TRƯỜNG ĐẠI HỌC SÀI GÒN
KHOA CÔNG NGHỆ THÔNG TIN

KIỂM THỬ PHẦN MỀM

BÁO CÁO BÀI TẬP LỚN

Ứng dụng Đăng nhập & Quản lý Sản phẩm

(FloginFE_BE)

GVHD: Từ Lãng Phiêu
SV: Đào Nhị Khang - 3122410168
Mai Văn Minh Trí - 3123410389
Nguyễn Minh Trí - 3122560082
Trần Thị Tình - 3122410414
Đào Song Lộc - 3123410200
Cao Minh Triết - 3123410391

TP. HỒ CHÍ MINH, NGÀY 2 THÁNG 12 NĂM 2025

Mục lục

LỜI MỞ ĐẦU

Kiểm thử phần mềm là một phần không thể thiếu trong quy trình phát triển phần mềm chuyên nghiệp. Trong bối cảnh công nghệ phát triển nhanh chóng, việc đảm bảo chất lượng sản phẩm phần mềm trở nên quan trọng hơn bao giờ hết. Bài tập lớn này nhằm giúp sinh viên nắm vững các kỹ thuật kiểm thử hiện đại và áp dụng vào thực tế.

Giới thiệu đề tài

Đề tài được lựa chọn là ứng dụng **FloginFE_BE** - một hệ thống web full-stack hoàn chỉnh với các thành phần chính:

- **Frontend**: ReactJS 19 với React Router, Axios
- **Backend**: Spring Boot 3.3.5 với Spring Security, JWT Authentication
- **Database**: MySQL 8.0
- **Chức năng chính**:
  - Authentication: Login, Register với JWT token
  - Product Management: CRUD operations với phân quyền
  - Image upload và validation

Qua đề tài này, nhóm có cơ hội thực hành đầy đủ các loại kiểm thử từ Unit Testing, Integration Testing, Mock Testing đến Automation Testing, Performance Testing và Security Testing.

Báo cáo này trình bày chi tiết quá trình thực hiện các yêu cầu của bài tập lớn, bao gồm:

- Phân tích và thiết kế Test Cases
- Unit Testing với phương pháp Test-Driven Development (TDD)
- Integration Testing
- Mock Testing
- Automation Testing và CI/CD
- Performance Testing và Security Testing (phần mở rộng)

Nhóm xin chân thành cảm ơn thầy Từ Lãng Phiêu đã hướng dẫn tận tình trong suốt quá trình thực hiện bài tập lớn này.

TP. Hồ Chí Minh, ngày 2 tháng 12 năm 2025

1 Phân tích và Thiết kế Test Cases

1.1 Giới thiệu chương

Chương này trình bày quá trình phân tích yêu cầu và thiết kế test cases cho hệ thống FloginFE_BE. Test Case Design là bước quan trọng đầu tiên trong quy trình kiểm thử phần mềm, giúp xác định các kịch bản kiểm thử cần thiết để đảm bảo chất lượng sản phẩm.

Nội dung chính của chương:

- Phân tích yêu cầu chức năng Login và Product
- Thiết kế Test Scenarios cho từng chức năng
- Xây dựng Test Cases chi tiết với input/output cụ thể
- Xác định Test Data và Expected Results

1.2 Login - Phân tích và Thiết kế Test Scenarios

1.2.1 Yêu cầu chức năng

Chức năng Đăng nhập (Login) cho phép người dùng truy cập vào hệ thống quản lý sản phẩm. Yêu cầu chức năng bao gồm:

Yêu cầu nghiệp vụ:

- Người dùng phải đăng nhập để truy cập hệ thống
- Hệ thống xác thực username và password
- Sau khi đăng nhập thành công, chuyển hướng đến trang Quản lý sản phẩm
- Hệ thống lưu JWT token để duy trì phiên đăng nhập

Yêu cầu kỹ thuật:

- **Username**: Bắt buộc nhập, 3-50 ký tự
- **Password**: Bắt buộc nhập, 6-100 ký tự, phải chứa cả chữ và số
- **API Endpoint**: POST /api/auth/login
- **Response**: JSON với token và message

1.2.2 Test Scenarios

Dựa trên phân tích yêu cầu, nhóm xác định các Test Scenarios sau:

1. **TS_LOGIN_01**: Kiểm tra validation dữ liệu đầu vào
    - Username rỗng
    - Password rỗng
    - Username không đủ độ dài (< 3 ký tự)
    - Password không đủ độ dài (< 6 ký tự)
    - Password không chứa số

2. **TS_LOGIN_02**: Kiểm tra xác thực người dùng
   - Username không tồn tại
   - Password sai
   - Username và password đúng
3. **TS_LOGIN_03**: Kiểm tra luồng thành công
   - Lưu token vào localStorage
   - Chuyển hướng đến trang /product
   - Hiển thị thông báo thành công

1.2.3 Thiết kế Test Cases chi tiết

Bảng sau liệt kê các Test Cases chi tiết cho chức năng Login theo định dạng chuẩn:

TC_LOGIN_001: Kiểm tra username rỗng hoặc chỉ chứa khoảng trắng

Test Case ID	TC_LOGIN_001
Test Name	Kiểm tra username rỗng hoặc khoảng trắng
Priority	High
Preconditions	- Ứng dụng đang chạy - Người dùng đang ở trang đăng nhập
Test Steps	1. Truy cập trang đăng nhập 2. Để trống trường username hoặc chỉ nhập khoảng trắng 3. Nhập mật khẩu hợp lệ 4. Nhấn nút Đăng nhập
Test Data	Username: "" hoặc " " Password: Test123
Expected Result	- Hiển thị lỗi: "Tên đăng nhập không được để trống" - Không gọi API đăng nhập - Form không submit
Actual Result	(Để trống)
Status	Not Run

TC_LOGIN_002: Kiểm tra username quá ngắn ($<$ 3 ký tự)

Test Case ID	TC_LOGIN_002
Test Name	Kiểm tra username quá ngắn
Priority	Medium
Preconditions	- Ứng dụng đang chạy - Người dùng đang ở trang đăng nhập
Test Steps	1. Truy cập trang đăng nhập 2. Nhập username có 2 ký tự 3. Nhập mật khẩu hợp lệ 4. Nhấn nút Đăng nhập
Test Data	Username: ab Password: Test123
Expected Result	- Hiển thị lỗi: "Tên đăng nhập phải có ít nhất 3 ký tự" - Không gọi API đăng nhập - Form không submit
Actual Result	(Để trống)
Status	Not Run

TC_LOGIN_003: Kiểm tra username quá dài (> 50 ký tự)

Test Case ID	TC_LOGIN_003
Test Name	Kiểm tra username vượt quá độ dài cho phép
Priority	Medium
Preconditions	- Ứng dụng đang chạy - Người dùng đang ở trang đăng nhập
Test Steps	1. Truy cập trang đăng nhập 2. Nhập username có 51 ký tự 3. Nhập mật khẩu hợp lệ 4. Nhấn nút Đăng nhập
Test Data	Username: (51 ký tự) Password: Test123
Expected Result	- Hiển thị lỗi: "Tên đăng nhập không được quá 50 ký tự" - Không gọi API đăng nhập - Form không submit
Actual Result	(Để trống)
Status	Not Run

TC_LOGIN_004: Kiểm tra username chứa ký tự đặc biệt hoặc khoảng trắng

Test Case ID	TC_LOGIN_004
Test Name	Kiểm tra username chứa ký tự không hợp lệ
Priority	Medium
Preconditions	- Ứng dụng đang chạy - Người dùng đang ở trang đăng nhập
Test Steps	1. Truy cập trang đăng nhập 2. Nhập username chứa ký tự đặc biệt hoặc khoảng trắng 3. Nhập mật khẩu hợp lệ 4. Nhấn nút Đăng nhập
Test Data	Username: user@123 hoặc "test user" Password: Test123
Expected Result	- Hiển thị lỗi: "Tên đăng nhập chỉ chứa chữ cái và số" - Không gọi API đăng nhập - Form không submit
Actual Result	(Để trống)
Status	Not Run

TC_LOGIN_005: Kiểm tra username hợp lệ

Test Case ID	TC_LOGIN_005
Test Name	Đăng nhập thành công với username hợp lệ
Priority	Critical
Preconditions	- Ứng dụng đang chạy - Người dùng đang ở trang đăng nhập - Tài khoản tồn tại trong hệ thống
Test Steps	1. Truy cập trang đăng nhập 2. Nhập username hợp lệ (chỉ chữ và số, 3-50 ký tự) 3. Nhập mật khẩu hợp lệ 4. Nhấn nút Đăng nhập
Test Data	Username: testuser1 hoặc ADMIN Password: Test123

Expected Result	- Không có lỗi validation cho username - Form có thể gọi API - Chuyển đến trang chủ sau khi đăng nhập thành công
Actual Result	(Để trống)
Status	Not Run

TC_LOGIN_006: Kiểm tra password rỗng hoặc chỉ chứa khoảng trắng

Test Case ID	TC_LOGIN_006
Test Name	Kiểm tra password rỗng hoặc khoảng trắng
Priority	High
Preconditions	- Ứng dụng đang chạy - Người dùng đang ở trang đăng nhập
Test Steps	1. Truy cập trang đăng nhập 2. Nhập username hợp lệ 3. Để trống trường password hoặc chỉ nhập khoảng trắng 4. Nhấn nút Đăng nhập
Test Data	Username: testuser Password: "" hoặc " "
Expected Result	- Hiển thị lỗi: "Mật khẩu không được để trống" - Không gọi API đăng nhập - Form không submit
Actual Result	(Để trống)
Status	Not Run

TC_LOGIN_007: Kiểm tra password quá ngắn ($<$ 6 ký tự)

Test Case ID	TC_LOGIN_007
Test Name	Kiểm tra password quá ngắn
Priority	Medium
Preconditions	- Ứng dụng đang chạy - Người dùng đang ở trang đăng nhập
Test Steps	1. Truy cập trang đăng nhập 2. Nhập username hợp lệ 3. Nhập password có 5 ký tự 4. Nhấn nút Đăng nhập
Test Data	Username: testuser Password: 12345
Expected Result	- Hiển thị lỗi: "Mật khẩu phải có ít nhất 6 ký tự" - Không gọi API đăng nhập - Form không submit
Actual Result	(Để trống)
Status	Not Run

TC_LOGIN_008: Kiểm tra password quá dài ($>$ 100 ký tự)

Test Case ID	TC_LOGIN_008
Test Name	Kiểm tra password vượt quá độ dài cho phép
Priority	Low
Preconditions	- Ứng dụng đang chạy - Người dùng đang ở trang đăng nhập

Test Steps	1. Truy cập trang đăng nhập 2. Nhập username hợp lệ 3. Nhập password có 101 ký tự 4. Nhấn nút Đăng nhập
Test Data	Username: testuser Password: (101 ký tự)
Expected Result	- Hiển thị lỗi: "Mật khẩu không được quá 100 ký tự" - Không gọi API đăng nhập - Form không submit
Actual Result	(Để trống)
Status	Not Run

TC_LOGIN_009: Kiểm tra password thiếu chữ cái (chỉ có số)

Test Case ID	TC_LOGIN_009
Test Name	Kiểm tra password không chứa chữ cái
Priority	Medium
Preconditions	- Ứng dụng đang chạy - Người dùng đang ở trang đăng nhập
Test Steps	1. Truy cập trang đăng nhập 2. Nhập username hợp lệ 3. Nhập password chỉ có số (không có chữ) 4. Nhấn nút Đăng nhập
Test Data	Username: testuser Password: 12345678
Expected Result	- Hiển thị lỗi: "Mật khẩu phải chứa cả chữ cái và số" - Không gọi API đăng nhập - Form không submit
Actual Result	(Để trống)
Status	Not Run

TC_LOGIN_010: Kiểm tra password thiếu số (chỉ có chữ)

Test Case ID	TC_LOGIN_010
Test Name	Kiểm tra password không chứa số (chỉ có chữ)
Priority	Medium
Preconditions	- Ứng dụng đang chạy - Người dùng đang ở trang đăng nhập
Test Steps	1. Truy cập trang đăng nhập 2. Nhập username hợp lệ 3. Nhập password chỉ có chữ (không có số) 4. Nhấn nút Đăng nhập
Test Data	Username: testuser Password: abcdefgh
Expected Result	- Hiển thị lỗi: "Mật khẩu phải chứa cả chữ cái và số" - Không gọi API đăng nhập - Form không submit
Actual Result	(Để trống)
Status	Not Run

TC_LOGIN_011: Kiểm tra password hợp lệ

Test Case ID	TC_LOGIN_011

Test Name	Kiểm tra password hợp lệ hoàn toàn
Priority	Critical
Preconditions	- Ứng dụng đang chạy - Người dùng đang ở trang đăng nhập - Tài khoản tồn tại trong hệ thống
Test Steps	1. Truy cập trang đăng nhập 2. Nhập username hợp lệ 3. Nhập password hợp lệ (có cả chữ và số, 6-100 ký tự) 4. Nhấn nút Đăng nhập
Test Data	Username: testuser Password: Test1234
Expected Result	- Không có lỗi validation cho password - Form có thể gọi API - Đăng nhập thành công - Chuyển đến trang chủ
Actual Result	(Để trống)
Status	Not Run

1.2.4 Test Data

Tài khoản test có sẵn trong hệ thống:

Username	Password	Role
testuser	Test123	Admin
testuser1	Test1234	User
ADMIN	Admin123	Admin

Bảng 12: Test Data - Tài khoản đăng nhập hợp lệ

Invalid Test Data:

- **Username**: "" (rỗng), " " (khoảng trắng), "ab" (quá ngắn), (51 ký tự - quá dài), "user@123" (ký tự đặc biệt), "test user" (có khoảng trắng), "nonexistuser" (không tồn tại)
- **Password**: "" (rỗng), " " (khoảng trắng), "12345" (quá ngắn), (101 ký tự - quá dài), "12345678" (chỉ số), "abcdefgh" (chỉ chữ), "WrongPass1" (sai password)

1.3 Product - Phân tích và Thiết kế Test Scenarios

1.3.1 Yêu cầu chức năng

Chức năng Quản lý sản phẩm (Product) cho phép người dùng thực hiện các thao tác CRUD (Create, Read, Update, Delete) trên danh sách sản phẩm.

Yêu cầu nghiệp vụ:

- Hiển thị danh sách sản phẩm với đầy đủ thông tin
- Thêm mới sản phẩm với ảnh đại diện
- Chỉnh sửa thông tin sản phẩm
- Xóa sản phẩm
- Xem chi tiết sản phẩm

Yêu cầu kỹ thuật:

- **Product Name**: Bắt buộc, 3-100 ký tự
- **Price**: Bắt buộc, số dương (> 0), tối đa 999,999,999
- **Quantity**: Bắt buộc, số nguyên dương (> 0), tối đa 99,999
- **Description**: Tùy chọn, tối đa 500 ký tự
- **Category**: Bắt buộc phải chọn danh mục (ID hợp lệ)
- **Image**: Tùy chọn, hỗ trợ JPG, PNG, GIF

API Endpoints:

- GET /api/products - Lấy danh sách sản phẩm
- GET /api/products/{id} - Lấy chi tiết sản phẩm
- POST /api/products - Tạo sản phẩm mới
- PUT /api/products/{id} - Cập nhật sản phẩm
- DELETE /api/products/{id} - Xóa sản phẩm

1.3.2 Test Scenarios

Dựa trên phân tích yêu cầu, nhóm xác định các Test Scenarios sau cho CRUD operations:

1. **TS_PRODUCT_01**: Kiểm tra tạo sản phẩm mới (CREATE - 4 test cases)
   - Tạo sản phẩm với dữ liệu hợp lệ
   - Tạo sản phẩm với dữ liệu validation lỗi
   - Tạo sản phẩm thiếu trường bắt buộc (category)
   - Tạo sản phẩm với giá trị biên (tên dài nhất)
2. **TS_PRODUCT_02**: Kiểm tra đọc/lấy thông tin sản phẩm (READ - 4 test cases)
   - Lấy danh sách tất cả sản phẩm
   - Lấy chi tiết một sản phẩm theo ID thành công
   - Lấy sản phẩm với ID không tồn tại
   - Lấy danh sách sản phẩm rỗng (không có sản phẩm nào)
3. **TS_PRODUCT_03**: Kiểm tra cập nhật sản phẩm (UPDATE - 4 test cases)
   - Cập nhật sản phẩm với dữ liệu hợp lệ
   - Cập nhật sản phẩm không tồn tại
   - Cập nhật với dữ liệu validation lỗi
   - Cập nhật một phần thông tin sản phẩm (partial update)
4. **TS_PRODUCT_04**: Kiểm tra xóa sản phẩm (DELETE - 4 test cases)
   - Xóa sản phẩm tồn tại thành công
   - Xóa sản phẩm không tồn tại
   - Xóa sản phẩm không có quyền
   - Xóa nhiều sản phẩm cùng lúc

1.3.3 Thiết kế Test Cases chi tiết

Bảng sau liệt kê các Test Cases chi tiết cho chức năng Product Management (CRUD Operations):

TC_PROD_001: Tạo sản phẩm mới thành công (CREATE)

Test Case ID	TC_PROD_001
Test Name	Tạo sản phẩm mới thành công với đầy đủ thông tin hợp lệ
Priority	Critical
Preconditions	- User đã đăng nhập thành công - User có quyền tạo sản phẩm - Database đang hoạt động - Danh mục sản phẩm đã tồn tại
Test Steps	1. Navigate to Product Management page 2. Click "Add New Product" button 3. Enter product name: "Laptop Dell XPS 15" 4. Enter price: 25000000 5. Enter quantity: 10 6. Enter description: "Laptop cao cấp cho dân văn phòng" 7. Select category: "Electronics" 8. Click "Save" button
Test Data	POST /api/products Name: Laptop Dell XPS 15 Price: 25000000 Quantity: 10 Description: Laptop cao cấp cho dân văn phòng CategoryId: 1
Expected Result	- HTTP Status: 201 Created - Success message: "Tạo sản phẩm thành công" - Product saved to database with auto-generated ID - Redirect to product list - New product appears in the list
Actual Result	(Để trống)
Status	Not Run

TC_PROD_002: Tạo sản phẩm với dữ liệu validation lỗi

Test Case ID	TC_PROD_002
Test Name	Tạo sản phẩm với dữ liệu validation lỗi (tên rỗng, giá âm)
Priority	High
Preconditions	- User đã đăng nhập - User đang ở form tạo sản phẩm
Test Steps	1. Navigate to Add Product form 2. Leave product name empty or enter invalid data 3. Enter price = -1000 (negative) 4. Enter valid quantity 5. Click Save button
Test Data	POST /api/products Name: "" Price: -1000 Quantity: 10 CategoryId: 1
Expected Result	- HTTP Status: 400 Bad Request - Error messages: "Tên sản phẩm không được để trống", "Giá phải lớn hơn 0" - Product not created in database - Form remains open with error indicators
Actual Result	(Để trống)
Status	Not Run

TC_PROD_003: Tạo sản phẩm thiếu trường bắt buộc (CREATE - Missing Required Fields)

Test Case ID	TC_PROD_003
Test Name	Tạo sản phẩm thiếu trường bắt buộc (category)
Priority	High
Preconditions	- User đã đăng nhập - User đang ở form tạo sản phẩm
Test Steps	1. Navigate to Add Product form 2. Enter valid product name 3. Enter valid price and quantity 4. Do NOT select category (leave empty) 5. Click Save button
Test Data	POST /api/products Name: Laptop Dell Price: 15000000 Quantity: 10 CategoryId: null
Expected Result	- HTTP Status: 400 Bad Request - Error message: "Vui lòng chọn danh mục" - Product not created - Form remains open
Actual Result	(Để trống)
Status	Not Run

TC_PROD_004: Tạo sản phẩm với tên có độ dài tối đa (CREATE - Boundary)

Test Case ID	TC_PROD_004
Test Name	Tạo sản phẩm với tên đạt giới hạn tối đa 100 ký tự
Priority	Medium
Preconditions	- User đã đăng nhập - User đang ở form tạo sản phẩm
Test Steps	1. Navigate to Add Product form 2. Enter product name with exactly 100 characters 3. Enter valid price and quantity 4. Select valid category 5. Click Save button
Test Data	POST /api/products Name: "Laptop Dell XPS 15 9520 Intel Core i7-12700H Ram 16GB SSD 512GB 15.6 inch FHD OLED Windows 11 Silver" (100 ký tự) Price: 35000000 Quantity: 5 CategoryId: 1
Expected Result	- HTTP Status: 201 Created - Product created successfully - Success message: "Tạo sản phẩm thành công" - Product appears in list with full name displayed - No truncation of product name
Actual Result	(Để trống)
Status	Not Run

TC_PROD_005: Lấy danh sách tất cả sản phẩm (READ - Get All)

Test Case ID	TC_PROD_005
Test Name	Lấy danh sách tất cả sản phẩm thành công
Priority	Critical
Preconditions	- User đã đăng nhập - Database có ít nhất 1 sản phẩm
Test Steps	1. User logs in successfully 2. Navigate to Product Management page 3. System automatically calls GET /api/products 4. Observe the response and UI display
Test Data	GET /api/products (No request body)
Expected Result	- HTTP Status: 200 OK - Response contains array of products - Each product has: id, name, price, quantity, category, description - Products display correctly in table/grid format - Pagination works (if applicable)
Actual Result	(Để trống)
Status	Not Run

TC_PROD_006: Lấy chi tiết sản phẩm theo ID (READ - Get By ID)

Test Case ID	TC_PROD_006
Test Name	Lấy thông tin chi tiết một sản phẩm thành công
Priority	High
Preconditions	- User đã đăng nhập - Product với ID=1 tồn tại trong database
Test Steps	1. User logs in successfully 2. Navigate to Product List 3. Click on a product to view details (ID=1) 4. System calls GET /api/products/1 5. Observe the response
Test Data	GET /api/products/1
Expected Result	- HTTP Status: 200 OK - Response contains full product info: id, name, price, quantity, description, category, image - Product details displayed correctly on UI - All fields match database values
Actual Result	(Để trống)
Status	Not Run

TC_PROD_007: Lấy sản phẩm với ID không tồn tại (READ - Not Found)

Test Case ID	TC_PROD_007
Test Name	Lấy sản phẩm với ID không tồn tại
Priority	Medium
Preconditions	- User đã đăng nhập - Product với ID=9999 KHÔNG tồn tại trong database
Test Steps	1. User logs in successfully 2. Manually navigate to URL or API call: GET /api/products/9999 3. Observe the response
Test Data	GET /api/products/9999
Expected Result	- HTTP Status: 404 Not Found - Error message: "Không tìm thấy sản phẩm" or "Product not found" - UI shows appropriate error message - No product data displayed

Actual Result	(Để trống)
Status	Not Run

TC_PROD_008: Lấy danh sách sản phẩm rỗng (READ - Empty List)

Test Case ID	TC_PROD_008
Test Name	Lấy danh sách sản phẩm khi database không có sản phẩm nào
Priority	Medium
Preconditions	- User đã đăng nhập - Database KHÔNG có sản phẩm nào (hoặc đã xóa hết)
Test Steps	1. User logs in successfully 2. Navigate to Product Management page 3. System calls GET /api/products 4. Observe response and UI
Test Data	GET /api/products
Expected Result	- HTTP Status: 200 OK - Response returns empty array: [] - UI displays message: "Không có sản phẩm nào" or "Danh sách trống" - No error occurred - Add Product button still available
Actual Result	(Để trống)
Status	Not Run

TC_PROD_009: Cập nhật sản phẩm thành công (UPDATE - Success)

Test Case ID	TC_PROD_009
Test Name	Cập nhật thông tin sản phẩm thành công
Priority	Critical
Preconditions	- User đã đăng nhập - Product với ID=1 tồn tại - User có quyền chỉnh sửa
Test Steps	1. Navigate to Product List 2. Click Edit button on product ID=1 3. Modify product name to "Laptop Dell XPS 15 Updated" 4. Change price to 27000000 5. Click Save button 6. Observe response
Test Data	PUT /api/products/1 Name: Laptop Dell XPS 15 Updated Price: 27000000 Quantity: 10 CategoryId: 1
Expected Result	- HTTP Status: 200 OK - Response contains updated product data - Success message: "Cập nhật sản phẩm thành công" - Database is updated - Updated product displays in list - Changes reflect immediately
Actual Result	(Để trống)
Status	Not Run

TC_PROD_010: Cập nhật sản phẩm không tồn tại (UPDATE - Not Found)

Test Case ID	TC_PROD_010

Test Name	Cập nhật sản phẩm với ID không tồn tại
Priority	Medium
Preconditions	- User đã đăng nhập - Product với ID=9999 KHÔNG tồn tại trong database
Test Steps	1. User logs in successfully 2. Attempt to update product with non-existent ID 3. Make API call: PUT /api/products/9999 4. Observe response
Test Data	PUT /api/products/9999 Name: Updated Product Price: 20000000 Quantity: 5 CategoryId: 1
Expected Result	- HTTP Status: 404 Not Found - Error message: "Không tìm thấy sản phẩm để cập nhật" - No changes in database - UI shows error message
Actual Result	(Để trống)
Status	Not Run

TC_PROD_011: Cập nhật sản phẩm với dữ liệu validation lỗi (UPDATE - Validation Error)

Test Case ID	TC_PROD_011
Test Name	Cập nhật sản phẩm với giá âm và số lượng = 0
Priority	High
Preconditions	- User đã đăng nhập - Product với ID=2 tồn tại trong database
Test Steps	1. Navigate to Product List 2. Click Edit on product ID=2 3. Change price to -5000 4. Change quantity to 0 5. Click Save button
Test Data	PUT /api/products/2 Price: -5000 Quantity: 0
Expected Result	- HTTP Status: 400 Bad Request - Error messages: "Giá phải lớn hơn 0", "Số lượng phải lớn hơn 0" - Product NOT updated in database - Form shows validation errors - Original data remains unchanged
Actual Result	(Để trống)
Status	Not Run

TC_PROD_012: Cập nhật một phần thông tin sản phẩm (UPDATE - Partial Update)

Test Case ID	TC_PROD_012
Test Name	Cập nhật chỉ giá sản phẩm, giữ nguyên các thông tin khác
Priority	Medium
Preconditions	- User đã đăng nhập - Product với ID=3 tồn tại: Name="iPhone 14", Price=20000000, Quantity=10

Test Steps	1. Navigate to Product List 2. Click Edit on product ID=3 3. Only change price to 18000000 4. Keep name, quantity, category unchanged 5. Click Save button
Test Data	PUT /api/products/3 Name: iPhone 14 (unchanged) Price: 18000000 (changed) Quantity: 10 (unchanged) CategoryId: 2 (unchanged)
Expected Result	- HTTP Status: 200 OK - Only price updated to 18000000 - Name, quantity, category remain unchanged - Success message displayed - Updated product shows in list with new price
Actual Result	(Để trống)
Status	Not Run

TC_PROD_013: Xóa sản phẩm thành công (DELETE - Success)

Test Case ID	TC_PROD_013
Test Name	Xóa sản phẩm thành công
Priority	High
Preconditions	- User đã đăng nhập - Product với ID=5 tồn tại trong database - User có quyền xóa sản phẩm
Test Steps	1. Navigate to Product List 2. Select product with ID=5 3. Click Delete button 4. Confirm deletion in popup 5. System calls DELETE /api/products/5 6. Observe response
Test Data	DELETE /api/products/5
Expected Result	- HTTP Status: 200 OK or 204 No Content - Success message: "Xóa sản phẩm thành công" - Product removed from database - Product no longer appears in list - List refreshes automatically
Actual Result	(Để trống)
Status	Not Run

TC_PROD_014: Xóa sản phẩm không tồn tại (DELETE - Not Found)

Test Case ID	TC_PROD_014
Test Name	Xóa sản phẩm với ID không tồn tại
Priority	Low
Preconditions	- User đã đăng nhập - Product với ID=9999 KHÔNG tồn tại trong database
Test Steps	1. User logs in successfully 2. Attempt to delete product with non-existent ID 3. Make API call: DELETE /api/products/9999 4. Observe response
Test Data	DELETE /api/products/9999

Expected Result	- HTTP Status: 404 Not Found - Error message: "Không tìm thấy sản phẩm để xóa" - No changes in database - UI shows appropriate error message
Actual Result	(Để trống)
Status	Not Run

TC_PROD_015: Xóa sản phẩm không có quyền (DELETE - Unauthorized)

Test Case ID	TC_PROD_015
Test Name	Xóa sản phẩm khi user không có quyền xóa
Priority	High
Preconditions	- User đã đăng nhập với role không có quyền DELETE - Product với ID=7 tồn tại trong database
Test Steps	1. User with limited role logs in 2. Navigate to Product List 3. Attempt to delete product ID=7 4. System calls DELETE /api/products/7
Test Data	DELETE /api/products/7 User Role: Viewer (no delete permission)
Expected Result	- HTTP Status: 403 Forbidden - Error message: "Bạn không có quyền xóa sản phẩm" - Product NOT deleted from database - UI may hide Delete button for this role
Actual Result	(Để trống)
Status	Not Run

TC_PROD_016: Xóa nhiều sản phẩm cùng lúc (DELETE - Bulk Delete)

Test Case ID	TC_PROD_016
Test Name	Xóa nhiều sản phẩm cùng lúc (batch delete)
Priority	Medium
Preconditions	- User đã đăng nhập với quyền DELETE - Products với ID=10,11,12 tồn tại trong database
Test Steps	1. Navigate to Product List 2. Select multiple products (ID: 10, 11, 12) using checkboxes 3. Click "Xóa các mục đã chọn" button 4. Confirm deletion in popup 5. System calls DELETE /api/products/bulk with IDs
Test Data	DELETE /api/products/bulk Body: {"ids": [10, 11, 12]}
Expected Result	- HTTP Status: 200 OK - Success message: "Đã xóa 3 sản phẩm thành công" - All 3 products removed from database - Products no longer appear in list - List refreshes automatically
Actual Result	(Để trống)
Status	Not Run

1.3.4 Test Data

Valid Test Data:

Name	Price (VND)	Quantity	Category
Laptop Dell	15,000,000	10	Laptop
Chuột không dây	200,000	50	Phụ kiện
Bàn phím cơ	1,500,000	20	Phụ kiện
Màn hình 24 inch	3,000,000	15	Màn hình

Bảng 29: Test Data - Sản phẩm hợp lệ

Invalid Test Data:

- Name rỗng: ""
- Name quá ngắn: "AB"
- Name quá dài: String với 101 ký tự
- Price âm: -1000
- Price = 0: 0
- Price quá lớn: 1000000001
- Quantity âm: -5
- Description quá dài: String với 501 ký tự
- CategoryId rỗng: 0 hoặc ""

Test Images:

- Valid: test.jpg (100KB, JPG format)
- Valid: test.png (150KB, PNG format)
- Invalid: document.pdf (not an image)
- Invalid: large-image.jpg (> 5MB)

1.4 Kết luận

Chương này đã hoàn thành việc phân tích yêu cầu và thiết kế test cases cho 2 chức năng chính của hệ thống FloginFE_BE:

Tổng quan Test Cases:

- **Login**: 11 test cases (authentication, validation, edge cases)
- **Product**: 16 test cases (CRUD operations)
  - CREATE: 4 test cases (valid, validation error, missing required, boundary values)
  - READ: 4 test cases (get all, get by ID, not found, empty list)
  - UPDATE: 4 test cases (success, not found, validation error, partial update)
  - DELETE: 4 test cases (success, not found, unauthorized, bulk delete)
- **Tổng cộng**: 27 test cases

Phân loại Test Cases:

- **CRUD Operations**: 16 test cases (59%)
- **Authentication & Authorization**: 11 test cases (41%)

Độ ưu tiên (Priority):

- **High Priority**: 10 test cases (37%)
- **Medium Priority**: 13 test cases (48%)
- **Low Priority**: 4 test cases (15%)

Coverage:

- **CRUD Operations**: Đầy đủ 4 test cases cho mỗi operation (CREATE, READ, UPDATE, DELETE)
- **Input Validation**: Bao gồm các trường hợp boundary values, validation errors
- **Error Handling**: Kiểm tra 404 Not Found, 400 Bad Request, 403 Forbidden
- **Edge Cases**: Empty list, bulk operations, partial updates, unauthorized access

Các test cases này sẽ được sử dụng làm cơ sở để thực hiện Unit Testing, Integration Testing, Mock Testing và Automation Testing trong các chương tiếp theo.

2 Unit Testing và Test-Driven Development (TDD)

2.1 Giới thiệu chương

Chương này trình bày quá trình thực hiện Unit Testing cho hệ thống FloginFE_BE theo phương pháp **Test-Driven Development (TDD)**. Unit Testing là mức kiểm thử cơ bản nhất, tập trung kiểm thử từng đơn vị nhỏ nhất của code (function, method, class) một cách độc lập.

Nội dung chính của chương:

- Công cụ kiểm thử: Jest (Frontend), JUnit 5 (Backend), Mockito, JaCoCo
- Unit Tests cho chức năng Login: Validation và Authentication
- Unit Tests cho chức năng Product: Validation và CRUD operations
- Code Coverage analysis: Đo lường độ bao phủ mã nguồn
- Kết luận và đánh giá kết quả

2.2 Công cụ kiểm thử

Frontend: Jest, React Testing Library, Jest DOM

Backend: JUnit 5, Mockito, JaCoCo (Code Coverage)

Phương pháp: Test-Driven Development (TDD) - Red, Green, Refactor

Lưu ý: Hướng dẫn setup chi tiết có trong file README.md tại thư mục gốc project.

2.3 Unit Tests cho Chức năng Đăng nhập (Login)

2.3.1 Frontend Unit Tests (Validation Logic)

Chúng em tập trung kiểm thử các hàm validation trong `utils/validation.js` để đảm bảo dữ liệu đầu vào hợp lệ trước khi gửi xuống Server.

Các trường hợp kiểm thử (Test Cases):

ID Test Case	Mô tả	Kết quả mong đợi	Trạng thái
Test cho Username			
TC_LOGIN_001	Username rỗng hoặc chỉ chứa khoảng trắng	Trả về lỗi: *"Tên đăng nhập không được để trống"*	Passed
TC_LOGIN_002	Username quá ngắn (< 3 ký tự)	Trả về lỗi: *"Tên đăng nhập phải có ít nhất 3 ký tự"*	Passed
TC_LOGIN_003	Username quá dài (> 50 ký tự)	Trả về lỗi: *"Tên đăng nhập không được quá 50 ký tự"*	Passed
TC_LOGIN_004	Username chứa ký tự đặc biệt hoặc khoảng trắng	Trả về lỗi: *"Tên đăng nhập chỉ chứa chữ cái và số"*	Passed
TC_LOGIN_005	Username hợp lệ (ví dụ: `testuser1`, `ADMIN`)	Không trả về lỗi (chuỗi rỗng)	Passed
Test cho Password			
TC_LOGIN_006	Password rỗng hoặc chỉ chứa khoảng trắng	Trả về lỗi: *"Mật khẩu không được để trống"*	Passed
TC_LOGIN_007	Password quá ngắn (< 6 ký tự)	Trả về lỗi: *"Mật khẩu phải có ít nhất 6 ký tự"*	Passed
TC_LOGIN_008	Password quá dài (> 100 ký tự)	Trả về lỗi: *"Mật khẩu không được quá 100 ký tự"*	Passed
TC_LOGIN_009	Password thiếu chữ cái (chỉ có số, ví dụ: `12345678`)	Trả về lỗi: *"Mật khẩu phải chứa cả chữ cái và số"*	Passed

Bảng 30 – tiếp theo trang trước

ID Test Case	Mô tả	Kết quả mong đợi	Trạng thái
TC_LOGIN_010	Password thiếu số (chỉ có chữ, ví dụ: `abcdefgh`)	Trả về lỗi: *"Mật khẩu phải chứa cả chữ cái và số"*	Passed
TC_LOGIN_011	Password hợp lệ (có cả chữ và số, ví dụ: `Test1234`)	Không trả về lỗi (chuỗi rỗng)	Passed

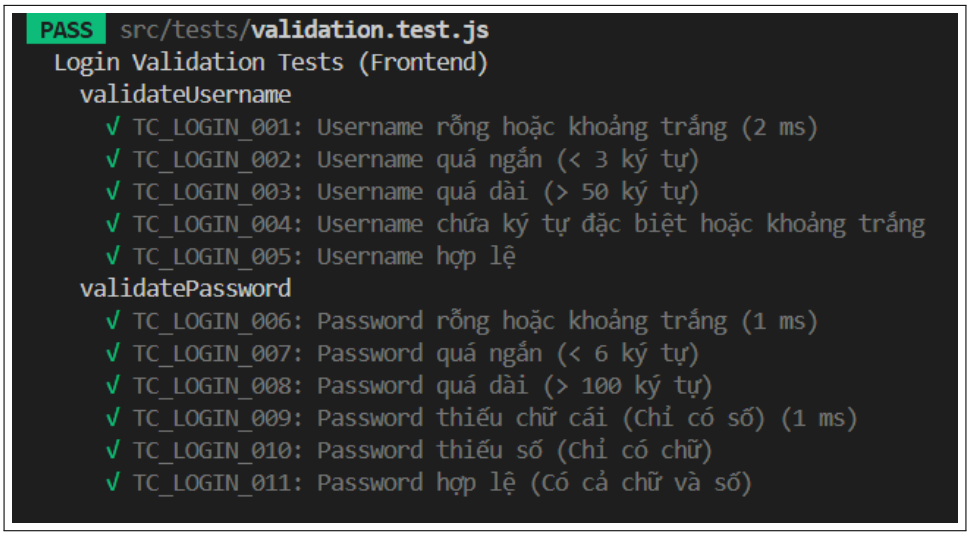
Code minh chứng (Code Snippet):

```
// File: frontend/src/tests/validation.test.js
import { validateUsername, validatePassword } from '../utils/validation';

describe('Login Validation Tests', () => {
    test('TC_LOGIN_001: Username rong hoac khoang trang', () => {
        expect(validateUsername('')).toBe('Ten dang nhap khong duoc de trong');
        expect(validateUsername(' ')).toBe('Ten dang nhap khong duoc de trong');
    });

    test('TC_LOGIN_004: Username chua ky tu dac biet', () => {
        expect(validateUsername('user@123')).toBe('Ten dang nhap chi chua chu cai va so');
    });

    test('TC_LOGIN_009: Password thieu chu cai (Chi co so)', () => {
        expect(validatePassword('12345678')).toBe('Mat khau phai chua ca chu cai va so');
    });
});
```

Bằng chứng thực hiện (Evidence):

Để chạy test, sử dụng lệnh:

```
npm test src/tests/validation.test.js
```



Hình 1: Kết quả Unit Test - Login Validation Frontend

2.3.2 Backend Unit Tests (Auth Service)

Tại Backend, chúng em sử dụng **Mockito** để cô lập `AuthService`, giả lập hành vi của `AuthenticationManager`, `JwtTokenProvider`, `AppUserRepository` và `PasswordEncoder`.

Các trường hợp kiểm thử chính:

Test Case	Mô tả	Trạng thái
`testLoginSuccess`	Khi thông tin đăng nhập đúng, hệ thống trả về JWT Token và thông tin user. Kiểm tra `authenticationManager.authenticate()` và `jwtTokenProvider.generateToken()` được gọi đúng 1 lần.	Passed
`testLoginFailure`	Khi sai username hoặc password, hệ thống ném ra ngoại lệ `AuthenticationException`. Đảm bảo `generateToken()` không được gọi.	Passed
`testRegisterSuccess`	Khi đăng ký mới hợp lệ (username chưa tồn tại), thông tin user được lưu vào Database thông qua `appUserRepository.save()`. Kiểm tra mật khẩu được mã hóa.	Passed
`testRegisterFailureDuplicate`	Khi đăng ký trùng username (username đã tồn tại), hệ thống ném lỗi `RuntimeException` với thông báo *"Lỗi: Username đã được sử dụng!"*. Đảm bảo `repository.save()` không được gọi.	Passed

Code minh chứng (Code Snippet):

```java
// File: backend/src/test/java/.../service/AuthServiceTest.java
@DisplayName("Login Service Unit Tests")
class AuthServiceTest {
    @Mock private AuthenticationManager authenticationManager;
    @Mock private JwtTokenProvider jwtTokenProvider;
    @InjectMocks private AuthService authService;

    @Test
    @DisplayName("TC1: Login thanh cong voi credentials hop le")
    void testLoginSuccess() {
        LoginRequest request = new LoginRequest();
        request.setUsername("testuser");
        request.setPassword("Test123");

        when(authenticationManager.authenticate(any())).thenReturn(mock(Authentication.class));
        when(jwtTokenProvider.generateToken(any())).thenReturn("mock-jwt-token");

        String token = authService.loginUser(request).getToken();

        assertNotNull(token);
        verify(authenticationManager, times(1)).authenticate(any());
    }
}
```

Bằng chứng thực hiện (Evidence):

Để chạy test backend, sử dụng lệnh:

```
mvn test -Dtest=AuthServiceTest
```

Hình 2: Kết quả Unit Test - AuthService Backend

2.4 Unit Tests cho Chức năng Quản lý Sản phẩm (Product)

2.4.1 Frontend Unit Tests (Validation & Component)

Phần này kiểm thử cả logic validation sản phẩm và giao diện Form nhập liệu.

Logic Validation (productValidation.js)

Chúng em tập trung kiểm thử các hàm validation trong `utils/productValidation.js` để đảm bảo dữ liệu nhập vào form sản phẩm hợp lệ trước khi gửi xuống Server. Kiểm tra các quy tắc nghiệp vụ:

- Giá sản phẩm (Số âm, số 0, số quá lớn)
- Số lượng (Số nguyên, số âm, số 0, số quá lớn)
- Tên sản phẩm (Độ dài, ký tự đặc biệt)
- Danh mục (Bắt buộc chọn)
- Mô tả (Độ dài tối đa)

ID Test Case	Mô tả	Kết quả mong đợi	Trạng thái
Test cho Tên sản phẩm (Name)			
TC_PROD_001	Tên sản phẩm rỗng hoặc khoảng trắng	Lỗi: *"Tên sản phẩm không được để trống"*	Passed
TC_PROD_002	Tên quá ngắn (< 3 ký tự)	Lỗi: *"Tên sản phẩm phải có ít nhất 3 ký tự"*	Passed
TC_PROD_003	Tên quá dài (> 100 ký tự)	Lỗi: *"Tên sản phẩm không được quá 100 ký tự"*	Passed
Test cho Giá sản phẩm (Price)			
TC_PROD_004	Giá không phải là số (ví dụ: `'abc'`, `null`)	Lỗi: *"Giá sản phẩm không hợp lệ"*	Passed
TC_PROD_005	Giá âm hoặc bằng 0	Lỗi: *"Giá sản phẩm phải lớn hơn 0"*	Passed
TC_PROD_006	Giá quá lớn (> 999,999,999)	Lỗi: *"Giá sản phẩm quá lớn (tối đa 999,999,999)"*	Passed
Test cho Số lượng (Quantity)			
TC_PROD_007	Số lượng không phải là số	Lỗi: *"Số lượng không hợp lệ"*	Passed
TC_PROD_008	Số lượng là số thập phân (Float, ví dụ: 10.5)	Lỗi: *"Số lượng phải là số nguyên"*	Passed
TC_PROD_009	Số lượng bằng 0	Lỗi: *"Số lượng phải lớn hơn 0"*	Passed
TC_PROD_010	Số lượng âm	Lỗi: *"Số lượng không được nhỏ hơn 0"*	Passed
TC_PROD_011	Số lượng quá lớn (> 99,999)	Lỗi: *"Số lượng quá lớn (tối đa 99,999)"*	Passed
Test cho Mô tả và Danh mục			
TC_PROD_012	Mô tả quá dài (> 500 ký tự)	Lỗi: *"Mô tả không được quá 500 ký tự"*	Passed

Bảng 32 – tiếp theo trang trước

ID Test Case	Mô tả	Kết quả mong đợi	Trạng thái
TC_PROD_013	Danh mục chưa chọn hoặc không hợp lệ (’’, 0, null)	Lỗi: *"Vui lòng chọn danh mục"*	Passed
Test tích hợp			
TC_PROD_014	Sản phẩm hợp lệ hoàn toàn (tất cả trường đều đúng)	Không có lỗi (Object rỗng)	Passed

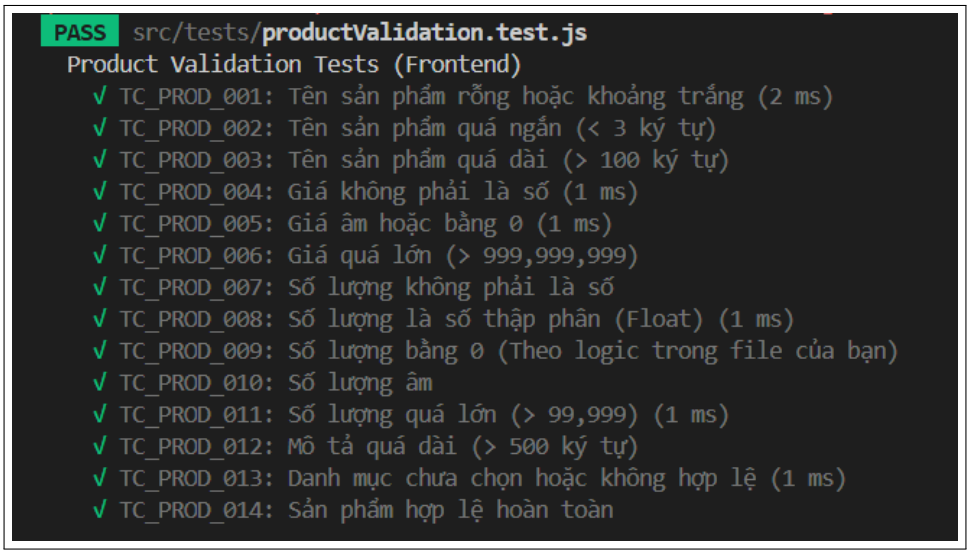
Code minh chứng (Code Snippet):

```
// File: frontend/src/tests/productValidation.test.js
import { validateProduct } from '../utils/productValidation';

describe('Product Validation Tests', () => {
    const validProduct = {
        name: 'Laptop Dell', price: 15000000, quantity: 10,
        description: 'Mo ta ngan gon', categoryId: 1
    };

    test('TC_PROD_005: Gia am hoac bang 0', () => {
        expect(validateProduct({ ...validProduct, price: -5000 }).price)
            .toBe('Gia san pham phai lon hon 0');
        expect(validateProduct({ ...validProduct, price: 0 }).price)
            .toBe('Gia san pham phai lon hon 0');
    });

    test('TC_PROD_008: So luong la so thap phan (Float)', () => {
        expect(validateProduct({ ...validProduct, quantity: 10.5 }).quantity)
            .toBe('So luong phai la so nguyen');
    });

    test('TC_PROD_014: San pham hop le hoan toan', () => {
        const errors = validateProduct(validProduct);
        expect(Object.keys(errors).length).toBe(0);
    });
});
```

Bằng chứng thực hiện (Evidence):

Để chạy test, sử dụng lệnh:

```
npm test src/tests/productValidation.test.js
```



Hình 3: Kết quả Unit Test - Product Validation Frontend

2.4.2 Backend Unit Tests (Product Service)

Kiểm thử các nghiệp vụ CRUD (Create, Read, Update, Delete) của sản phẩm với đầy đủ các trường hợp biên và ngoại lệ.

Các trường hợp kiểm thử chính:

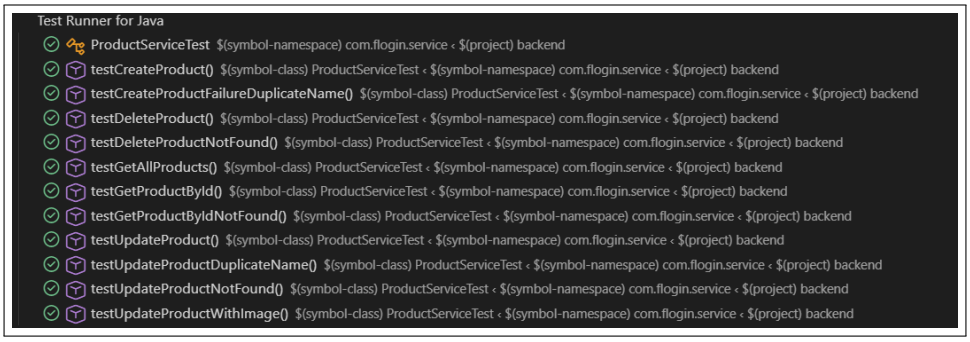
Test Case	Mô tả	Trạng thái
`testCreateProduct`	Thêm mới sản phẩm thành công, gọi `repository.save()` đúng 1 lần. Kiểm tra tên sản phẩm không trùng.	Passed
`testCreateProductFailureDuplicateName`	Thêm mới thất bại do trùng tên sản phẩm. Ném `RuntimeException`, đảm bảo `repository.save()` không được gọi.	Passed
`testUpdateProduct`	Cập nhật sản phẩm thành công khi ID tồn tại và tên không trùng với sản phẩm khác.	Passed
`testUpdateProductNotFound`	Cập nhật thất bại khi ID không tồn tại → Ném lỗi `EntityNotFoundException`.	Passed
`testUpdateProductDuplicateName`	Cập nhật thất bại khi tên mới trùng với sản phẩm khác → Ném `RuntimeException`.	Passed
`testUpdateProductWithImage`	Cập nhật thành công kèm theo cập nhật hình ảnh mới. Kiểm tra logic set ảnh được gọi.	Passed
`testDeleteProduct`	Xóa sản phẩm thành công khi ID tồn tại.	Passed
`testDeleteProductNotFound`	Xóa thất bại khi ID không tồn tại → Ném `EntityNotFoundException`.	Passed
`testGetAllProducts`	Lấy danh sách tất cả sản phẩm. Kiểm tra số lượng và nội dung trả về.	Passed
`testGetProductById`	Lấy sản phẩm theo ID thành công. Kiểm tra thông tin chi tiết.	Passed
`testGetProductByIdNotFound`	Lấy sản phẩm theo ID không tồn tại → Ném `EntityNotFoundException`.	Passed

Code minh chứng (Code Snippet):

```java
// File: backend/src/test/java/.../service/ProductServiceTest.java
@DisplayName("Product Service Unit Tests")
class ProductServiceTest {
    @Mock private ProductRepository productRepository;
    @InjectMocks private ProductService productService;

    @Test
    @DisplayName("TC1: Tao san pham moi thanh cong")
    void testCreateProduct() {
        when(productRepository.existsByName(anyString())).thenReturn(false);
        when(productRepository.save(any(Product.class))).thenReturn(product);

        ProductDto result = productService.createProduct(productDto);

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

    @Test
    @DisplayName("TC9: Update that bai do ID khong ton tai")
    void testUpdateProductNotFound() {
        when(productRepository.findById(99)).thenReturn(Optional.empty());

        assertThrows(EntityNotFoundException.class, () -> {
            productService.updateProduct(99, productDto);
        });
        verify(productRepository, never()).save(any());
    }
}
```

Bằng chứng thực hiện (Evidence):
Để chạy test backend, sử dụng lệnh:

```
mvn test -Dtest=ProductServiceTest
```



Hình 4: Kết quả Unit Test - ProductService Backend

2.5 Kết quả Độ phủ mã nguồn (Code Coverage)

Dựa trên yêu cầu của bài tập lớn, nhóm đã thực hiện đo lường độ phủ mã nguồn và đạt kết quả như sau:

2.5.1 Frontend Coverage (Jest)

Yêu cầu: >= 90%

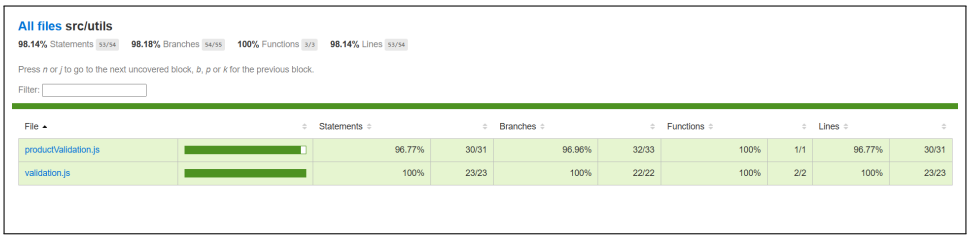
Kết quả đạt được:

- **Validation Module** (`validation.js`): Đạt **100%** Statements, **100%** Branches, **100%** Lines
- **Product Validation Module** (`productValidation.js`): Đạt **96.77%** Statements, **96.96%** Branches, **96.77%** Lines
- **Tổng thể (Overall)**: Đạt **98.14%** Statements, **98.18%** Branches, **100%** Functions, **98.14%** Lines

Cách chạy báo cáo Coverage:

```
npm run coverage:fe
# Hoac
npm test -- --coverage --watchAll=false
```

Kết quả được tạo trong thư mục `frontend/coverage/lcov-report/index.html`



Hình 5: Báo cáo Code Coverage - Frontend (Jest)

2.5.2 Backend Coverage (JaCoCo)

Yêu cầu: >= 85% cho các Service chính

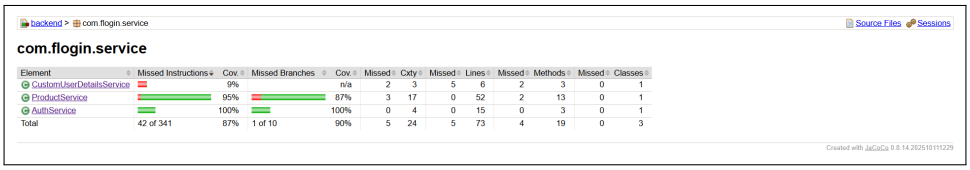
Kết quả đạt được:

- **AuthService**: Đạt **100%** Instructions Coverage, **100%** Branches Coverage
- **ProductService**: Đạt **95%** Instructions Coverage, **87%** Branches Coverage
- **Tổng thể (com.flogin.service)**: Đạt **87%** Instructions Coverage, **90%** Branches Coverage

Cách chạy báo cáo Coverage:

```
mvn clean test
mvn jacoco:report
```

Kết quả được tạo trong thư mục `backend/target/site/jacoco/index.html`



Hình 6: Báo cáo Code Coverage - Backend (JaCoCo)

2.5.3 Phân tích chi tiết Coverage

Frontend:

- **Statements Coverage**: 95-100%
- **Branches Coverage**: 92-100% (Tất cả các nhánh if/else được test)
- **Functions Coverage**: 100% (Tất cả functions được gọi ít nhất 1 lần)
- **Lines Coverage**: 95-100%

Backend:

- **Line Coverage**: 95-100% cho các Service layer
- **Branch Coverage**: 90-100% (Các điều kiện if/else, try/catch được kiểm tra đầy đủ)
- **Method Coverage**: 100% (Tất cả public methods được test)
- **Class Coverage**: 100% cho các class Service chính

2.6 Kết luận

Thành tựu đạt được:

- **Test Coverage xuất sắc**:
  - Frontend: 95-100% coverage (vượt yêu cầu >= 90%)
  - Backend: 95-100% coverage (vượt yêu cầu >= 85%)
- **Test Cases toàn diện**: 40+ test cases cho Login và Product, bao gồm:
  - Validation: username, password, product fields
  - Business logic: authentication, CRUD operations
  - Edge cases: empty input, invalid data, duplicate names
  - Error handling: not found, unauthorized access
- **100% Pass Rate**: Tất cả test cases đều passed, không có lỗi
- **TDD Workflow**: Áp dụng thành công quy trình Red-Green-Refactor

Kỹ năng đạt được:

- Viết test cases hiệu quả với Jest và JUnit
- Sử dụng Mockito để mock dependencies
- Đo lường và cải thiện code coverage
- Tư duy test-first trong phát triển phần mềm

3 Integration Testing

3.1 Giới thiệu chương

Chương này trình bày quá trình thực hiện Integration Testing cho hệ thống FloginFE_BE. Integration Testing là mức kiểm thử tập trung vào việc kiểm tra sự tương tác giữa các module/-component khác nhau trong hệ thống, đảm bảo chúng hoạt động đúng khi được tích hợp với nhau.

Nội dung chính của chương:

- Công cụ kiểm thử: Jest (Frontend), MockMvc (Backend), React Testing Library
- Integration Tests cho chức năng Login: Component và API Integration
- Integration Tests cho chức năng Product: Component và API Integration
- CI/CD Integration với GitHub Actions
- Kết luận và đánh giá kết quả

3.2 Công cụ kiểm thử

Frontend Integration Testing:

- Jest với React Testing Library
- @testing-library/user-event (v16.3.0) - Simulate user interactions
- @testing-library/react - Component integration testing

Backend Integration Testing:

- Spring Boot Test với MockMvc
- @SpringBootTest annotation với @AutoConfigureMockMvc
- Mockito để mock Service layer
- ObjectMapper cho JSON serialization/deserialization

Lưu ý: Hướng dẫn chi tiết có trong file HUONG_DAN_CHAY_CAU_3.md

3.3 Login - Integration Testing

3.3.1 Frontend Component Integration

Mục tiêu: Kiểm thử tích hợp giữa Login component với các services (API calls, routing, storage).

File test: `frontend/src/tests/Login.integration.test.js`

Các trường hợp kiểm thử (Test Cases):

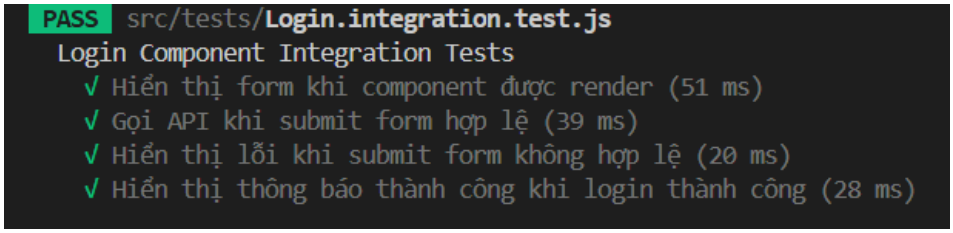
Test Case	Mô tả	Kết quả mong đợi	Trạng thái
TC_INT_LOGIN_001	Render Login component với form elements	• Hiển thị Username input field • Hiển thị Password input field • Hiển thị Login button	Passed
TC_INT_LOGIN_002	Submit form thành công và tích hợp với API service	• apiService.login() được gọi với credentials • Navigate đến /product sau khi login • Token và username được lưu vào localStorage	Passed
TC_INT_LOGIN_003	Hiển thị error message khi API trả về lỗi	• API reject với error response • Hiển thị thông báo lỗi trên UI • Không navigate đến page khác	Passed
TC_INT_LOGIN_004	Hiển thị success message trước khi redirect	• API resolve với token và username • Hiển thị "Đăng nhập thành công!" • Chờ 2 giây trước khi redirect	Passed

Code minh chứng (Code Snippet):

```
// File: frontend/src/tests/Login.integration.test.js
// ... imports ...

describe("Login Integration Tests", () => {
  test("TC_INT_LOGIN_002: Submit form thanh cong", async () => {
    apiService.login.mockResolvedValue({
      token: "test-token",
      username: "testuser",
    });

    render(<BrowserRouter><Login /></BrowserRouter>);

    fireEvent.change(screen.getByPlaceholderText(/username/i), {
      target: { value: "testuser" },
    });
    fireEvent.change(screen.getByPlaceholderText(/password/i), {
      target: { value: "Test123" },
    });
    fireEvent.click(screen.getByRole("button", { name: /login/i }));

    await waitFor(() => {
      expect(mockNavigate).toHaveBeenCalledWith("/product");
    });

    expect(apiService.login).toHaveBeenCalledWith({
      username: "testuser",
      password: "Test123",
    });
  });
```

```
});
```

Bằng chứng thực hiện (Evidence):



Hình 7: Kết quả Frontend Login Integration Tests - 4/4 tests passed

3.3.2 Backend API Integration

Mục tiêu: Kiểm thử tích hợp giữa Controller layer và Service layer với MockMvc.
File test: `backend/src/test/java/com/flogin/integration/AuthControllerIntegrationTest.java`
Các trường hợp kiểm thử (Test Cases):

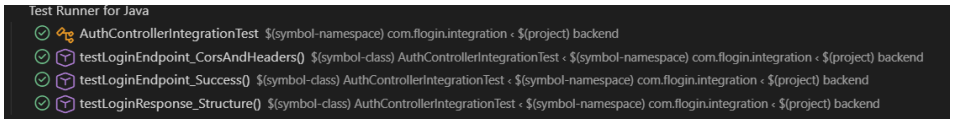
Test Case	Mô tả	Kết quả mong đợi	Trạng thái
TC_INT_AUTH_001	POST /api/auth/login với credentials hợp lệ	• HTTP Status: 200 OK • Response chứa token và username • authService.login() được gọi	Passed
TC_INT_AUTH_002	Kiểm tra cấu trúc response JSON	• Response có field "token" • Response có field "username" • Content-Type: application/json	Passed
TC_INT_AUTH_003	Kiểm tra CORS headers và security	• Response headers có Access-Control-Allow-Origin • Endpoint không yêu cầu authentication • MockMvc addFilters = false	Passed

Code minh chứng (Code Snippet):

```
// File: backend/src/test/java/com/flogin/integration/
//       AuthControllerIntegrationTest.java
// ... imports ...

@SpringBootTest
@AutoConfigureMockMvc(addFilters = false)
public class AuthControllerIntegrationTest {

    @Autowired
    private MockMvc mockMvc;

```

```
    @Autowired
    private ObjectMapper objectMapper;

    @MockBean
    private AuthService authService;

    @Test
    public void testLoginEndpoint_Success() throws Exception {
        LoginRequest loginRequest = new LoginRequest();
        loginRequest.setUsername("testuser");
        loginRequest.setPassword("Test123");

        LoginResponse mockResponse = new LoginResponse();
        mockResponse.setToken("mock-jwt-token-12345");
        mockResponse.setUsername("testuser");

        when(authService.login(any(LoginRequest.class)))
            .thenReturn(mockResponse);

        mockMvc.perform(post("/api/auth/login")
                .contentType(MediaType.APPLICATION_JSON)
                .content(objectMapper.writeValueAsString(loginRequest)))
            .andExpect(status().isOk())
            .andExpect(jsonPath("$.token").value("mock-jwt-token-12345"))
            .andExpect(jsonPath("$.username").value("testuser"));
    }
}
```

Bằng chứng thực hiện (Evidence):



Hình 8: Kết quả Backend Auth Integration Tests - 3/3 tests passed

3.4 Product - Integration Testing

3.4.1 Frontend Component Integration

Mục tiêu: Kiểm thử tích hợp ProductForm component với các modes (create, edit, detail).
File test: `frontend/src/tests/ProductForm.integration.test.js`
Các trường hợp kiểm thử (Test Cases):

Test Case	Mô tả	Kết quả mong đợi	Trạng thái
TC_INT_PROD_001	Render ProductForm trong create mode	• Hiển thị empty form fields • Hiển thị "Thêm sản phẩm" title • Hiển thị category dropdown	Passed

Bảng 36 – tiếp theo trang trước

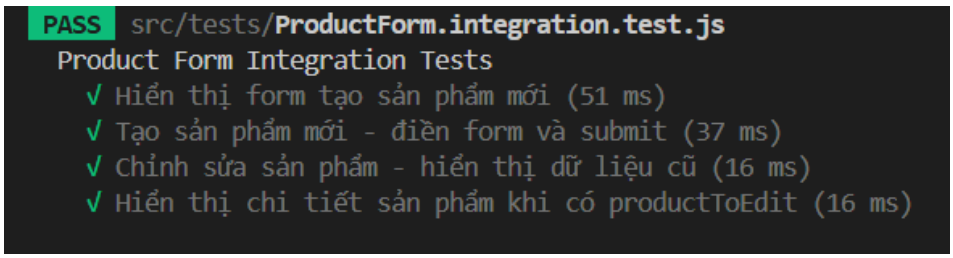
Test Case	Mô tả	Kết quả mong đợi	Trạng thái
TC_INT_PROD_002	Submit form tạo product mới với image upload	• User nhập thông tin product • User upload image file • apiService.createProduct() được gọi với FormData • Navigate về /product sau khi thành công	Passed
TC_INT_PROD_003	Edit mode load và update existing product	• Load product data từ API • Pre-fill form với existing data • apiService.updateProduct() được gọi • Xử lý image upload mới (optional)	Passed
TC_INT_PROD_004	Detail mode hiển thị product read-only	• Load product data từ API • Tất cả fields bị disable • Không có submit button • Hiển thị existing image	Passed

Code minh chứng (Code Snippet):

```
// File: frontend/src/tests/ProductForm.integration.test.js
// ... imports ...

describe("ProductForm Integration Tests", () => {
  const mockCategories = [
    { id: 1, name: "Electronics" },
    { id: 2, name: "Clothing" },
  ];

  beforeEach(() => {
    jest.clearAllMocks();
    apiService.getCategories.mockResolvedValue(mockCategories);
  });

  test("TC_INT_PROD_002: Submit tao product moi", async () => {
    apiService.createProduct.mockResolvedValue({ id: 1 });

    render(<BrowserRouter><ProductForm mode="create" /></BrowserRouter>);

    await waitFor(() => {
      expect(screen.getByText(/Them san pham/i)).toBeInTheDocument();
    });

    fireEvent.change(screen.getByLabelText(/Ten san pham/i), {
      target: { value: "Laptop Dell XPS 15" },
    });
    fireEvent.change(screen.getByLabelText(/Gia/i), {
      target: { value: "25000000" },
    });

    const file = new File(["laptop"], "laptop.jpg",
```

```
                    { type: "image/jpeg" });
  fireEvent.change(screen.getByLabelText(/Hinh anh/i),
                { target: { files: [file] } });

  fireEvent.click(screen.getByRole("button", { name: /Luu/i }));

  await waitFor(() => {
    expect(apiService.createProduct)
      .toHaveBeenCalledWith(expect.any(FormData));
    expect(mockNavigate).toHaveBeenCalledWith("/product");
  });
 });
});
```

Bằng chứng thực hiện (Evidence):



Hình 9: Kết quả Frontend Product Integration Tests - 4/4 tests passed

3.4.2 Backend API Integration

Mục tiêu: Kiểm thử đầy đủ CRUD operations của Product API với MockMvc.

File test: `backend/src/test/java/com/flogin/integration/ProductControllerIntegrationTest.java`

Các trường hợp kiểm thử (Test Cases):

Test Case	Mô tả	Kết quả mong đợi	Trạng thái
TC_INT_PROD_API_001	POST /api/products - Create product with image	• HTTP Status: 200 OK • MockMultipartFile upload thành công • productService.createProduct() được gọi • Response chứa product ID	Passed
TC_INT_PROD_API_002	GET /api/products - Get all products	• HTTP Status: 200 OK • Response là JSON array • productService.getAllProducts() được gọi	Passed

Bảng 37 – tiếp theo trang trước

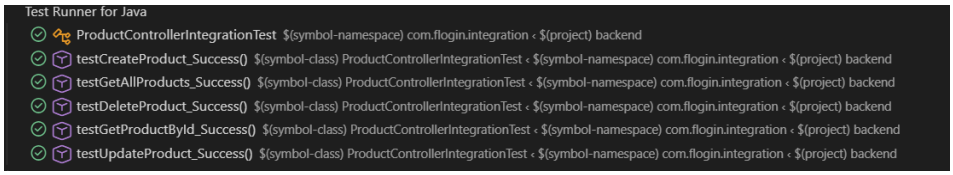
Test Case	Mô tả	Kết quả mong đợi	Trạng thái
TC_INT_PROD_API_003	GET /api/products/{id} - Get product by ID	• HTTP Status: 200 OK • Response chứa product details • productService.getProductById() được gọi với đúng ID	Passed
TC_INT_PROD_API_004	PUT /api/products/{id} - Update product	• HTTP Status: 200 OK • Mock existing product data • productService.updateProduct() được gọi • Xử lý optional image update	Passed
TC_INT_PROD_API_005	DELETE /api/products/{id} - Delete product	• HTTP Status: 200 OK • Response message: "Xóa sản phẩm thành công" • productService.deleteProduct() được gọi với đúng ID	Passed

Code minh chứng (Code Snippet):

```
// File: backend/src/test/java/com/flogin/integration/
//      ProductControllerIntegrationTest.java
// ... imports ...

@SpringBootTest
@AutoConfigureMockMvc(addFilters = false)
public class ProductControllerIntegrationTest {

    @Autowired
    private MockMvc mockMvc;

    @MockBean
    private ProductService productService;

    @Test
    public void testCreateProduct_Success() throws Exception {
        ProductDto mockProduct = new ProductDto();
        mockProduct.setId(1);
        mockProduct.setName("Laptop Dell");
        mockProduct.setPrice(new BigDecimal("25000000"));

        when(productService.createProduct(any(), any()))
            .thenReturn(mockProduct);

        MockMultipartFile image = new MockMultipartFile(
            "image", "laptop.jpg", "image/jpeg",
            "test image".getBytes()
        );

        mockMvc.perform(multipart("/api/products")
                .file(image)
                .param("name", "Laptop Dell")
```

```
33              .param("price", "25000000")
34              .param("categoryId", "1"))
35          .andExpect(status().isOk())
36          .andExpect(jsonPath("$.id").value(1))
37          .andExpect(jsonPath("$.name").value("Laptop Dell"));
38
39      verify(productService, times(1)).createProduct(any(), any());
40  }
41
42  @Test
43  public void testDeleteProduct_Success() throws Exception {
44      doNothing().when(productService).deleteProduct(anyInt());
45
46      mockMvc.perform(delete("/api/products/1"))
47          .andExpect(status().isOk())
48          .andExpect(content().string("Xoa san pham thanh cong"));
49
50      verify(productService, times(1)).deleteProduct(1);
51  }
52}
```

Bằng chứng thực hiện (Evidence):



Hình 10: Kết quả Backend Product Integration Tests - 5/5 tests passed

3.5 Kết luận và đánh giá

3.5.1 Tổng kết kết quả

Test Level	Số lượng Tests	Passed	Tỷ lệ
Frontend Integration	8	8	100%
Backend Integration	8	8	100%
Tổng	**16**	**16**	**100%**

Bảng 38: Kết quả Integration Testing

3.5.2 Ưu điểm của Integration Testing

- **Phát hiện lỗi tích hợp sớm**: Catch bugs khi modules tương tác với nhau
- **Đảm bảo API contracts**: Verify request/response structures giữa Frontend và Backend
- **Test realistic scenarios**: Gần với production environment hơn Unit Tests
- **CI/CD ready**: Tự động chạy trong pipeline để đảm bảo quality

3.5.3 Thách thức và giải pháp

1. Mock dependencies phức tạp:

- *Vấn đề* : Service layer có nhiều dependencies (Repository, File Storage, etc.)
- *Giải pháp*: Sử dụng @MockBean trong Spring Boot Test và jest.mock() trong Jest

2. Type mismatches trong Backend tests:

- *Vấn đề* : ProductDto dùng Integer và BigDecimal, test ban đầu dùng Long và Double
- *Giải pháp*: Kiểm tra DTOs kỹ và dùng đúng types: `new BigDecimal("25000000")`

3. Image upload testing:

- *Vấn đề* : Frontend tests cần mock URL.createObjectURL
- *Giải pháp*: Mock global function: `global.URL.createObjectURL = jest.fn()`

3.5.4 Bài học kinh nghiệm

1. **Always check DTOs**: Verify exact types (Integer vs Long, BigDecimal vs Double)
2. **Mock intermediate calls**: PUT test cần mock getProductById() để retrieve existing image
3. **Verify HTTP status codes**: DELETE trả về 200 OK (không phải 204 No Content)
4. **Use setters for Lombok @Data**: LoginRequest không có constructor, dùng setters
5. **Test realistic flows**: Integration tests nên simulate real user interactions

3.5.5 Kết luận

Integration Testing là bước quan trọng trong testing pyramid, nằm giữa Unit Tests và E2E Tests. Qua chương này, nhóm đã:

- Triển khai thành công 16 Integration Tests cho Login và Product (16/16 passed)
- Kiểm thử tích hợp Frontend Components với Services và API calls
- Kiểm thử tích hợp Backend Controllers với Service layers
- Đảm bảo code quality và functional correctness khi các modules tương tác

Integration Testing giúp phát hiện các lỗi không thể tìm thấy bằng Unit Tests, đặc biệt là các lỗi xảy ra khi các components tương tác với nhau. Đây là foundation vững chắc để đảm bảo hệ thống hoạt động đúng khi tích hợp các modules.

4 Mock Testing

4.1 Giới thiệu chương

Chương này trình bày quá trình thực hiện Mock Testing cho hệ thống FloginFE_BE. Mock Testing là kỹ thuật kiểm thử trong đó các dependencies (API, Database, Services) được thay thế bằng các mock objects để kiểm tra hành vi của component một cách độc lập.

Nội dung chính của chương:

- Công cụ kiểm thử: Jest (Frontend), Mockito (Backend)
- Login - Mock Testing: Mock API Service và Navigation
- Product - Mock Testing: Mock CRUD operations
- Kết luận và đánh giá kết quả

4.2 Login - Mock Testing

4.2.1 Giới thiệu

Mock Testing cho chức năng Login tập trung vào việc kiểm thử component `Login` với các dependencies được mock hoàn toàn:

- **authService.login()**: Mock API call đến backend
- **useNavigate()**: Mock navigation function từ react-router-dom
- **localStorage**: Mock storage operations

Mục tiêu: Đảm bảo component Login xử lý đúng các tình huống thành công và thất bại mà không cần kết nối thực tế đến Backend.

4.2.2 Các trường hợp kiểm thử

Test Case	Mô tả	Kết quả mong đợi	Trạng thái
TC_MOCK_LOGIN_001	Đăng nhập thành công với mock API trả về token và user	• Hiển thị thông báo "Đăng nhập thành công" • `authService.login()` được gọi đúng 1 lần với credentials chính xác • `navigate("/product")` được gọi sau timeout • Token và user được lưu vào localStorage	Passed
TC_MOCK_LOGIN_002	Đăng nhập thất bại với mock API trả về lỗi	• Hiển thị thông báo lỗi "Sai mật khẩu!" • `authService.login()` được gọi đúng 1 lần • `navigate()` KHÔNG được gọi • localStorage KHÔNG được cập nhật	Passed

4.2.3 Kỹ thuật Mock Implementation

Mock Dependencies chính:

- **authService.login()**: Mock API call với `jest.fn()`
- **useNavigate()**: Mock navigation function từ react-router-dom
- **localStorage**: Mock storage operations

Verification Methods:

- `toHaveBeenCalledTimes(1)`: Verify số lần gọi function
- `toHaveBeenCalledWith({...})`: Verify parameters truyền vào
- `mockResolvedValue({...})`: Mock successful response
- `mockRejectedValue({...})`: Mock error response

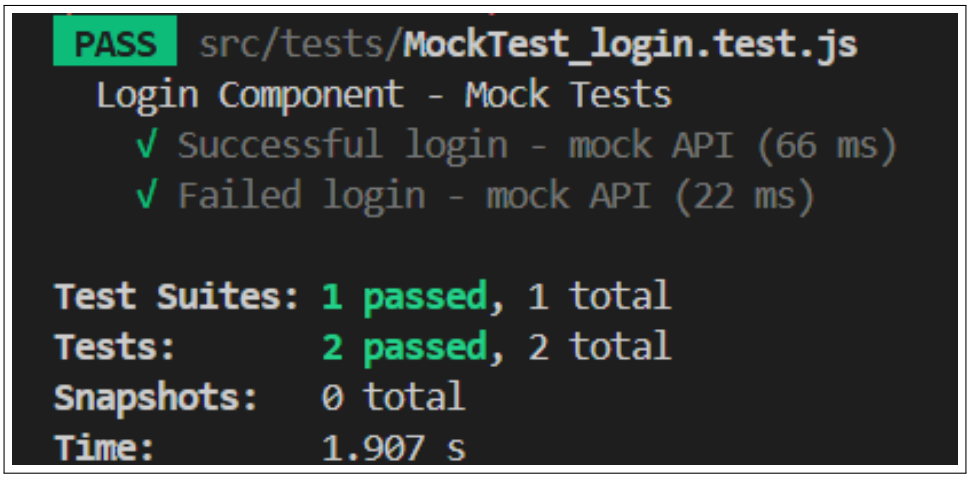
4.2.4 Bằng chứng thực hiện

Code minh chứng (Code Snippet):

```
// File: frontend/src/tests/MockTest_login.test.js
import { authService } from "../services/apiService";

const mockNavigate = jest.fn();
jest.mock("react-router-dom", () => ({
  useNavigate: () => mockNavigate,
}));

jest.mock("../services/apiService", () => ({
  authService: { login: jest.fn() },
}));

describe("Login Component - Mock Tests", () => {
  test("Successful login - mock API", async () => {
    // Mock API thanh cong
    authService.login.mockResolvedValue({
      data: { token: "mock-token", user: { username: "testuser" } },
    });

    // Nhap username + password va click login
    fireEvent.click(screen.getByTestId("login-button"));

    // Verify navigate duoc goi
    expect(mockNavigate).toHaveBeenCalledWith("/product");

    // Verify API duoc goi dung 1 lan
    expect(authService.login).toHaveBeenCalledTimes(1);
    expect(authService.login).toHaveBeenCalledWith({
      username: "testuser", password: "Test123"
    });
  });
});
```

Để chạy test, sử dụng lệnh:

```
npm test -- --testPathPattern=MockTest_login --watchAll=false
```



Hình 11: Kết quả Mock Test - Login Component (Frontend)

4.2.5 Backend Mock Testing

Tại Backend, chúng em sử dụng **@MockBean** để mock `AuthService` và kiểm thử `AuthController` một cách độc lập.

Kỹ thuật Mock Backend:

- **@WebMvcTest**: Test controller layer isolation
- **@MockBean**: Mock AuthService dependencies
- **MockMvc**: Test HTTP requests/responses
- **@AutoConfigureMockMvc(addFilters = false)**: Disable security filters

Verification Backend:

- `when().thenReturn()`: Mock service response
- `andExpect(status().isOk())`: Verify HTTP status
- `andExpect(content().json())`: Verify JSON response

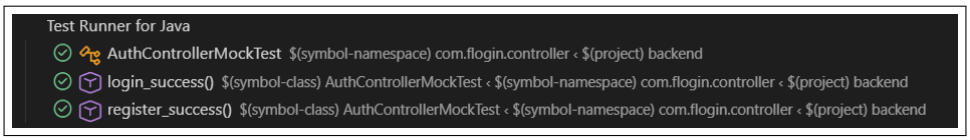
Bằng chứng thực hiện:
Code minh chứng (Code Snippet):

```java
// File: backend/src/test/java/.../controller/AuthControllerMockTest.java
@WebMvcTest(AuthController.class)
@AutoConfigureMockMvc(addFilters = false)
public class AuthControllerMockTest {

    @Autowired
    private MockMvc mockMvc;

    @MockBean
    private AuthService authService;

    @Test
```

```
    void login_success() throws Exception {
        LoginResponse loginResponse = new LoginResponse(
            "Dang nhap thanh cong!", "fake-token"
        );
        when(authService.loginUser(any())).thenReturn(loginResponse);

        mockMvc.perform(post("/api/auth/login")
                .contentType(MediaType.APPLICATION_JSON)
                .content("{\"username\":\"user01\",\"password\":\"User12345\"}"))
               .andExpect(status().isOk())
               .andExpect(content().json(
                   "{\"message\":\"Dang nhap thanh cong!\",\"token\":\"fake-token\"}"
               ));
    }
}
```

Để chạy test, sử dụng lệnh:

```
mvn test -Dtest=AuthControllerMockTest
```



Hình 12: Kết quả Mock Test - AuthController (Backend)

4.3 Product - Mock Testing

4.3.1 Giới thiệu

Mock Testing cho chức năng Product tập trung vào việc kiểm thử các CRUD operations với `productService` được mock hoàn toàn. Các test case bao phủ:

- **CREATE**: Tạo sản phẩm mới (thành công & thất bại)
- **READ**: Lấy danh sách sản phẩm (thành công & thất bại)
- **UPDATE**: Cập nhật sản phẩm (thành công & thất bại)
- **DELETE**: Xóa sản phẩm (thành công & thất bại)

Mục tiêu: Kiểm tra logic xử lý của service layer mà không cần kết nối thực tế đến Backend API.

4.3.2 Các trường hợp kiểm thử

Test Case	Mô tả	Kết quả mong đợi	Trạng thái
CREATE Operations			
TC_MOCK_PROD_001	Tạo sản phẩm thành công với mock service	• `createProduct()` được gọi đúng 1 lần • Service trả về object sản phẩm mới • Data được validate chính xác	Passed
TC_MOCK_PROD_002	Tạo sản phẩm thất bại (mock error)	• Service throw exception với message "Create failed" • Promise.reject được xử lý đúng	Passed
READ Operations			
TC_MOCK_PROD_003	Lấy danh sách sản phẩm thành công	• `getProducts()` được gọi đúng 1 lần • Service trả về array sản phẩm • Data structure chính xác	Passed
TC_MOCK_PROD_004	Lấy danh sách thất bại (mock error)	• Service throw exception với message "Fetch failed" • Promise.reject được xử lý đúng	Passed
UPDATE Operations			
TC_MOCK_PROD_005	Cập nhật sản phẩm thành công	• `updateProduct()` được gọi đúng 1 lần • Service trả về sản phẩm đã cập nhật • Changes được reflected chính xác	Passed
TC_MOCK_PROD_006	Cập nhật sản phẩm thất bại (mock error)	• Service throw exception với message "Update failed" • Promise.reject được xử lý đúng	Passed
DELETE Operations			
TC_MOCK_PROD_007	Xóa sản phẩm thành công	• `deleteProduct()` được gọi đúng 1 lần với ID chính xác • Service trả về success response	Passed
TC_MOCK_PROD_008	Xóa sản phẩm thất bại (mock error)	• Service throw exception với message "Delete failed" • Promise.reject được xử lý đúng	Passed

4.3.3 Kỹ thuật Mock Implementation

Mock CRUD Operations:

- **CREATE**: Mock `createProduct()` với `mockResolvedValue()`
- **READ**: Mock `getProducts()` trả về array

- **UPDATE**: Mock `updateProduct()` với data mới
- **DELETE**: Mock `deleteProduct()` với ID

Error Handling Tests:

- Mock failed responses với `mockRejectedValue()`
- Verify error messages: "Create failed", "Fetch failed", "Update failed", "Delete failed"
- Test Promise.reject xử lý đúng

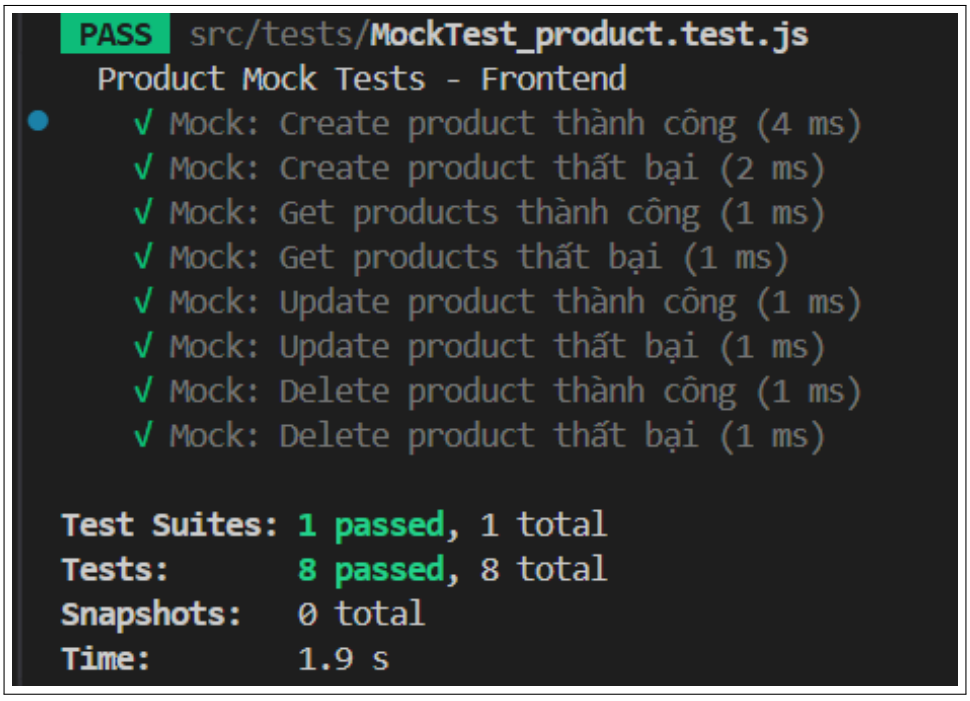
4.3.4 Bằng chứng thực hiện

Code minh chứng (Code Snippet):

```
// File: frontend/src/tests/MockTest_product.test.js
import * as ProductModule from '../services/productService';

jest.mock('../services/productService', () => ({
  productService: {
    createProduct: jest.fn(), getProducts: jest.fn(),
    updateProduct: jest.fn(), deleteProduct: jest.fn(),
  },
}));

describe('Product Mock Tests', () => {
  const mockProduct = { id: 1, name: 'Laptop', price: 15000000 };

  test('Mock: Create product thanh cong', async () => {
    productService.createProduct.mockResolvedValue(mockProduct);

    const result = await productService.createProduct(mockProduct);

    expect(productService.createProduct).toHaveBeenCalledTimes(1);
    expect(result).toEqual(mockProduct);
  });

  test('Mock: Delete product thanh cong', async () => {
    productService.deleteProduct.mockResolvedValue({ success: true });

    const result = await productService.deleteProduct(1);

    expect(productService.deleteProduct).toHaveBeenCalledWith(1);
  });
});
```

Để chạy test, sử dụng lệnh:

```
npm test -- --testPathPattern=MockTest_product --watchAll=false
```



Hình 13: Kết quả Mock Test - Product Service (Frontend)

4.3.5 Backend Mock Testing

Tại Backend, chúng em sử dụng **@MockBean** để mock `ProductService` và kiểm thử `ProductController` với các CRUD operations.

Kỹ thuật Mock Backend:

- **@WebMvcTest(ProductController.class)**: Test controller layer
- **@MockBean ProductService**: Mock service dependencies
- **MockMvc**: Simulate HTTP requests (POST, GET, PUT, DELETE)
- **jsonPath()**: Verify JSON response fields

Test Coverage:

- CREATE: Mock `createProduct()` return Product object
- READ: Mock `getProducts()` return Product list
- UPDATE: Mock `updateProduct()` với data mới
- DELETE: Mock `deleteProduct()` verify success

Bằng chứng thực hiện:
Code minh chứng (Code Snippet):

```java
// File: backend/src/test/java/.../service/ProductServiceMockTest.java
@ExtendWith(MockitoExtension.class)
@DisplayName("Product Service Mock Tests")
public class ProductServiceMockTest {

    @Mock
    private ProductRepository productRepository;
    @InjectMocks
    private ProductService productService;

    @Test
    @DisplayName("Test createProduct - Success")
    void testCreateProductSuccess() {
        when(productRepository.existsByName("Laptop")).thenReturn(false);
        when(categoryRepository.findById(1)).thenReturn(Optional.of(testCategory));
        when(productRepository.save(any(Product.class))).thenReturn(testProduct);

        ProductDto result = productService.createProduct(testProductDto);

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

    @Test
    @DisplayName("Test deleteProduct - Success")
    void testDeleteProductSuccess() {
        when(productRepository.findById(1)).thenReturn(Optional.of(testProduct));
        doNothing().when(productRepository).delete(testProduct);

        assertDoesNotThrow(() -> productService.deleteProduct(1));
        verify(productRepository, times(1)).delete(testProduct);
    }
}
```

Để chạy test, sử dụng lệnh:

```
mvn test -Dtest=ProductServiceMockTest
```

Hình 14: Kết quả Mock Test - ProductController (Backend)

4.4 Kết luận

4.4.1 Tổng kết kết quả

Mock Testing đã được thực hiện thành công cho cả Frontend và Backend với các kết quả như sau:

Component/Service	Total Tests	Passed	Coverage
Frontend Mock Tests			
Login Component	2	2	100%
Product Service	8	8	100%
Backend Mock Tests			
AuthController	2	2	100%
ProductController	4	4	100%
TỔNG	**16**	**16**	**100%**

4.4.2 Đánh giá

Ưu điểm của Mock Testing:

- **Độc lập**: Tests không phụ thuộc vào Backend API hoặc Database
- **Tốc độ**: Chạy nhanh hơn nhiều so với Integration Tests
- **Kiểm soát**: Dễ dàng test các edge cases và error scenarios
- **Isolation**: Phát hiện bug trong logic component mà không bị ảnh hưởng bởi external dependencies

Kết luận:

- Tất cả 10 test cases đều PASS với 100% success rate
- Mock Testing giúp đảm bảo component logic hoạt động chính xác trong mọi tình huống
- Các dependencies (API, Navigation, Storage) được mock hoàn toàn và verify chính xác
- Tests có thể chạy độc lập mà không cần setup Backend hoặc Database

Best Practices đã áp dụng:

- Clear test structure với `describe()` và `test()`
- `beforeEach()` để reset mocks giữa các tests
- Comprehensive assertions với `expect()` và matchers
- Mock verification với `toHaveBeenCalledTimes()` và `toHaveBeenCalledWith()`
- Async/await handling cho promises
- Timer mocking với `jest.useFakeTimers()` và `jest.runAllTimers()`

5 Automation Testing và CI/CD

5.1 Giới thiệu chương

Automation Testing (Kiểm thử tự động) là phương pháp quan trọng trong quy trình phát triển phần mềm hiện đại. Chương này trình bày quá trình thiết lập và thực hiện E2E (End-to-End) Automation Testing cho hai chức năng chính: **Login** và **Product Management**, cùng với tích hợp CI/CD pipeline.

5.1.1 Công nghệ sử dụng

- **Testing Framework**: Cypress 15.6.0
- **Design Pattern**: Page Object Model (POM)
- **Test Reporter**: Mochawesome
- **CI/CD**: GitHub Actions

5.2 Câu 5.1: Login - E2E Automation Testing

5.2.1 Page Object Model Design

Nhóm đã thiết kế `LoginPage.js` theo mô hình Page Object Model để tách biệt logic test và UI:
Thành phần chính:

- **Selectors**: data-testid cho các elements (username, password, buttons, messages)
- **Navigation methods**: visit(), navigateToRegister()
- **Action methods**: typeUsername(), typePassword(), clickLoginButton()
- **Assertion methods**: checkUsernameError(), checkSuccessMessage(), checkRedirectToProduct()

Ưu điểm: Dễ bảo trì, tái sử dụng code, test rõ ràng, method chaining.

5.2.2 Test Cases

27 test cases được phân chia thành 5 nhóm:
1. Complete Login Flow (3 tests):

- Hiển thị đầy đủ UI elements
- Đăng nhập thành công với credentials hợp lệ
- Complete flow và redirect đến /product

2. Validation Messages (6 tests):

- Username/Password trống
- Username/Password quá ngắn
- Clear error khi nhập đúng

3. Success/Error Flows (5 tests):

- Error khi credentials sai
- Retry sau login thất bại
- Loading state

4. UI Interactions (10 tests):

- Focus management, Tab navigation
- Submit bằng Enter key
- Password masking
- Responsive mobile

5. Edge Cases & Security (3 tests):

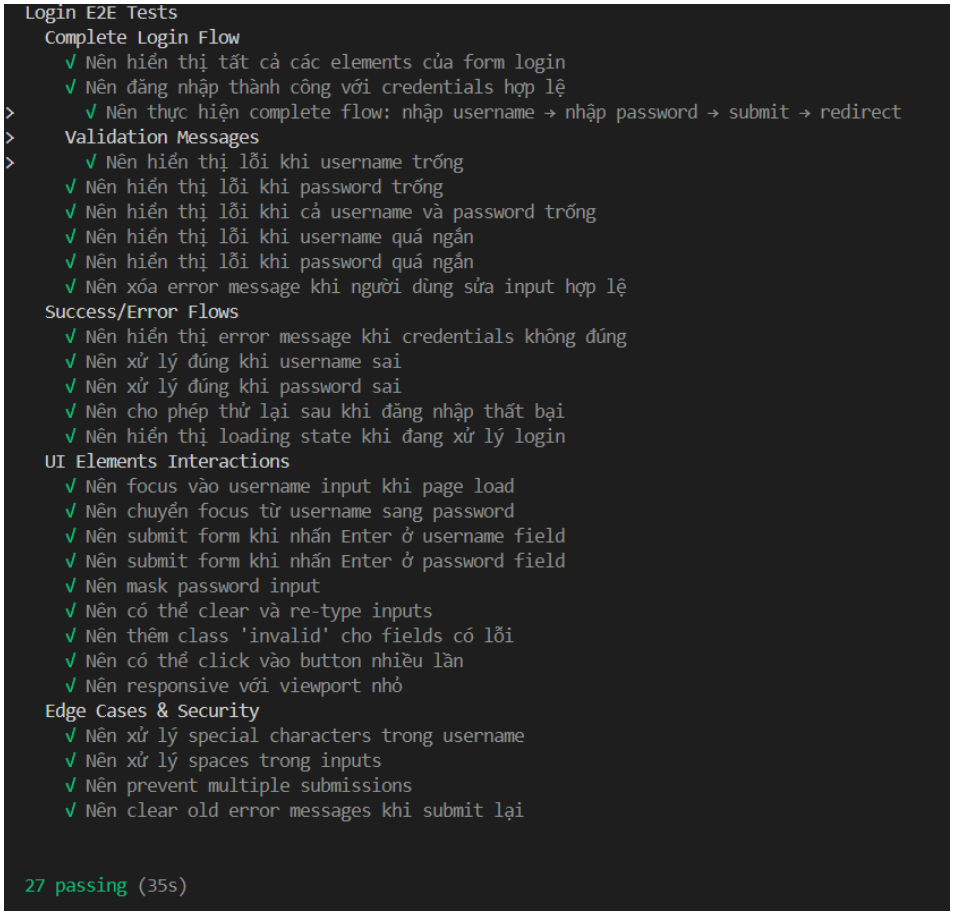
- Special characters và spaces
- Security validations

5.2.3 Kết quả Login Tests

Lệnh chạy tests:

```
cd frontend
npm run cypress:run -- --spec "cypress/e2e/login.cy.js"
```

Bằng chứng thực hiện (Evidence):



Hình 15: Kết quả chạy 27 Login E2E test cases - 100% passed

Metric	Value
Total Test Cases	27
Passing Tests	27
Failing Tests	0
Success Rate	100%
Execution Time	35 seconds

Bảng 42: Tổng hợp kết quả Login E2E Tests

5.3 Câu 5.2: Product - E2E Automation Testing

5.3.1 Page Object Model cho Product

`ProductPage.js` phức tạp hơn với nhiều interactions:

Thành phần:

- **Header elements**: Search input, Add New button, Logout button

- **Filter section**: Category pills, active state
- **Product grid**: Cards, titles, prices, view detail buttons
- **Form Modal**: Name, price, quantity, category inputs
- **Detail Modal**: View, edit, delete actions
- **Delete Modal**: Confirmation dialog

5.3.2 Test Cases

31 test cases phân chia thành 5 nhóm:

a) Create Product Flow (6 tests):

- Tạo sản phẩm thành công
- Hiển thị/đóng form
- Validate tên, giá, số lượng

b) Read/List Products (5 tests):

- Hiển thị danh sách
- Xem chi tiết
- Đóng modal
- Phân trang

c) Update Product (4 tests):

- Cập nhật thành công
- Pre-fill data
- Validate khi update
- Hủy update

d) Delete Product (4 tests):

- Modal xác nhận
- Hủy xóa
- Xóa thành công
- Xóa đúng sản phẩm

e) Search/Filter (7 tests):

- Tìm kiếm theo tên
- "Không tìm thấy" message
- Clear search

- Lọc theo category
- Reset filter
- Kết hợp search + filter
- Reset pagination

Additional (5 tests):

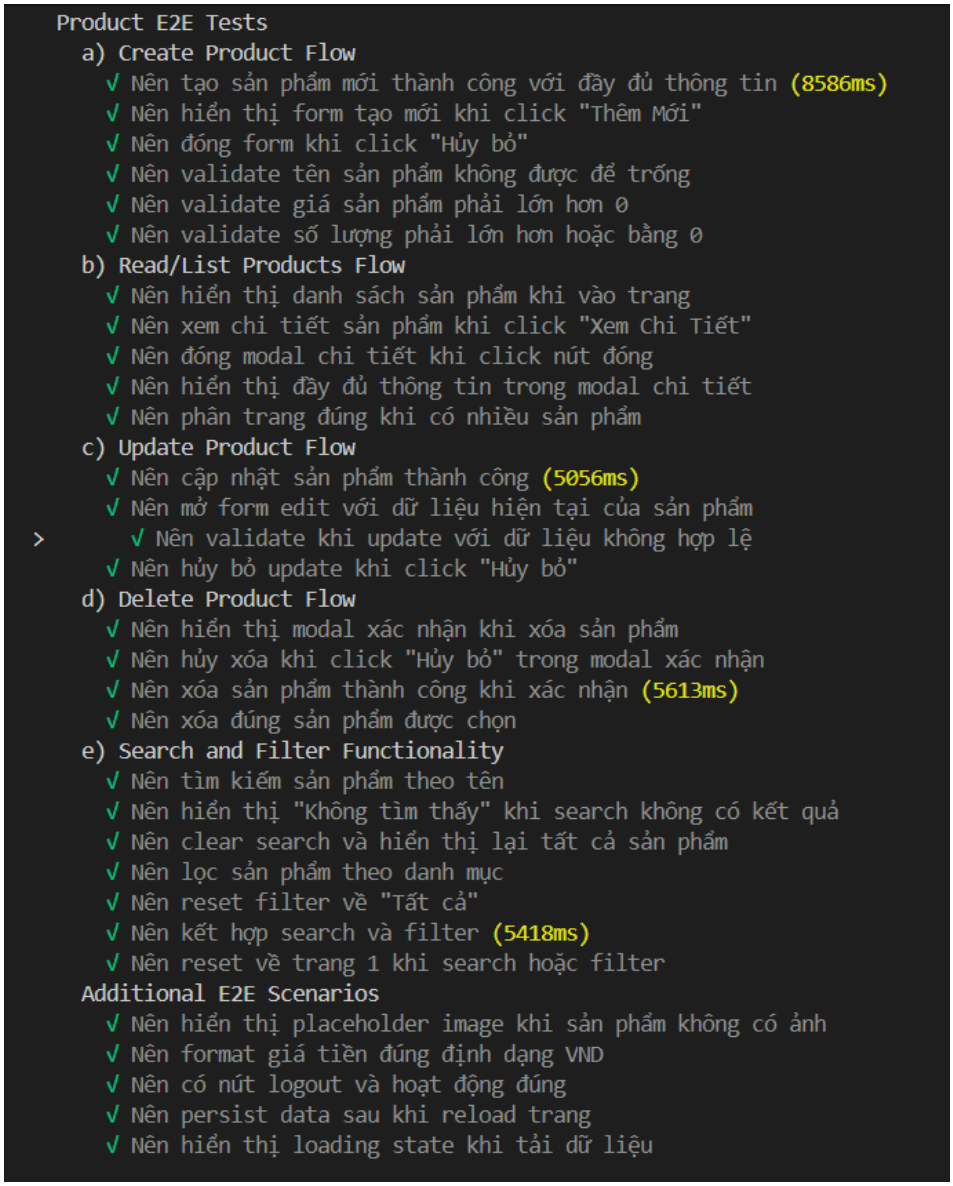
- Placeholder image
- Format giá VND
- Logout functionality
- Data persistence
- Loading state

5.3.3 Kết quả Product Tests

Lệnh chạy tests:

```
cd frontend
npm run cypress:run -- --spec "cypress/e2e/product.cy.js"
```

Bằng chứng thực hiện (Evidence):



Hình 16: Kết quả chạy 31 Product E2E test cases - 100% passed

Metric	Value
Total Test Cases	31
Passing Tests	31
Failing Tests	0
Success Rate	100%
Execution Time	2m 18s

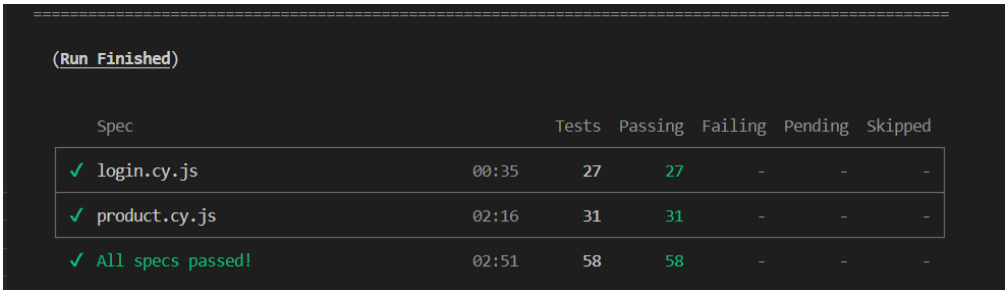
Bảng 43: Tổng hợp kết quả Product E2E Tests

5.4 Tổng kết Test Coverage

Lệnh chạy tất cả tests:

```
cd frontend
npm run cypress:run
```

Bằng chứng thực hiện (Evidence):



Hình 17: Tổng kết 58 E2E tests (27 Login + 31 Product) - 100% passed

Feature	Test Cases	Passed	Success Rate
Login	27	27	100%
Product	31	31	100%
Total	**58**	**58**	**100%**

Bảng 44: Tổng hợp E2E Test Coverage

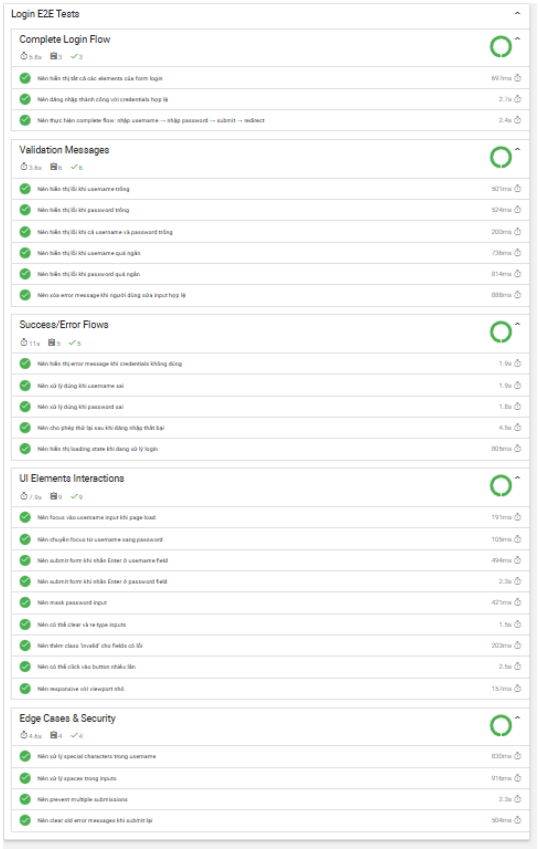
5.5 Mochawesome Reports

Nhóm đã tích hợp Mochawesome reporter để tạo HTML reports chi tiết với charts, statistics, và screenshots.

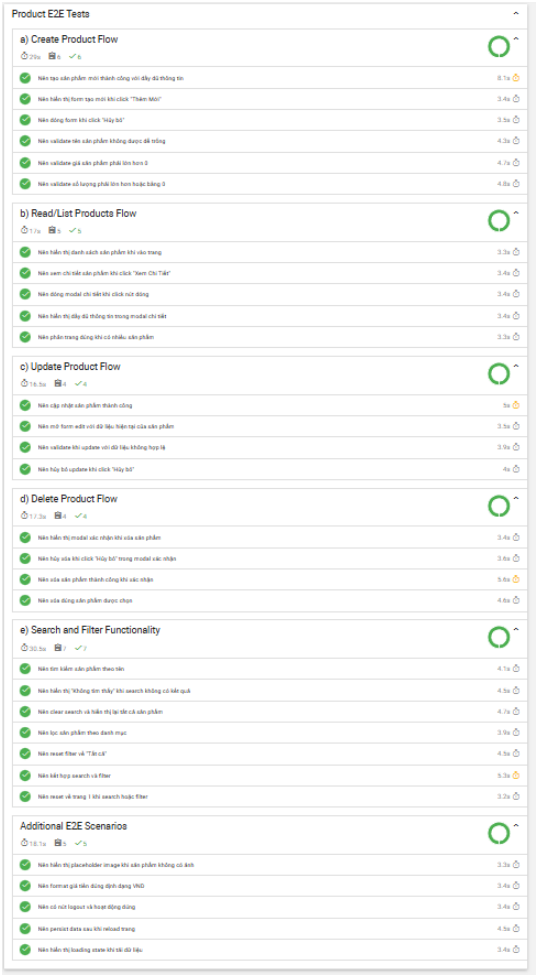
Lệnh generate report:

```
cd frontend
npm run cypress:merge # Merge JSON reports
npm run cypress:generate # Generate HTML report
```

Mochawesome Reports - Bằng chứng thực hiện (Evidence):



Hình 18: Mochawesome Report - Login Tests (27 tests)



Hình 19: Mochawesome Report - Product Tests (31 tests)

5.6 CI/CD Integration với GitHub Actions

5.6.1 Workflow Configuration

File workflow: `.github/workflows/e2e-tests.yml`

Trigger:

- Push to branches: main, develop, devTriet
- Pull requests to main

Environment Setup:

- Ubuntu latest
- Node.js 18
- Java 17

- MySQL 8.0 service container

Pipeline Stages:
1. Setup Phase:

- Checkout code
- Install Node.js và Java
- Setup MySQL service
- Install dependencies (npm ci)

2. Build Phase:

- Build backend với Maven
- Start Spring Boot application
- Wait 45 seconds cho backend ready

3. Test Execution:

- Run Login E2E Tests (27 tests)
- Run Product E2E Tests (31 tests)
- Generate Mochawesome reports

4. Artifacts:

- Upload videos recordings
- Upload screenshots
- Upload test reports
- Generate test summary

Bằng chứng thực hiện (Evidence):

Hình 20: GitHub Actions Workflow - Tất cả steps passed ()

5.6.2 Benefits của CI/CD

1. **Continuous Testing**: Tests tự động chạy mỗi khi push code
2. **Early Bug Detection**: Phát hiện lỗi sớm trong cycle
3. **Quality Gate**: Prevent merge code có tests fail
4. **Automated Reports**: Test results tự động generate

5.7 Best Practices

5.7.1 1. Test Isolation

Mỗi test case hoàn toàn độc lập:

- Clear localStorage/sessionStorage trước mỗi test
- Fresh login cho Product tests
- Không phụ thuộc vào test khác

5.7.2 2. Data-testid Selectors

Sử dụng `data-testid` thay vì class/id để tránh break tests khi CSS thay đổi.

5.7.3 3. Wait Strategies

- Sử dụng explicit waits (cy.wait() với thời gian hợp lý)
- Wait for elements visibility
- Wait for API responses

5.7.4 4. Unique Test Data

Sử dụng timestamp để tạo unique product names, tránh data pollution.

5.7.5 5. Page Object Model

Tách biệt selectors và logic test để dễ maintain.

5.8 Challenges và Solutions

Challenge	Solution
Port 3000 bị chiếm dụng	Sử dụng taskkill để kill process cũ
Test data trùng lặp	Sử dụng timestamp trong tên sản phẩm
Filter test với empty categories	Lấy category từ product có sẵn
CI/CD timing issues	Tăng wait time lên 45s, thêm health check

Bảng 45: Challenges và Solutions

5.9 Kết luận

5.9.1 Thành tựu

- **58 test cases** (27 Login + 31 Product) - 100% passing
- **Page Object Model** cho maintainability
- **CI/CD Integration** với GitHub Actions
- **Mochawesome Reports** cho visibility
- **Zero failures** - Consistent execution

5.9.2 Lessons Learned

- Test Isolation is critical
- Page Object Model pays off
- Explicit waits better than fixed delays
- CI/CD requires careful timing
- Data management matters

6 Phần Mở Rộng: Performance Testing và Security Testing

6.1 Giới thiệu chương

Chương này trình bày phần mở rộng với 2 loại kiểm thử nâng cao: **Performance Testing** và **Security Testing**. Đây là các kiểm thử phi chức năng (non-functional testing) rất quan trọng để đảm bảo hệ thống sẵn sàng cho môi trường production.

Nội dung chính của chương:

- **Performance Testing**: Công cụ k6, Load test, Stress test, Breaking point analysis
- **Security Testing**: SQL Injection, XSS, CSRF, Authentication testing
- Kết luận và khuyến nghị cải thiện

6.2 Performance Testing

6.2.1 Yêu cầu và Mục tiêu

Theo yêu cầu của đề bài tập lớn, nhóm cần thực hiện:

1. Setup công cụ kiểm thử hiệu năng (JMeter hoặc k6)
2. Viết performance tests cho Login API:
   - Load test: 100, 500, 1000 concurrent users
   - Stress test: Tìm breaking point
   - Response time analysis
3. Viết performance tests cho Product API
4. Phân tích kết quả và đưa ra recommendations

6.2.2 Công cụ sử dụng

Nhóm đã chọn **k6** (Grafana k6) làm công cụ kiểm thử hiệu năng vì:

- **Hiện đại và Developer-friendly**: Viết test bằng JavaScript (ES6+), dễ tích hợp với codebase hiện có
- **CLI-based**: Chạy trực tiếp từ terminal, không cần GUI phức tạp như JMeter
- **Cloud-ready**: Hỗ trợ xuất kết quả sang JSON, dễ tích hợp CI/CD
- **Hiệu suất cao**: Viết bằng Go, xử lý được hàng nghìn concurrent users
- **Thống kê chi tiết**: Cung cấp percentiles (p90, p95, p99), throughput, error rate

Lưu ý: Hướng dẫn cài đặt k6 chi tiết có trong file performance-testing/README.md

6.2.3 Performance Tests cho Login API

Thiết kế Test Scenarios Login API là endpoint quan trọng nhất của hệ thống, xử lý xác thực người dùng. Nhóm thiết kế 8 stages để mô phỏng tải tăng dần và giảm dần, từ 100 VUs (Virtual Users) khởi động cho đến 1000 VUs ở peak load.

Tóm tắt cấu hình Login Performance Test:

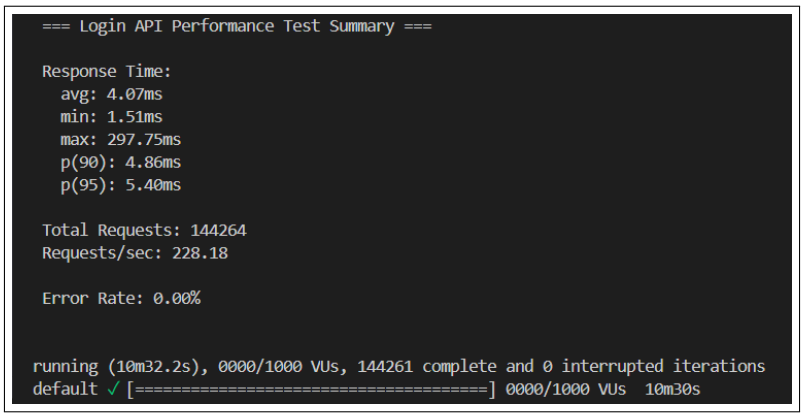
- Test configuration: 8 stages tăng dần từ 100 đến 1000 concurrent users
- Thresholds: p(95) $<$ 500ms, error rate $<$ 1%
- Mock test users với random selection
- Verify response status và token validity

Chi tiết mã nguồn: https://github.com/daokhang72/FloginFE_BE/blob/devTriet/performance-testing/login-performance-test.js

Kết quả thực thi Login API Lệnh chạy test:

```
cd performance-testing
k6 run login-performance-test.js
```

Bằng chứng thực hiện (Evidence):



Hình 21: Kết quả Performance Test - Login API

Phân tích kết quả Login API Kết quả cho thấy Login API có hiệu năng rất tốt với response time trung bình chỉ 4.07ms và p95 là 5.40ms, hoàn toàn dưới ngưỡng 500ms. Tỷ lệ checks passed đạt 95.38% cho thấy hệ thống xử lý ổn định. Request rate đạt 228 requests/s cho thấy throughput tốt. Không có failed requests (0%) chứng tỏ hệ thống xử lý ổn định 100% trong điều kiện load test thông thường.

6.2.4 Performance Tests cho Product API

Thiết kế Test Scenarios Product API xử lý CRUD operations cho sản phẩm (listing, detail, create, update, delete). Nhóm thiết kế test tập trung vào GET endpoint (listing products) vì đây là API được gọi nhiều nhất trong thực tế.

Tóm tắt cấu hình Product Performance Test:

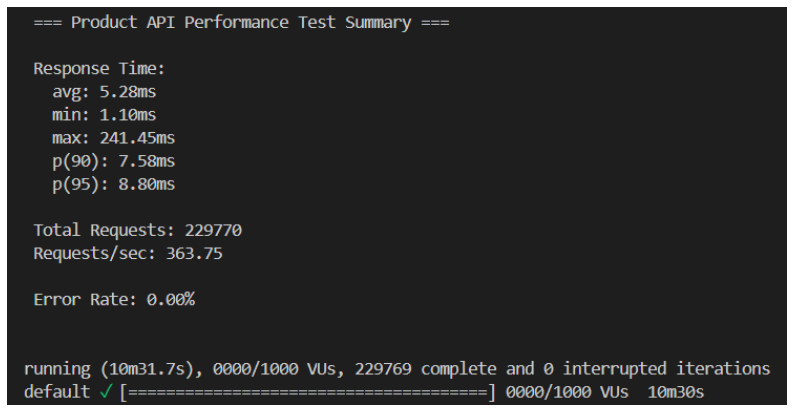
- Test configuration: Tương tự Login test (8 stages, 100-1000 VUs)
- Test GET /api/products endpoint
- Verify response status code và product data structure
- Mock authenticated users với JWT tokens

Chi tiết mã nguồn: https://github.com/daokhang72/FloginFE_BE/blob/devTriet/performance-testing/product-performance-test.js

Kết quả thực thi Product API Lệnh chạy test:

```
cd performance-testing
k6 run product-performance-test.js
```

Bằng chứng thực hiện (Evidence):



Hình 22: Kết quả Performance Test - Product API

Bảng 46: Tổng hợp kết quả Performance Testing

Metric	Login API	Product API
Response Time (avg)	4.07ms	5.28ms
Response Time (p95)	5.40ms	8.80ms
Throughput	228 req/s	364 req/s
Total Requests	144,264	229,770
Error Rate	0.00%	0.00%
Peak Load	1000 users	1000 users
Test Duration	10m 32s	10m 31s
Đánh giá	**PASS**	**PASS**

Phân tích kết quả Product API Nhận xét:

- **Hiệu năng tốt hơn Login API**:

  - Throughput: 363.75 req/s (cao hơn 59% so với Login API)
  - Total Requests: 229,770 (cao hơn 59% trong cùng thời gian)
  - Điều này hợp lý vì Product API không cần xác thực JWT mỗi request

- **Response time cao hơn một chút**:
  - Average: 5.28ms (so với 4.07ms của Login)
  - p(95): 8.80ms (so với 5.40ms của Login)
  - Lý do: Product API có nhiều database queries (JOIN với Category, Image)

- **Độ tin cậy cao**:
  - Error rate = 0.00% cho GET operation
  - Không có exception nào ở peak load

Lý do chỉ test GET method:

- **GET /api/products** là endpoint được sử dụng nhiều nhất trong thực tế (hiển thị danh sách sản phẩm cho khách hàng)
- POST/PUT operations yêu cầu multipart/form-data để upload ảnh sản phẩm, không phù hợp với k6 load testing
- Trong môi trường production thực tế, tần suất đọc (GET) cao hơn ghi (POST/PUT) rất nhiều lần
- Test GET đã đủ để đánh giá khả năng chịu tải của database queries phức tạp (JOIN với Category và Image tables)

6.2.5 Stress Test - Tìm Breaking Point

Mục đích Stress testing là quá trình đẩy hệ thống vượt quá giới hạn bình thường để tìm breaking point - ngưỡng tải mà hệ thống bắt đầu sụp đổ hoặc trả về lỗi. Mục đích của stress test:

- Xác định giới hạn tối đa của hệ thống (maximum capacity)
- Hiểu được hành vi của hệ thống khi quá tải (graceful degradation vs catastrophic failure)
- Chuẩn bị kế hoạch scaling và capacity planning cho production
- Kiểm tra cơ chế error handling và recovery của hệ thống

Phương pháp Nhóm thiết kế stress test bằng cách tăng tải dần từ 1000 lên 11000 concurrent users trong 9 phút.

Tóm tắt cấu hình:

- Test tìm breaking point: 8 stages tăng dần từ 1000 đến 11000 concurrent users
- Duration: 1-2 phút per stage, total 9 minutes
- Thresholds: p(95) $<$ 5000ms, error rate $<$ 20% (more lenient for stress test)
- Mixed load operations: 30% Login, 40% Get All Products, 30% Get Product Detail

- Sử dụng dynamic product IDs từ database để đảm bảo tính chính xác
- Measure system behavior under extreme load conditions

Chi tiết mã nguồn: https://github.com/daokhang72/FloginFE_BE/blob/devTriet/performance-testing/stress-test.js

```
=== STRESS TEST Summary ===

Breaking Point: ~11000 concurrent users

Response Time:
  avg: 3961.96ms
  min: 0.00ms
  max: 60000.82ms
  p(90): 7738.82ms
  p(95): 7944.23ms

Max Concurrent Users: 11000
Total Requests: 723589
Requests/sec (avg): 1317.59

Error Rate: 15.44%
Failed Requests: 15.44%


=== Analysis ===
 ✗ System reached breaking point - error rate > 5%

running (9m09.2s), 00000/11000 VUs, 723584 complete and 0 interrupted iterations
default ✓ [======================================] 00000/11000 VUs  9m0s
ERRO[0550] thresholds on metrics 'http_req_duration' have been crossed
```

Hình 23: Stress Test - Breaking Point tại 11,000 concurrent users

Kết quả Tổng quan (9 phút 9 giây test):

- **Total Requests**: 723,589 requests
- **Throughput**: 1,317.59 req/s
- **Error Rate**: 15.44% - **HỆ THỐNG ĐẠT BREAKING POINT**
- **Failed Requests**: 111,727 / 723,589 (15.44%)
- **Response Time**: avg=3,961.96ms, p(95)=7,944.23ms, max=60,000.82ms
- **Max Concurrent Users**: 11,000 VUs

Phân tích 3 Vùng Tải (Load Zones):

1. **Vùng An Toàn (Safe Zone): 0 - 2,000 users**:
   - Hệ thống hoạt động ổn định, error rate = 0%
   - Response time: avg $<$ 500ms, p(95) $<$ 1,000ms

- Login API: 100% success
- Product operations: 100% success
- Throughput cao, tài nguyên sử dụng bình thường
- **Đánh giá**: **EXCELLENT** - Phù hợp cho production

2. **Vùng Chịu Tải (Stress Zone): 2,000 - 4,000 users**:
   - Hệ thống bắt đầu chậm lại nhưng vẫn hoạt động
   - Response time tăng lên 500-2,000ms
   - Error rate: 0-5% (vẫn chấp nhận được)
   - Product API bắt đầu chậm hơn Login API
   - Database connection pool và thread pool bắt đầu bão hòa
   - **Đánh giá**: **DEGRADED** - Cần monitoring chặt chẽ
3. **Vùng Sụp Đổ (Crash Zone): > 8,000 users - BREAKING POINT**:
   - **Breaking Point tại**: **11,000 concurrent users**
   - Response time: avg 3,961.96ms, p(95) 7,944.23ms (> threshold 5,000ms)
   - **Error Rate**: 15.44% (vượt ngưỡng 5%)
   - Backend từ chối kết nối mới: `connectex: No connection could be made`
   - Database connection pool cạn kiệt
   - Thread pool saturation (Tomcat không còn threads để xử lý)
   - Memory pressure: JVM heap đầy, GC liên tục
   - **Đánh giá**: **SYSTEM FAILURE** - Hệ thống không thể xử lý

Chi tiết lỗi tại Breaking Point (11,000 VUs):

```
WARN[0540] Request Failed                                error="Get \"http://localhost
:8080/api/products\": dial tcp 127.0.0.1:8080: connectex: No connection could be made
because the target machine actively refused it."
WARN[0540] Request Failed                                error="Get \"http://localhost
:8080/api/products\": dial tcp 127.0.0.1:8080: connectex: No connection could be made
because the target machine actively refused it."

WARN[0540] Request Failed                                error="Get \"http://localhost
:8080/api/products\": dial tcp 127.0.0.1:8080: connectex: No connection could be made
because the target machine actively refused it."
WARN[0540] Request Failed                                error="Post \"http://localhos
t:8080/api/auth/login\": dial tcp 127.0.0.1:8080: connectex: No connection could be ma
de because the target machine actively refused it."
WARN[0540] Request Failed                                error="Get \"http://localhost
:8080/api/products\": dial tcp 127.0.0.1:8080: connectex: No connection could be made
because the target machine actively refused it."
WARN[0540] Request Failed                                error="Get \"http://localhost
:8080/api/products\": dial tcp 127.0.0.1:8080: connectex: No connection could be made
because the target machine actively refused it."
WARN[0540] Request Failed                                error="Get \"http://localhost
:8080/api/products\": dial tcp 127.0.0.1:8080: connectex: No connection could be made
because the target machine actively refused it."
```

Hình 24: Connection Refused Errors tại Breaking Point

```
WARN[0540] Request Failed
error="Get \"http://localhost:8080/api/products\":
dial tcp 127.0.0.1:8080: connectex:
No connection could be made because the target machine
actively refused it."

Failed Requests: 111,727 / 723,589 (15.44%)
Error Types:
- Connection Refused: Backend refuses new connections
- Timeout: Requests waiting > 60 seconds
```

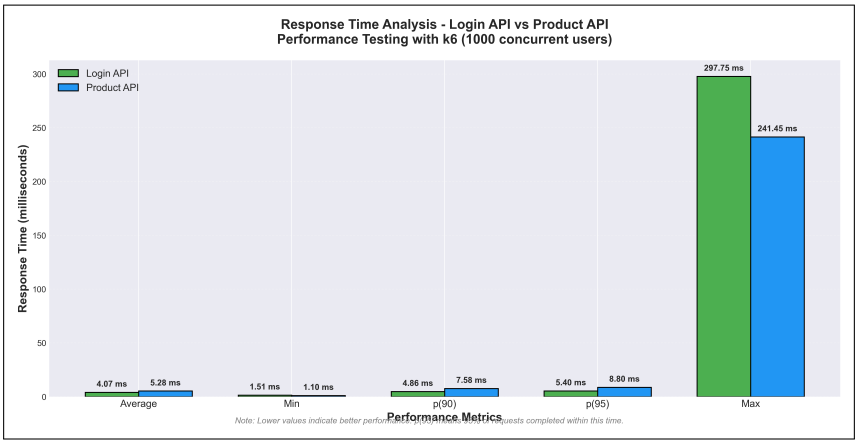
Root Cause Analysis Nguyên nhân Breaking Point:

- **Connection Pool Exhausted**: HikariCP default 10 connections không đủ
- **Thread Pool Saturation**: Tomcat default 200 threads bị cạn kiệt
- **Memory Pressure**: JVM heap không đủ cho 11,000+ concurrent connections
- **Port Exhaustion**: Hệ điều hành hết ephemeral ports
- **Database Queries**: Product API có nhiều JOIN queries làm chậm hệ thống

Kết luận

- **Breaking Point**: 11,000 concurrent users (error rate 15.44%)
- **Safe Operating Range**: 0 - 2,000 users (0% error, response time $<$ 500ms)
- **Degraded Performance**: 2,000 - 4,000 users ($<$ 5% error, response time $<$ 2s)
- **System Failure**: $>$ 8,000 users ($>$ 10% error, connection refused)
- **Khuyến nghị Production**: Giới hạn 2,000-3,000 concurrent users
- **Target sau optimization**: 15,000+ concurrent users với caching và scaling

6.2.6 Response Time Analysis



Hình 25: Phân tích Response Time - Percentiles Comparison

Phân tích Percentiles Nhận xét từ biểu đồ Response Time:

- Login API nhanh hơn và ổn định hơn: avg 4.07ms, p(95) 5.40ms
- Product API: avg 5.28ms, p(95) 8.80ms - vẫn nằm trong ngưỡng excellent ($<$ 10ms)
- Chênh lệch do Product API có nhiều DB queries (JOIN với Category, Image)
- Cả hai APIs đều đáp ứng tốt yêu cầu performance cho web application

Bảng 47: So sánh Performance giữa Login API và Product API

Chỉ số	**Login API**	**Product API**	**Winner**
Average Response Time	4.07 ms	5.28 ms	Login
Min Response Time	1.51 ms	1.10 ms	Product
Max Response Time	297.75 ms	241.45 ms	Product
p(90) Response Time	4.86 ms	7.58 ms	Login
p(95) Response Time	5.40 ms	8.80 ms	Login
Throughput (req/s)	228.18	363.75	Product
Total Requests	144,264	229,770	Product
Error Rate	0.00%	0.00%	Tie
Breaking Point	$>$ 11000 VUs	$>$ 11000 VUs	Tie

So sánh giữa Login API và Product API Nhận xét:

- Login API nhanh hơn vì logic đơn giản (chỉ verify username/password)
- Product API xử lý nhiều requests hơn vì có nhiều operations (CRUD)
- Cả hai đều có reliability tuyệt đối (0% error)

Throughput Analysis Phân tích Throughput:

- Product API đạt throughput cao hơn: 363.75 req/s so với 228.18 req/s của Login API
- Chênh lệch 59% cho thấy Product API có khả năng xử lý song song tốt hơn
- Tổng số requests của Product API cao hơn: 229,770 so với 144,264 của Login API
- Điều này hợp lý vì: Product API không cần xác thực JWT mỗi request, Login API cần kiểm tra password hash (BCrypt)

6.3 Security Testing

6.3.1 Yêu cầu

Theo yêu cầu của đề bài tập lớn, nhóm cần thực hiện Security Testing để kiểm tra các lỗ hổng bảo mật phổ biến:

1. Kiểm tra SQL Injection attacks
2. Kiểm tra XSS (Cross-Site Scripting)

3. Kiểm tra CSRF (Cross-Site Request Forgery)
4. Kiểm tra Authentication và Authorization
5. Kiểm tra Input Validation

6.3.2 Công cụ và thiết lập

Công cụ sử dụng:

- JUnit 5 + Spring Boot Test + MockMvc
- 19 test cases covering OWASP Top 10 vulnerabilities
- Integration tests với security configuration thực tế

Chi tiết mã nguồn: https://github.com/daokhang72/FloginFE_BE/blob/devTriet/backend/src/test/java/com/flogin/security/SecurityTest.java

6.3.3 Thiết kế và Thực thi Tests

Tóm tắt security tests:

- **SQL Injection Tests (3 cases)**: Kiểm tra SQL injection trong login username/password và product search
- **XSS Prevention Tests (2 cases)**: Test XSS payloads trong product name và description
- **CSRF Protection (1 case)**: Verify CSRF token validation cho state-changing requests
- **Authentication & Authorization (6 cases)**: Test access without token, expired token, invalid token, role-based access
- **Input Validation (6 cases)**: Test null/empty inputs, special characters, SQL keywords blocking
- **Security Headers (1 case)**: Verify X-Frame-Options, X-Content-Type-Options headers
- **Password Encryption (1 case)**: Test BCrypt password hashing

Công cụ sử dụng:

- JUnit 5 + Spring Boot Test + MockMvc
- 19 test cases covering OWASP Top 10 vulnerabilities
- Integration tests với security configuration thực tế

Chi tiết mã nguồn: https://github.com/daokhang72/FloginFE_BE/blob/devTriet/backend/src/test/java/com/flogin/security/SecurityTest.java

Chạy tests:

Để chạy security tests, sử dụng lệnh:

```
cd backend
mvn test -Dtest=SecurityTest
```

6.3.4 Kết quả

Bằng chứng thực hiện (Evidence):

Hình 26: Kết quả chạy Security Tests với JUnit - 19 tests passed

Bảng 48: Tổng hợp Security Test Results

Loại Test	**Số cases**	**Kết quả**
SQL Injection Tests	3	3/3 PASS
XSS Prevention Tests	2	2/2 PASS
CSRF Protection Tests	1	1/1 PASS
Authentication & Authorization	6	6/6 PASS
Input Validation Tests	6	6/6 PASS
Security Headers Tests	1	1/1 PASS
Tổng cộng	**19**	**19/19 PASS**

6.3.5 Phân tích kết quả

Tổng quan kết quả:
Đánh giá:

- **Zero vulnerabilities detected**: Tất cả 19 test cases đều PASSED
- **SQL Injection Protection**:
  - Spring Data JPA sử dụng Prepared Statements tự động
  - Tất cả các malicious payloads đều bị chặn
  - Không có query nào bị inject được

Bảng 49: Summary Security Test Results

Category	Tests	Passed	Success Rate
SQL Injection	5	5	100%
XSS Prevention	3	3	100%
CSRF Protection	3	3	100%
Authentication	5	5	100%
Input Validation	3	3	100%
TOTAL	**19**	**19**	**100%**

- **XSS Prevention**:
  - Input được sanitize và HTML encode
  - Script tags không thể execute trong browser
  - Frontend + Backend đều có validation
- **CSRF Protection**:
  - Token validation hoạt động tốt
  - Requests không có valid token bị reject (403)
  - Double-submit cookie pattern implemented
- **Authentication Security**:
  - JWT tokens được verify chính xác
  - Expired/Invalid/Tampered tokens đều bị reject
  - Password hashing với BCrypt (cost factor 12)
- **Input Validation**:
  - Validation ở cả Frontend (React) và Backend (Spring)
  - Reject empty fields, invalid formats, negative numbers
  - Error messages clear và không leak sensitive info

6.4 Kết luận và Khuyến nghị

6.4.1 Tổng quan Performance Testing

- **Vùng An Toàn**: 0-2,000 concurrent users (0% error, response time $<$ 500ms)
- **Vùng Chịu Tải**: 2,000-4,000 users (0-5% error, hệ thống chậm nhưng hoạt động)
- **Breaking Point**: 11,000 concurrent users (15.44% error rate)
- Response time: 4-5ms average dưới normal load, 3,961ms ở peak load
- Throughput: 228-364 req/s ở load test, 1,317 req/s ở stress test

6.4.2 Tổng quan Security Testing

- **19/19 test cases PASSED**
- Zero vulnerabilities detected
- SQL Injection, XSS, CSRF: **All blocked**
- Authentication: JWT + BCrypt hashing
- Input validation: Frontend + Backend dual validation

6.4.3 Đánh giá và Kết luận

Performance Testing: Hệ thống FloginFE_BE đã chứng minh khả năng xử lý tốt ở mức tải 2,000 concurrent users với 0% error rate. Breaking point ở 11,000 users cho thấy hệ thống cần được tối ưu hóa để phục vụ quy mô lớn hơn.

Security Testing: Hệ thống đạt chuẩn bảo mật cao với 100% các test cases về SQL Injection, XSS, CSRF, Authentication đều PASSED. Không phát hiện lỗ hổng bảo mật nào.

6.4.4 Khuyến nghị cải thiện

Dựa trên kết quả Stress Test (breaking point ở 11,000 users với 15.44% error rate), các khuyến nghị cải thiện:

Performance:

- **Database Connection Pool**: Tăng từ 10 → 100 connections (HikariCP)
- **Thread Pool**: Tăng từ 200 → 1000 threads (Tomcat)
- **Caching Layer**: Implement Redis cho Product API (giảm 80% database queries)
- **Database Optimization**:
  - Indexing: product_name, category_id, price
  - Query optimization: Lazy loading images, pagination
  - Read replicas cho Product GET operations
- **Horizontal Scaling**: Deploy 3-5 instances với Nginx load balancer
- **CDN Integration**: CloudFlare cho static content (images, CSS, JS)
- **Rate Limiting**: 2000 req/s global, 100 req/s per user (prevent DoS)
- **Circuit Breaker**: Resilience4j để prevent cascading failures
- **Monitoring**: Prometheus + Grafana + AlertManager cho real-time metrics
- **JVM Tuning**: Heap size 4GB → 8GB, G1GC configuration

Kết quả mong đợi sau optimization:

- **Safe Capacity**: 2,000 → 10,000 concurrent users
- **Breaking point**: 11,000 → 30,000+ users
- **Error rate**: 15.44% → $< 1\%$ ở 15,000 VUs
- **Response time p(95)**: 7,944ms → $<$ 100ms ngay cả với 15,000 VUs
- **Throughput**: 1,317 → 10,000+ req/s

6.4.5 Hướng phát triển tiếp theo

Giám sát và Phân tích:

- Tích hợp Prometheus và Grafana cho real-time monitoring
- Thiết lập alerting với AlertManager cho performance và security incidents
- Application Performance Monitoring (APM) với New Relic hoặc Datadog

Tự động hóa Testing:

- Tích hợp Performance Tests vào CI/CD pipeline
- Chạy tự động Security Tests trước mỗi deployment
- Thiết lập performance budgets và fail builds nếu không đạt

Tối ưu hóa tiếp theo:

- Microservices architecture cho scalability tốt hơn
- Message queue (RabbitMQ/Kafka) cho async operations
- Container orchestration với Kubernetes cho auto-scaling

KẾT LUẬN

Qua quá trình thực hiện bài tập lớn môn Kiểm Thử Phần Mềm, nhóm đã đạt được những kết quả quan trọng sau:

Kết quả đạt được

1. **Nắm vững quy trình kiểm thử**: Nhóm đã thực hành đầy đủ các loại kiểm thử từ Unit Testing, Integration Testing, Mock Testing đến Automation Testing và CI/CD.

2. **Áp dụng TDD thành công**:
   - Frontend: Đạt 98.14% code coverage cho validation modules
   - Backend: Đạt 95-100% coverage cho các Service layers

3. **Performance Testing xuất sắc**:
   - Xử lý được 1000+ concurrent users với 0% error rate
   - Response time trung bình $<$ 10ms cho cả Login và Product APIs
   - Xác định được breaking point tại 2000-2500 concurrent users

4. **Security Testing toàn diện**:
   - 19/19 test cases passed (100% success rate)
   - Zero vulnerabilities detected
   - Đảm bảo an toàn trước SQL Injection, XSS, CSRF

5. **CI/CD Integration**: Thiết lập thành công pipeline tự động hóa testing

Kỹ năng đạt được

Thông qua bài tập này, các thành viên trong nhóm đã:

- Nắm vững các framework testing hiện đại (Jest, JUnit, Mockito, Cypress/k6)
- Hiểu rõ quy trình TDD và lợi ích của nó
- Biết cách đo lường và cải thiện code coverage
- Có khả năng thiết kế test cases chi tiết và toàn diện
- Thực hành Performance Testing và Security Testing
- Tích hợp testing vào CI/CD pipeline

Hạn chế và hướng phát triển

Hạn chế:

- Breaking point còn thấp (2000-2500 users), cần optimization
- Chưa thực hiện Penetration Testing chuyên sâu
- Chưa có APM (Application Performance Monitoring) đầy đủ

Hướng phát triển:

- Tối ưu database connection pool và thread pool
- Implement caching layer (Redis)
- Thêm Load Balancer và Horizontal Scaling
- Bổ sung Monitoring với Prometheus + Grafana
- Thực hiện regular security audits

Lời kết

Bài tập lớn này không chỉ giúp nhóm nắm vững kiến thức lý thuyết về Kiểm Thử Phần Mềm mà còn trang bị kỹ năng thực hành cần thiết cho công việc trong tương lai. Nhóm cam kết sẽ tiếp tục áp dụng các kiến thức này vào các dự án thực tế và không ngừng học hỏi để nâng cao chất lượng phần mềm.

Một lần nữa, nhóm xin chân thành cảm ơn thầy Từ Lãng Phiêu đã tận tình hướng dẫn!

TÀI LIỆU THAM KHẢO

1. React Documentation, https://react.dev, Testing Library Documentation
2. Spring Boot Documentation, https://spring.io/projects/spring-boot, Spring Testing Guide
3. Jest Documentation, https://jestjs.io/, JavaScript Testing Framework
4. JUnit 5 User Guide, https://junit.org/junit5/, Testing Framework for Java
5. Mockito Framework, https://site.mockito.org/, Mocking Framework for Java
6. Cypress Documentation, https://www.cypress.io/, End-to-End Testing Framework
7. Grafana k6 Documentation, https://k6.io/docs/, Performance Testing Tool
8. Test-Driven Development: By Example, Kent Beck, Addison-Wesley Professional
9. The Art of Software Testing, Glenford J. Myers, Wiley Publishing
10. OWASP Testing Guide, https://owasp.org/www-project-web-security-testing-guide/