TRƯỜNG ĐẠI HỌC SÀI GÒN
KHOA CÔNG NGHỆ THÔNG TIN

CÔNG NGHỆ PHẦN MỀM

Bài Tập Lớn - Kiểm Thử Phần Mềm

Ứng dụng Đăng nhập & Quản lý Sản phẩm

(Version 1.0)

GVHD: Từ Lãng Phiêu

TP. HỒ CHÍ MINH, THÁNG 10/2025

Mục lục

1 Giới thiệu về Dự án

1.1 Tổng quan

Dự án **FloginFE_BE** là một ứng dụng web hoàn chỉnh bao gồm:

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

1.2 Công nghệ sử dụng

1.2.1 Frontend

- React 18+ - Framework JavaScript
- React Testing Library - Testing cho React
- Jest - Testing framework
- Axios - HTTP client
- CSS3 - Styling với animations

1.2.2 Backend

- Spring Boot 3.2+ - Framework Java
- Java 17+
- JUnit 5 - Testing framework
- Mockito - Mock framework
- Maven - Build tool
- Spring Data JPA - Database operations

1.3 Cấu trúc dự án

FloginFE_BE/

- **frontend/** - React Application
  - src/
    - components/ - Login, Product components
    - services/ - API services
    - utils/ - Validation utilities
    - tests/ - Test files

  - package.json

- **backend/** - Spring Boot API
  - src/
    - main/java/com/flogin/
      - controller/ - AuthController, ProductController
      - service/ - Business logic
      - dto/ - Data Transfer Objects
      - entity/ - Database entities
      - repository/ - Data access
    - test/java/ - Test files
  - pom.xml

2 Câu 1: Phân tích và Thiết kế Test Cases (20 điểm)

2.1 Câu 1.1: Login - Phân tích và Test Scenarios (10 điểm)

2.1.1 Yêu cầu (5 điểm)

Dựa trên chức năng đăng nhập (Login), hãy thực hiện:

a) Phân tích đầy đủ các yêu cầu chức năng của tính năng Login (2 điểm)

- Validation rules cho username
- Validation rules cho password
- Authentication flow
- Error handling

b) Liệt kê và mô tả ít nhất 10 test scenarios cho Login bao gồm (2 điểm):

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

c) Phân loại test scenarios theo mức độ ưu tiên (Critical, High, Medium, Low) và giải thích (1 điểm)

Validation Rules cho Login:

- Username: 3-50 ký tự, chỉ chứa a-z, A-Z, 0-9, -, ., _
- Password: 6-100 ký tự, phải có cả chữ và số

2.1.2 Thiết kế Test Cases chi tiết (5 điểm)

Thiết kế 5 test cases quan trọng nhất cho Login theo template:
Sinh viên cần thiết kế thêm 4 test cases tương tự

Test Case ID	TC_LOGIN_001
Test Name	Đăng nhập thành công với credentials hợp lệ
Priority	Critical
Preconditions	- User account exists - Application is running
Test Steps	1. Navigate to login page 2. Enter valid username 3. Enter valid password 4. Click Login button
Test Data	Username: testuser Password: Test123
Expected Result	- Success message displayed - Token stored - Redirect to dashboard
Actual Result	(Để trống)
Status	Not Run

Bảng 1: Template Test Case cho Login

2.2 Câu 1.2: Product - Phân tích và Test Scenarios (10 điểm)

2.2.1 Yêu cầu (5 điểm)

Dựa trên chức năng quản lý sản phẩm (Product Management), hãy thực hiện:

a) Phân tích đầy đủ các yêu cầu chức năng của Product CRUD (2 điểm)

- Create: Thêm sản phẩm mới
- Read: Xem danh sách/chi tiết sản phẩm
- Update: Cập nhật thông tin sản phẩm
- Delete: Xóa sản phẩm

b) Liệt kê và mô tả ít nhất 10 test scenarios cho Product bao gồm (2 điểm):

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

c) Phân loại test scenarios theo mức độ ưu tiên và giải thích (1 điểm)

Validation Rules cho Product:

- Product Name: 3-100 ký tự, không được rỗng
- Price: > 0, $<= 999{,}999{,}999$
- Quantity: $>= 0$, $<= 99{,}999$
- Description: $<= 500$ ký tự
- Category: Phải thuộc danh sách categories có sẵn

2.2.2 Thiết kế Test Cases chi tiết (5 điểm)

Thiết kế 5 test cases quan trọng nhất cho Product Management:

Test Case ID	TC_PRODUCT_001
Test Name	Tạo sản phẩm mới thành công
Priority	Critical
Preconditions	- User đã đăng nhập - User có quyền tạo sản phẩm
Test Steps	1. Navigate to Product page 2. Click "Add New Product" 3. Enter product information 4. Click Save button
Test Data	Name: Laptop Dell Price: 15000000 Quantity: 10 Category: Electronics
Expected Result	- Product created successfully - Success message displayed - Product appears in list
Actual Result	(Để trống)
Status	Not Run

Bảng 2: Template Test Case cho Product

Sinh viên cần thiết kế thêm 4 test cases tương tự cho Update, Delete, Read operations

3 Câu 2: Unit Testing và Test-Driven Development (20 điểm)

3.1 Câu 2.1: Login - Unit Tests Frontend và Backend (10 điểm)

3.1.1 Frontend Unit Tests - Validation Login (5 điểm)

Áp dụng TDD để phát triển unit tests cho validation module của Login:

Yêu cầu:

a) Viết unit tests cho `validateUsername()` (2 điểm):

- Test username rỗng
- Test username quá ngắn/dài
- Test ký tự đặc biệt không hợp lệ
- Test username hợp lệ

b) Viết unit tests cho `validatePassword()` (2 điểm):

- Test password rỗng
- Test password quá ngắn/dài
- Test password không có chữ hoặc số
- Test password hợp lệ

c) Coverage >= 90% cho validation module (1 điểm)

Ví dụ Frontend Unit Test:

```
import { validateUsername, validatePassword } from './validation';

describe('Login Validation Tests', () => {
  test('TC1: Username rong - nen tra ve loi', () => {
    expect(validateUsername('')).toBe(
      'Ten dang nhap khong duoc de trong'
    );
  });

  test('TC2: Username qua ngan - nen tra ve loi', () => {
    expect(validateUsername('ab')).toBe(
      'Ten dang nhap phai co it nhat 3 ky tu'
    );
  });

  test('Username hop le - khong co loi', () => {
    expect(validateUsername('user123')).toBe('');
  });

  // Sinh vien can viet them cac test cases khac
});
```

Listing 1: validation.test.js

3.1.2 Backend Unit Tests - Login Service (5 điểm)

Viết unit tests cho `AuthService` của backend:

Yêu cầu:

a) Test method `authenticate()` với các scenarios (3 điểm):

- Login thành công
- Login với username không tồn tại
- Login với password sai
- Validation errors

b) Test validation methods riêng lẻ (1 điểm)

c) Coverage >= 85% cho AuthService (1 điểm)

Ví dụ Backend Unit Test:

```
@DisplayName("Login Service Unit Tests")
class AuthServiceTest {

    private AuthService authService;

    @BeforeEach
    void setUp() {
        authService = new AuthService();
    }

    @Test
    @DisplayName("TC1: Login thanh cong voi credentials hop le")
    void testLoginSuccess() {
        LoginRequest request = new LoginRequest(
            "testuser", "Test123"
        );
        LoginResponse response = authService.authenticate(request);

        assertTrue(response.isSuccess());
        assertEquals("Dang nhap thanh cong",
                     response.getMessage());
        assertNotNull(response.getToken());
    }

    @Test
    @DisplayName("TC2: Login that bai voi username sai")
    void testLoginFailure() {
        LoginRequest request = new LoginRequest(
            "wronguser", "Pass123"
        );
        LoginResponse response = authService.authenticate(request);

        assertFalse(response.isSuccess());
    }

    // Sinh vien can viet them cac test cases khac
}
```

Listing 2: AuthServiceTest.java

3.2 Câu 2.2: Product - Unit Tests Frontend và Backend (10 điểm)

3.2.1 Frontend Unit Tests - Product Validation (5 điểm)

Áp dụng TDD cho validation của Product:

Yêu cầu:

a) Viết unit tests cho `validateProduct()` (3 điểm):

- Test product name validation
- Test price validation (boundary tests)
- Test quantity validation
- Test description length
- Test category validation

b) Viết tests cho Product form component (1 điểm)

c) Coverage >= 90% (1 điểm)

Ví dụ Product Validation Test:

```
import { validateProduct } from './productValidation';

describe('Product Validation Tests', () => {
  test('TC1: Product name rong - nen tra ve loi', () => {
    const product = {
      name: '',
      price: 1000,
      quantity: 10
    };
    const errors = validateProduct(product);

    expect(errors.name).toBe('Ten san pham khong duoc de trong');
  });

  test('TC2: Price am - nen tra ve loi', () => {
    const product = {
      name: 'Laptop',
      price: -1000,
      quantity: 10
    };
    const errors = validateProduct(product);

    expect(errors.price).toBe('Gia san pham phai lon hon 0');
  });

  test('TC3: Product hop le - khong co loi', () => {
    const product = {
      name: 'Laptop Dell',
      price: 15000000,
      quantity: 10,
      category: 'Electronics'
    };
    const errors = validateProduct(product);

    expect(Object.keys(errors).length).toBe(0);
  });

  // Sinh vien can viet them tests cho boundary values
```

```
});
```

Listing 3: productValidation.test.js

3.2.2 Backend Unit Tests - Product Service (5 điểm)

Viết unit tests cho `ProductService`:

Yêu cầu:

a) Test CRUD operations (4 điểm):

- Test createProduct()
- Test getProduct()
- Test updateProduct()
- Test deleteProduct()
- Test getAll() với pagination

b) Coverage >= 85% cho ProductService (1 điểm)

Ví dụ Product Service Test:

```
@DisplayName("Product Service Unit Tests")
class ProductServiceTest {

    @Mock
    private ProductRepository productRepository;

    @InjectMocks
    private ProductService productService;

    @BeforeEach
    void setUp() {
        MockitoAnnotations.openMocks(this);
    }

    @Test
    @DisplayName("TC1: Tao san pham moi thanh cong")
    void testCreateProduct() {
        ProductDto productDto = new ProductDto(
            "Laptop", 15000000, 10, "Electronics"
        );
        Product product = new Product(
            1L, "Laptop", 15000000, 10, "Electronics"
        );

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

        ProductDto result = productService.createProduct(productDto);

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

    @Test
    @DisplayName("TC2: Cap nhat san pham thanh cong")
    void testUpdateProduct() {
```

```
38          // Implementation here
39      }
40
41      // Sinh vien can viet them cac test cases khac
42  }
```

Listing 4: ProductServiceTest.java

4 Câu 3: Integration Testing (20 điểm)

4.1 Câu 3.1: Login - Integration Testing (10 điểm)

4.1.1 Frontend Component Integration (5 điểm)

Test tích hợp Login component với API service:

Yêu cầu:

a) Test rendering và user interactions (2 điểm)

b) Test form submission và API calls (2 điểm)

c) Test error handling và success messages (1 điểm)

Ví dụ Component Integration Test:

```
import { render, screen, fireEvent, waitFor } from '@testing-library/react';
import Login from './Login';

describe('Login Component Integration Tests', () => {
  test('Hien thi loi khi submit form rong', async () => {
    render(<Login />);

    const submitButton = screen.getByTestId('login-button');
    fireEvent.click(submitButton);

    await waitFor(() => {
      expect(screen.getByTestId('username-error'))
        .toBeInTheDocument();
    });
  });

  test('Goi API khi submit form hop le', async () => {
    render(<Login />);

    const usernameInput = screen.getByTestId('username-input');
    const passwordInput = screen.getByTestId('password-input');
    const submitButton = screen.getByTestId('login-button');

    fireEvent.change(usernameInput, {
      target: { value: 'testuser' }
    });
    fireEvent.change(passwordInput, {
      target: { value: 'Test123' }
    });
    fireEvent.click(submitButton);

    await waitFor(() => {
      expect(screen.getByTestId('login-message'))
        .toHaveTextContent('thanh cong');
    });
  });
});
```

Listing 5: Login.integration.test.js

4.1.2 Backend API Integration (5 điểm)

Test API endpoints của Login với MockMvc:

Yêu cầu:

a) Test POST /api/auth/login endpoint (3 điểm)

b) Test response structure và status codes (1 điểm)

c) Test CORS và headers (1 điểm)

Ví dụ API Integration Test:

```java
@WebMvcTest(AuthController.class)
@DisplayName("Login API Integration Tests")
class AuthControllerIntegrationTest {

    @Autowired
    private MockMvc mockMvc;

    @Autowired
    private ObjectMapper objectMapper;

    @MockBean
    private AuthService authService;

    @Test
    @DisplayName("POST /api/auth/login - Thanh cong")
    void testLoginSuccess() throws Exception {
        LoginRequest request = new LoginRequest(
            "testuser", "Test123"
        );
        LoginResponse mockResponse = new LoginResponse(
            true, "Dang nhap thanh cong", "token123",
            new UserDto("testuser", "testuser@example.com")
        );

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

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

Listing 6: AuthControllerIntegrationTest.java

4.2 Câu 3.2: Product - Integration Testing (10 điểm)

4.2.1 Frontend Component Integration (5 điểm)

Test tích hợp Product components:

Yêu cầu:

a) Test ProductList component với API (2 điểm)

b) Test ProductForm component (create/edit) (2 điểm)

c) Test ProductDetail component (1 điểm)

Ví dụ Product Component Test:

```
import { render, screen, fireEvent, waitFor } from '@testing-library/react';
import ProductForm from './ProductForm';

describe('Product Form Integration Tests', () => {
  test('Tao san pham moi thanh cong', async () => {
    render(<ProductForm />);

    fireEvent.change(screen.getByLabelText('Ten san pham'), {
      target: { value: 'Laptop Dell' }
    });
    fireEvent.change(screen.getByLabelText('Gia'), {
      target: { value: '15000000' }
    });
    fireEvent.change(screen.getByLabelText('So luong'), {
      target: { value: '10' }
    });

    fireEvent.click(screen.getByText('Luu'));

    await waitFor(() => {
      expect(screen.getByText('Them san pham thanh cong'))
        .toBeInTheDocument();
    });
  });
});
```

Listing 7: ProductForm.integration.test.js

4.2.2 Backend API Integration (5 điểm)

Test các API endpoints của Product:

Yêu cầu:

a) Test POST /api/products (Create) (1 điểm)

b) Test GET /api/products (Read all) (1 điểm)

c) Test GET /api/products/{id} (Read one) (1 điểm)

d) Test PUT /api/products/{id} (Update) (1 điểm)

e) Test DELETE /api/products/{id} (Delete) (1 điểm)

Ví dụ Product API Test:

```java
@WebMvcTest(ProductController.class)
@DisplayName("Product API Integration Tests")
class ProductControllerIntegrationTest {

    @Autowired
    private MockMvc mockMvc;

    @MockBean
    private ProductService productService;

    @Test
    @DisplayName("GET /api/products - Lay danh sach san pham")
    void testGetAllProducts() throws Exception {
        List<ProductDto> products = Arrays.asList(
            new ProductDto("Laptop", 15000000, 10),
            new ProductDto("Mouse", 200000, 50)
        );

        when(productService.getAllProducts(any()))
            .thenReturn(products);

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

    @Test
    @DisplayName("POST /api/products - Tao san pham moi")
    void testCreateProduct() throws Exception {
        // Implementation here
    }
}
```

Listing 8: ProductControllerIntegrationTest.java

5 Câu 4: Mock Testing (10 điểm)

5.1 Câu 4.1: Login - Mock Testing (5 điểm)

5.1.1 Frontend Mocking (2.5 điểm)

Mock external dependencies cho Login component:

Yêu cầu:

a) Mock authService.loginUser() (1 điểm)

b) Test với mocked successful/failed responses (1 điểm)

c) Verify mock calls (0.5 điểm)

Ví dụ Frontend Mock Test:

```
import { render, screen, fireEvent, waitFor } from '@testing-library/react';
import Login from './Login';
import * as authService from '../services/authService';

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

describe('Login Mock Tests', () => {
  beforeEach(() => {
    jest.clearAllMocks();
  });

  test('Mock: Login thanh cong', async () => {
    authService.loginUser.mockResolvedValue({
      success: true,
      token: 'mock-token-123',
      user: { username: 'testuser' }
    });

    render(<Login />);

    fireEvent.change(screen.getByTestId('username-input'), {
      target: { value: 'testuser' }
    });
    fireEvent.change(screen.getByTestId('password-input'), {
      target: { value: 'Test123' }
    });
    fireEvent.click(screen.getByTestId('login-button'));

    await waitFor(() => {
      expect(authService.loginUser).toHaveBeenCalledWith(
        'testuser', 'Test123'
      );
      expect(screen.getByText(/thanh cong/i)).toBeInTheDocument();
    });
  });

  test('Mock: Login that bai', async () => {
    authService.loginUser.mockRejectedValue({
      message: 'Invalid credentials'
    });

    // Implementation here
  });
```

```
});
```

Listing 9: Login.mock.test.js

5.1.2 Backend Mocking (2.5 điểm)

Mock dependencies trong Backend tests:

Yêu cầu:

a) Mock AuthService với @MockBean (1 điểm)

b) Test controller với mocked service (1 điểm)

c) Verify mock interactions (0.5 điểm)

Ví dụ Backend Mock Test:

```
@WebMvcTest(AuthController.class)
class AuthControllerMockTest {

    @Autowired
    private MockMvc mockMvc;

    @MockBean
    private AuthService authService;

    @Test
    @DisplayName("Mock: Controller voi mocked service success")
    void testLoginWithMockedService() throws Exception {
        LoginResponse mockResponse = new LoginResponse(
            true, "Success", "mock-token"
        );

        when(authService.authenticate(any()))
            .thenReturn(mockResponse);

        mockMvc.perform(post("/api/auth/login")
                .contentType(MediaType.APPLICATION_JSON)
                .content("{\"username\":\"test\",\"password\":\"Pass123\"}"))
                .andExpect(status().isOk());

        verify(authService, times(1)).authenticate(any());
    }
}
```

Listing 10: AuthControllerMockTest.java

5.2 Câu 4.2: Product - Mock Testing (5 điểm)

5.2.1 Frontend Mocking (2.5 điểm)

Mock ProductService trong component tests:

Yêu cầu:

a) Mock CRUD operations (1.5 điểm)

b) Test success và failure scenarios (0.5 điểm)

c) Verify all mock calls (0.5 điểm)

Ví dụ Product Mock Test:

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

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

describe('Product Mock Tests', () => {
  test('Mock: Create product thanh cong', async () => {
    const mockProduct = {
      id: 1,
      name: 'Laptop',
      price: 15000000
    };

    productService.createProduct.mockResolvedValue(mockProduct);

    // Test implementation

    expect(productService.createProduct).toHaveBeenCalledTimes(1);
  });

  test('Mock: Get products with pagination', async () => {
    const mockProducts = {
      data: [/* products */],
      page: 1,
      total: 100
    };

    productService.getProducts.mockResolvedValue(mockProducts);

    // Test implementation
  });
});
```

Listing 11: Product.mock.test.js

5.2.2 Backend Mocking (2.5 điểm)

Mock ProductRepository trong service tests:

Yêu cầu:

a) Mock ProductRepository (1 điểm)

b) Test service layer với mocked repository (1 điểm)

c) Verify repository interactions (0.5 điểm)

Ví dụ Repository Mock Test:

```java
class ProductServiceMockTest {

    @Mock
    private ProductRepository productRepository;

    @InjectMocks
    private ProductService productService;

    @Test
    void testGetProductById() {
        Product mockProduct = new Product(
            1L, "Laptop", 15000000
        );

        when(productRepository.findById(1L))
            .thenReturn(Optional.of(mockProduct));

        ProductDto result = productService.getProductById(1L);

        assertNotNull(result);
        assertEquals("Laptop", result.getName());

        verify(productRepository).findById(1L);
    }
}
```

Listing 12: ProductServiceMockTest.java

6 Câu 5: Automation Testing và CI/CD (10 điểm)

6.1 Câu 5.1: Login - E2E Automation Testing (5 điểm)

6.1.1 Setup và Configuration (1 điểm)

Yêu cầu:

- Cài đặt Cypress hoặc Selenium
- Cấu hình test environment
- Setup Page Object Model

6.1.2 E2E Test Scenarios cho Login (2.5 điểm)

Viết automated tests cho toàn bộ login flow:

Yêu cầu:

a) Test complete login flow (1 điểm)

b) Test validation messages (0.5 điểm)

c) Test success/error flows (0.5 điểm)

d) Test UI elements interactions (0.5 điểm)

Ví dụ Cypress E2E Test:

```
describe('Login E2E Tests', () => {
  beforeEach(() => {
    cy.visit('http://localhost:3000');
  });

  it('Nen hien thi form login', () => {
    cy.get('[data-testid="username-input"]').should('be.visible');
    cy.get('[data-testid="password-input"]').should('be.visible');
    cy.get('[data-testid="login-button"]').should('be.visible');
  });

  it('Nen login thanh cong voi credentials hop le', () => {
    cy.get('[data-testid="username-input"]').type('testuser');
    cy.get('[data-testid="password-input"]').type('Test123');
    cy.get('[data-testid="login-button"]').click();

    cy.get('[data-testid="login-message"]')
      .should('contain', 'thanh cong');
    cy.url().should('include', '/dashboard');
  });

  it('Nen hien thi loi voi credentials khong hop le', () => {
    cy.get('[data-testid="username-input"]').type('ab');
    cy.get('[data-testid="password-input"]').type('123');
    cy.get('[data-testid="login-button"]').click();

    cy.get('[data-testid="username-error"]')
      .should('be.visible');
  });
});
```

Listing 13: login.e2e.spec.js

6.1.3 CI/CD Integration cho Login Tests (1.5 điểm)

Yêu cầu:

- Tạo GitHub Actions workflow
- Run login tests automatically
- Generate test reports

Ví dụ CI/CD Config:

```yaml
name: Login Tests CI

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

jobs:
  test-login:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v2

      - name: Setup Node.js
        uses: actions/setup-node@v2
        with:
          node-version: '18'

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

      - name: Run Login Unit Tests
        run: |
          cd frontend
          npm test -- --testPathPattern=Login

      - name: Run Login E2E Tests
        run: |
          cd frontend
          npm run test:e2e -- --spec "cypress/e2e/login.spec.js"
```

Listing 14: .github/workflows/login-tests.yml

6.2 Câu 5.2: Product - E2E Automation Testing (5 điểm)

6.2.1 Setup Page Object Model (1 điểm)

Implement POM cho Product pages:

Ví dụ Page Object:

```
class ProductPage {
  visit() {
    cy.visit('/products');
  }

  clickAddNew() {
    cy.get('[data-testid="add-product-btn"]').click();
  }

  fillProductForm(product) {
    cy.get('[data-testid="product-name"]').type(product.name);
    cy.get('[data-testid="product-price"]').type(product.price);
    cy.get('[data-testid="product-quantity"]').type(product.quantity);
  }

  submitForm() {
    cy.get('[data-testid="submit-btn"]').click();
  }

  getSuccessMessage() {
    return cy.get('[data-testid="success-message"]');
  }

  getProductInList(name) {
    return cy.contains('[data-testid="product-item"]', name);
  }
}

export default ProductPage;
```

Listing 15: ProductPage.js

6.2.2 E2E Test Scenarios cho Product (2.5 điểm)

Viết automated tests cho CRUD operations:

Yêu cầu:

a) Test Create product flow (0.5 điểm)

b) Test Read/List products (0.5 điểm)

c) Test Update product (0.5 điểm)

d) Test Delete product (0.5 điểm)

e) Test Search/Filter functionality (0.5 điểm)

Ví dụ Product E2E Test:

```
import ProductPage from '../pages/ProductPage';

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

```
  beforeEach(() => {
    cy.login('testuser', 'Test123'); // Custom command
    productPage.visit();
  });

  it('Nen tao san pham moi thanh cong', () => {
    productPage.clickAddNew();
    productPage.fillProductForm({
      name: 'Laptop Dell',
      price: '15000000',
      quantity: '10'
    });
    productPage.submitForm();

    productPage.getSuccessMessage()
      .should('contain', 'thanh cong');
    productPage.getProductInList('Laptop Dell')
      .should('exist');
  });

  it('Nen cap nhat san pham thanh cong', () => {
    productPage.getProductInList('Laptop Dell').click();
    cy.get('[data-testid="edit-btn"]').click();
    cy.get('[data-testid="product-price"]').clear().type('14000000');
    productPage.submitForm();

    cy.get('[data-testid="product-price"]')
      .should('contain', '14,000,000');
  });

  it('Nen xoa san pham thanh cong', () => {
    productPage.getProductInList('Laptop Dell').click();
    cy.get('[data-testid="delete-btn"]').click();
    cy.get('[data-testid="confirm-delete"]').click();

    productPage.getProductInList('Laptop Dell')
      .should('not.exist');
  });
});
```

Listing 16: product.e2e.spec.js

6.2.3 CI/CD Integration (1.5 điểm)

Setup complete CI/CD pipeline:

Ví dụ Complete CI/CD:

```
name: Complete CI/CD Pipeline

on:
  push:
    branches: [ main ]

jobs:
  test:
    runs-on: ubuntu-latest

    services:
      postgres:
        image: postgres:14
```

```
      env:
        POSTGRES_DB: testdb
        POSTGRES_PASSWORD: password
      ports:
        - 5432:5432

  steps:
    - uses: actions/checkout@v2

    - name: Setup Java
      uses: actions/setup-java@v2
      with:
        java-version: '17'

    - name: Setup Node.js
      uses: actions/setup-node@v2
      with:
        node-version: '18'

    - name: Backend Tests
      run: |
        cd backend
        ./mvnw clean test

    - name: Frontend Tests
      run: |
        cd frontend
        npm install
        npm test -- --coverage

    - name: E2E Tests
      run: |
        cd frontend
        npm run test:e2e

    - name: Upload Coverage
      uses: codecov/codecov-action@v2
      with:
        files: ./frontend/coverage/lcov.info,./backend/target/site/jacoco/jacoco
  .xml
```

Listing 17: .github/workflows/ci.yml

7 Phần Mở Rộng (Bonus 20 điểm)

7.1 Performance Testing (10 điểm)

7.1.1 Yêu cầu

a) Setup JMeter hoặc k6 cho performance testing (2 điểm)

b) Viết performance tests cho Login API (3 điểm):

- Load test: 100, 500, 1000 concurrent users
- Stress test: Tìm breaking point
- Response time analysis

c) Viết performance tests cho Product API (3 điểm)

d) Phân tích kết quả và đưa ra recommendations (2 điểm)

7.2 Security Testing (10 điểm)

7.2.1 Yêu cầu

a) Test common vulnerabilities (5 điểm):

- SQL Injection
- XSS (Cross-Site Scripting)
- CSRF (Cross-Site Request Forgery)
- Authentication bypass attempts

b) Test input validation và sanitization (3 điểm)

c) Security best practices implementation (2 điểm):

- Password hashing
- HTTPS enforcement
- CORS configuration
- Security headers

8 Tiêu chí Đánh giá

8.1 Bảng Phân bổ Điểm

Nội dung	Login	Product	Tổng
Câu 1: Phân tích và Test Cases	10	10	20
Câu 2: Unit Testing và TDD	10	10	20
Câu 3: Integration Testing	10	10	20
Câu 4: Mock Testing	5	5	10
Câu 5: Automation và CI/CD	5	5	10
Tổng điểm bắt buộc	**40**	**40**	**80**
Phần Mở Rộng (Bonus)			+20
Điểm tối đa			**100**

Bảng 3: Bảng phân bổ điểm chi tiết

8.2 Tiêu chí Chất lượng

8.2.1 Code Quality (30%)

- Clean code principles
- Proper test structure (AAA pattern)
- Meaningful test names
- Test coverage >= 80%
- All tests pass

8.2.2 Documentation (20%)

- Test cases đầy đủ và rõ ràng
- Screenshots và evidences
- Test reports
- README với hướng dẫn chạy tests

8.2.3 Completeness (30%)

- Hoàn thành tất cả yêu cầu bắt buộc
- Đầy đủ test cases cho cả Login và Product
- CI/CD pipeline hoạt động

8.2.4 Best Practices (20%)

- Áp dụng TDD đúng cách
- Proper mocking strategy
- Good test data management
- Automation best practices

9 Hướng dẫn Nộp bài

9.1 Format Nộp bài

9.1.1 Source Code

- Git repository (GitHub/GitLab) - Public hoặc add giảng viên
- Commit history rõ ràng
- README.md đầy đủ
- Có .gitignore phù hợp

9.1.2 Báo cáo

- File PDF (tối đa 50 trang)
- Bao gồm:
  - Giới thiệu project
  - Test cases chi tiết (Login + Product)
  - Test execution results
  - Screenshots và evidences
  - Coverage reports
  - CI/CD pipeline documentation
  - Kết luận

9.2 Thời hạn

- Deadline: [11/11/2025]
- Không chấp nhận nộp trễ

9.3 Demo (Optional)

- Video demo 10-15 phút
- Demo chạy tests
- Giải thích CI/CD pipeline
- Q&A với giảng viên

Tài liệu

[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] Test-Driven Development: By Example, *Kent Beck*, Addison-Wesley Professional