) => r.json('token') !== '', // return token
  });

  sleep(1);
}

```

- Viết performance tests cho Product API

```

import http from 'k6/http';
import { check, sleep } from 'k6';

export const options = {
  stages: [
    { duration: '2m', target: 500 },
    { duration: '2m', target: 1500 },
    { duration: '1m', target: 1500 },
    { duration: '1m', target: 0 },
  ],
  thresholds: {
    http_req_duration: ['p(95)<1000'],
    http_req_failed: ['rate<0.01'],
  },
};

export default function () {
  const loginUrl = 'http://localhost:8080/api/auth/login';
  const payload = JSON.stringify({
    username: 'admin',
    password: 'Admin123',
  });

  const params = {
    headers: {
      'Content-Type': 'application/json',
    },
  };

  const loginRes = http.post(loginUrl, payload, params);

  if (loginRes.status !== 200) {
    console.error('Login_failed');
    return;
  }
}

```

```

const token = loginRes.json('token');

const productUrl = 'http://localhost:8080/api/products';

const productParams = {
  headers: {
    'Content-Type': 'application/json',
    'Authorization': `Bearer ${token}`,
  },
};

const res = http.get(productUrl, productParams);

check(res, {
  'status_is_200': (r) => r.status === 200,
  'loaded_products': (r) => r.json().length >= 0,
});

sleep(1);
}

```

8 Test coverage

FloginBE

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
com.sgu.login.dto		21%		n/a	22	29	37	49	22	29	3	5
com.sgu.login.model		78%		n/a	6	22	6	24	6	22	0	2
com.sgu.login.config		0%		0%	4	4	5	5	3	3	1	1
com.sgu.login.service		95%		95%	2	25	1	45	1	13	0	2
com.sgu.login		37%		n/a	1	2	2	3	1	2	0	1
com.sgu.login.controller		100%		75%	1	10	0	19	0	8	0	2
Total	163 of 566	71%	4 of 30	86%	36	92	51	145	33	77	4	13

com.sgu.login.service

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
AuthService		91%		95%	2	16	1	26	1	4	0	1
ProductService		100%		n/a	0	9	0	19	0	9	0	1
Total	10 of 207	95%	1 of 24	95%	2	25	1	45	1	13	0	2

com.sgu.login.controller

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
ProductController		100%		75%	1	8	0	13	0	6	0	1
AuthController		100%		n/a	0	2	0	6	0	2	0	1
Total	0 of 99	100%	1 of 4	75%	1	10	0	19	0	8	0	2

Test Files	15 passed (15)				
Tests	57 passed (57)				
Start at	23:19:34				
Duration	5.17s (transform 1.25s, setup 3.87s, collect 4.13s, tests 2.61s, environment 22.88s, prepare 3.46s)				
% Coverage report from v8					
File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Line #s
All files	94.93	86	89.28	94.93	
src	100	100	100	100	
App.jsx	100	100	100	100	
main.jsx	100	100	100	100	
src/components	92.43	81.48	80	92.43	
DashboardLayout.jsx	100	100	100	100	
LoginForm.jsx	96.72	88.23	100	96.72	35-36
ProductDetail.jsx	100	87.5	100	100	11
ProductForm.jsx	89.04	68.42	100	89.04	16-19,32-33,37-38
ProductList.jsx	87.32	87.5	40	87.32	22-30
Sidebar.jsx	100	100	100	100	
src/services	100	0	100	100	
api.js	100	0	100	100	2
authService.js	100	100	100	100	
productService.js	100	100	100	100	
src/utlis	100	92.5	100	100	
validateLogin.js	100	92.59	100	100	8,20
validateProduct.js	100	92.3	100	100	2

9 Test results

STT	Tên File Test (Test Class)	Loại Kiểm Thử (Test Level)	Chức năng (Feature)	Số lượng Test Case	Trạng thái (Status)
1	ValidateInputTest	Unit Test	Login / Common	12	Passed
2	AuthServiceTest	Unit Test	Login	11	Passed
3	ProductServiceTest	Unit Test	Product	8	Passed
4	AuthControllerMockTest	Mock Test	Login	6	Passed
5	ProductServiceMockTest	Mock Test	Product	11	Passed
6	AuthControllerIntegrationTest	Integration Test	Login	2	Passed
7	ProductControllerIntegrationTest	Integration Test	Product	6	Passed
-	TỔNG CỘNG	Toàn bộ Backend	All	56	100% Passed