

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 Lăng Phiêu

TP. HỒ CHÍ MINH, THÁNG 10/2025

Mục lục

1	Giới thiệu về Dự án	1
1.1	Tổng quan	1
1.2	Công nghệ sử dụng	1
1.2.1	Frontend	1
1.2.2	Backend	1
1.3	Cấu trúc dự án	1
2	Câu 1: Phân tích và Thiết kế Test Cases (20 điểm)	3
2.1	Câu 1.1: Login - Phân tích và Test Scenarios (10 điểm)	3
2.1.1	Yêu cầu (5 điểm)	3
2.1.2	Thiết kế Test Cases chi tiết (5 điểm)	3
2.2	Câu 1.2: Product - Phân tích và Test Scenarios (10 điểm)	4
2.2.1	Yêu cầu (5 điểm)	4
2.2.2	Thiết kế Test Cases chi tiết (5 điểm)	5
3	Câu 2: Unit Testing và Test-Driven Development (20 điểm)	6
3.1	Câu 2.1: Login - Unit Tests Frontend và Backend (10 điểm)	6
3.1.1	Frontend Unit Tests - Validation Login (5 điểm)	6
3.1.2	Backend Unit Tests - Login Service (5 điểm)	7
3.2	Câu 2.2: Product - Unit Tests Frontend và Backend (10 điểm)	8
3.2.1	Frontend Unit Tests - Product Validation (5 điểm)	8
3.2.2	Backend Unit Tests - Product Service (5 điểm)	9
4	Câu 3: Integration Testing (20 điểm)	11
4.1	Câu 3.1: Login - Integration Testing (10 điểm)	11
4.1.1	Frontend Component Integration (5 điểm)	11
4.1.2	Backend API Integration (5 điểm)	12
4.2	Câu 3.2: Product - Integration Testing (10 điểm)	13
4.2.1	Frontend Component Integration (5 điểm)	13
4.2.2	Backend API Integration (5 điểm)	13
5	Câu 4: Mock Testing (10 điểm)	15
5.1	Câu 4.1: Login - Mock Testing (5 điểm)	15
5.1.1	Frontend Mocking (2.5 điểm)	15
5.1.2	Backend Mocking (2.5 điểm)	16
5.2	Câu 4.2: Product - Mock Testing (5 điểm)	17
5.2.1	Frontend Mocking (2.5 điểm)	17
5.2.2	Backend Mocking (2.5 điểm)	17
6	Câu 5: Automation Testing và CI/CD (10 điểm)	19
6.1	Câu 5.1: Login - E2E Automation Testing (5 điểm)	19
6.1.1	Setup và Configuration (1 điểm)	19
6.1.2	E2E Test Scenarios cho Login (2.5 điểm)	19
6.1.3	CI/CD Integration cho Login Tests (1.5 điểm)	20
6.2	Câu 5.2: Product - E2E Automation Testing (5 điểm)	21
6.2.1	Setup Page Object Model (1 điểm)	21
6.2.2	E2E Test Scenarios cho Product (2.5 điểm)	21
6.2.3	CI/CD Integration (1.5 điểm)	22



7	Phân Mở Rộng (Bonus 20 điểm)	24
7.1	Performance Testing (10 điểm)	24
7.1.1	Yêu cầu	24
7.2	Security Testing (10 điểm)	24
7.2.1	Yêu cầu	24
8	Tiêu chí Đánh giá	25
8.1	Bảng Phân bổ Điểm	25
8.2	Tiêu chí Chất lượng	25
8.2.1	Code Quality (30%)	25
8.2.2	Documentation (20%)	25
8.2.3	Completeness (30%)	25
8.2.4	Best Practices (20%)	26
9	Hướng dẫn Nộp bài	27
9.1	Format Nộp bài	27
9.1.1	Source Code	27
9.1.2	Báo cáo	27
9.2	Thời hạn	27
9.3	Demo (Optional)	27



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 , $\leq 999,999,999$
- Quantity: ≥ 0 , $\leq 99,999$
- 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 $\geq 90\%$ cho validation module (1 điểm)

Ví dụ Frontend Unit Test:

```
1 import { validateUsername, validatePassword } from './validation';
2
3 describe('Login Validation Tests', () => {
4   test('TC1: Username rong - nen tra ve loi', () => {
5     expect(validateUsername('')).toBe(
6       'Ten dang nhap khong duoc de trong'
7     );
8   });
9
10  test('TC2: Username qua ngan - nen tra ve loi', () => {
11    expect(validateUsername('ab')).toBe(
12      'Ten dang nhap phai co it nhat 3 ky tu'
13    );
14  });
15
16  test('Username hop le - khong co loi', () => {
17    expect(validateUsername('user123')).toBe('');
18  });
19
20  // Sinh vien can viet them cac test cases khac
21 });
```

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 $\geq 85\%$ cho AuthService (1 điểm)

Ví dụ Backend Unit Test:

```
1 @DisplayName("Login Service Unit Tests")
2 class AuthServiceTest {
3
4     private AuthService authService;
5
6     @BeforeEach
7     void setUp() {
8         authService = new AuthService();
9     }
10
11     @Test
12     @DisplayName("TC1: Login thanh cong voi credentials hop le")
13     void testLoginSuccess() {
14         LoginRequest request = new LoginRequest(
15             "testuser", "Test123"
16         );
17         LoginResponse response = authService.authenticate(request);
18
19         assertTrue(response.isSuccess());
20         assertEquals("Đăng nhập thành công",
21             response.getMessage());
22         assertNotNull(response.getToken());
23     }
24
25     @Test
26     @DisplayName("TC2: Login that bai voi username sai")
27     void testLoginFailure() {
28         LoginRequest request = new LoginRequest(
29             "wronguser", "Pass123"
30         );
31         LoginResponse response = authService.authenticate(request);
32
33         assertFalse(response.isSuccess());
34     }
35
36     // Sinh vien can viet them cac test cases khac
37 }
```

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 $\geq 90\%$ (1 điểm)

Ví dụ Product Validation Test:

```
1 import { validateProduct } from './productValidation';
2
3 describe('Product Validation Tests', () => {
4   test('TC1: Product name rong - nen tra ve loi', () => {
5     const product = {
6       name: '',
7       price: 1000,
8       quantity: 10
9     };
10    const errors = validateProduct(product);
11
12    expect(errors.name).toBe('Ten san pham khong duoc de trong');
13  });
14
15   test('TC2: Price am - nen tra ve loi', () => {
16     const product = {
17       name: 'Laptop',
18       price: -1000,
19       quantity: 10
20     };
21     const errors = validateProduct(product);
22
23     expect(errors.price).toBe('Gia san pham phai lon hon 0');
24   });
25
26   test('TC3: Product hop le - khong co loi', () => {
27     const product = {
28       name: 'Laptop Dell',
29       price: 15000000,
30       quantity: 10,
31       category: 'Electronics'
32     };
33     const errors = validateProduct(product);
34
35     expect(Object.keys(errors).length).toBe(0);
36   });
37
38   // Sinh vien can viet them tests cho boundary values
```



39 });

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 $\geq 85\%$ cho ProductService (1 điểm)

Ví dụ Product Service Test:

```
1 @DisplayName("Product Service Unit Tests")
2 class ProductServiceTest {
3
4     @Mock
5     private ProductRepository productRepository;
6
7     @InjectMocks
8     private ProductService productService;
9
10    @BeforeEach
11    void setUp() {
12        MockitoAnnotations.openMocks(this);
13    }
14
15    @Test
16    @DisplayName("TC1: Tao san pham moi thanh cong")
17    void testCreateProduct() {
18        ProductDto productDto = new ProductDto(
19            "Laptop", 15000000, 10, "Electronics"
20        );
21        Product product = new Product(
22            1L, "Laptop", 15000000, 10, "Electronics"
23        );
24
25        when(productRepository.save(any(Product.class)))
26            .thenReturn(product);
27
28        ProductDto result = productService.createProduct(productDto);
29
30        assertNotNull(result);
31        assertEquals("Laptop", result.getName());
32        verify(productRepository, times(1)).save(any(Product.class));
33    }
34
35    @Test
36    @DisplayName("TC2: Cap nhat san pham thanh cong")
37    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:

```
1 import { render, screen, fireEvent, waitFor } from '@testing-library/react';
2 import Login from './Login';
3
4 describe('Login Component Integration Tests', () => {
5   test('Hiện thị lỗi khi submit form rỗng', async () => {
6     render(<Login />);
7
8     const submitButton = screen.getByTestId('login-button');
9     fireEvent.click(submitButton);
10
11     await waitFor(() => {
12       expect(screen.getByTestId('username-error'))
13         .toBeInTheDocument();
14     });
15   });
16
17   test('Goi API khi submit form hợp lệ', async () => {
18     render(<Login />);
19
20     const usernameInput = screen.getByTestId('username-input');
21     const passwordInput = screen.getByTestId('password-input');
22     const submitButton = screen.getByTestId('login-button');
23
24     fireEvent.change(usernameInput, {
25       target: { value: 'testuser' }
26     });
27     fireEvent.change(passwordInput, {
28       target: { value: 'Test123' }
29     });
30     fireEvent.click(submitButton);
31
32     await waitFor(() => {
33       expect(screen.getByTestId('login-message'))
34         .toHaveTextContent('thanh cong');
35     });
36   });
37 });
```

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:

```
1 @WebMvcTest(AuthController.class)
2 @DisplayName("Login API Integration Tests")
3 class AuthControllerIntegrationTest {
4
5     @Autowired
6     private MockMvc mockMvc;
7
8     @Autowired
9     private ObjectMapper objectMapper;
10
11     @MockBean
12     private AuthService authService;
13
14     @Test
15     @DisplayName("POST /api/auth/login - Thành công")
16     void testLoginSuccess() throws Exception {
17         LoginRequest request = new LoginRequest(
18             "testuser", "Test123"
19         );
20         LoginResponse mockResponse = new LoginResponse(
21             true, "Đăng nhập thành công", "token123",
22             new UserDto("testuser", "testuser@example.com")
23         );
24
25         when(authService.authenticate(any(LoginRequest.class)))
26             .thenReturn(mockResponse);
27
28         mockMvc.perform(post("/api/auth/login")
29             .contentType(MediaType.APPLICATION_JSON)
30             .content(objectMapper.writeValueAsString(request)))
31             .andExpect(status().isOk())
32             .andExpect(jsonPath("$.success").value(true))
33             .andExpect(jsonPath("$.token").exists());
34     }
35 }
```

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:

```
1 import { render, screen, fireEvent, waitFor } from '@testing-library/react';
2 import ProductForm from './ProductForm';
3
4 describe('Product Form Integration Tests', () => {
5   test('Tạo sản phẩm mới thành công', async () => {
6     render(<ProductForm />);
7
8     fireEvent.change(screen.getByLabelText('Tên sản phẩm'), {
9       target: { value: 'Laptop Dell' }
10    });
11    fireEvent.change(screen.getByLabelText('Giá'), {
12      target: { value: '15000000' }
13    });
14    fireEvent.change(screen.getByLabelText('Số lượng'), {
15      target: { value: '10' }
16    });
17
18    fireEvent.click(screen.getByText('Lưu'));
19
20    await waitFor(() => {
21      expect(screen.getByText('Thêm sản phẩm thành công'))
22        .toBeInTheDocument();
23    });
24  });
25 });
```

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:



```
1 @WebMvcTest(ProductController.class)
2 @DisplayName("Product API Integration Tests")
3 class ProductControllerIntegrationTest {
4
5     @Autowired
6     private MockMvc mockMvc;
7
8     @MockBean
9     private ProductService productService;
10
11     @Test
12     @DisplayName("GET /api/products - Lay danh sach san pham")
13     void testGetAllProducts() throws Exception {
14         List<ProductDto> products = Arrays.asList(
15             new ProductDto("Laptop", 15000000, 10),
16             new ProductDto("Mouse", 200000, 50)
17         );
18
19         when(productService.getAllProducts(any()))
20             .thenReturn(products);
21
22         mockMvc.perform(get("/api/products"))
23             .andExpect(status().isOk())
24             .andExpect(jsonPath("$", hasSize(2)))
25             .andExpect(jsonPath("$[0].name").value("Laptop"));
26     }
27
28     @Test
29     @DisplayName("POST /api/products - Tao san pham moi")
30     void testCreateProduct() throws Exception {
31         // Implementation here
32     }
33 }
```

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:

```
1 import { render, screen, fireEvent, waitFor } from '@testing-library/react';
2 import Login from './Login';
3 import * as authService from '../services/authService';
4
5 jest.mock('../services/authService');
6
7 describe('Login Mock Tests', () => {
8   beforeEach(() => {
9     jest.clearAllMocks();
10   });
11
12   test('Mock: Login thanh cong', async () => {
13     authService.loginUser.mockResolvedValue({
14       success: true,
15       token: 'mock-token-123',
16       user: { username: 'testuser' }
17     });
18
19     render(<Login />);
20
21     fireEvent.change(screen.getByTestId('username-input'), {
22       target: { value: 'testuser' }
23     });
24     fireEvent.change(screen.getByTestId('password-input'), {
25       target: { value: 'Test123' }
26     });
27     fireEvent.click(screen.getByTestId('login-button'));
28
29     await waitFor(() => {
30       expect(authService.loginUser).toHaveBeenCalledWith(
31         'testuser', 'Test123'
32       );
33       expect(screen.getByText(/thanh cong/i)).toBeInTheDocument();
34     });
35   });
36
37   test('Mock: Login that bai', async () => {
38     authService.loginUser.mockRejectedValue({
39       message: 'Invalid credentials'
40     });
41
42     // Implementation here
43   });
```



44 });

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:

```
1 @WebMvcTest(AuthController.class)
2 class AuthControllerMockTest {
3
4     @Autowired
5     private MockMvc mockMvc;
6
7     @MockBean
8     private AuthService authService;
9
10    @Test
11    @DisplayName("Mock: Controller voi mocked service success")
12    void testLoginWithMockedService() throws Exception {
13        LoginResponse mockResponse = new LoginResponse(
14            true, "Success", "mock-token"
15        );
16
17        when(authService.authenticate(any()))
18            .thenReturn(mockResponse);
19
20        mockMvc.perform(post("/api/auth/login")
21            .contentType(MediaType.APPLICATION_JSON)
22            .content("{\"username\":\"test\",\"password\":\"Pass123\"}"))
23            .andExpect(status().isOk());
24
25        verify(authService, times(1)).authenticate(any());
26    }
27 }
```

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:

```
1 import * as productService from '../services/productService';
2
3 jest.mock('../services/productService');
4
5 describe('Product Mock Tests', () => {
6   test('Mock: Create product thanh cong', async () => {
7     const mockProduct = {
8       id: 1,
9       name: 'Laptop',
10      price: 15000000
11    };
12
13    productService.createProduct.mockResolvedValue(mockProduct);
14
15    // Test implementation
16
17    expect(productService.createProduct).toHaveBeenCalledTimes(1);
18  });
19
20   test('Mock: Get products with pagination', async () => {
21     const mockProducts = {
22       data: [/* products */],
23       page: 1,
24       total: 100
25     };
26
27     productService.getProducts.mockResolvedValue(mockProducts);
28
29     // Test implementation
30   });
31 });
```

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:

```
1 class ProductServiceMockTest {
2
3     @Mock
4     private ProductRepository productRepository;
5
6     @InjectMocks
7     private ProductService productService;
8
9     @Test
10    void testGetProductById() {
11        Product mockProduct = new Product(
12            1L, "Laptop", 15000000
13        );
14
15        when(productRepository.findById(1L))
16            .thenReturn(Optional.of(mockProduct));
17
18        ProductDto result = productService.getProductById(1L);
19
20        assertNotNull(result);
21        assertEquals("Laptop", result.getName());
22
23        verify(productRepository).findById(1L);
24    }
25 }
```

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:

```
1 describe('Login E2E Tests', () => {
2   beforeEach(() => {
3     cy.visit('http://localhost:3000');
4   });
5
6   it('Nên hiển thị form login', () => {
7     cy.get('[data-testid="username-input"]').should('be.visible');
8     cy.get('[data-testid="password-input"]').should('be.visible');
9     cy.get('[data-testid="login-button"]').should('be.visible');
10  });
11
12  it('Nên login thành công với credentials hợp lệ', () => {
13    cy.get('[data-testid="username-input"]').type('testuser');
14    cy.get('[data-testid="password-input"]').type('Test123');
15    cy.get('[data-testid="login-button"]').click();
16
17    cy.get('[data-testid="login-message"]')
18      .should('contain', 'thành công');
19    cy.url().should('include', '/dashboard');
20  });
21
22  it('Nên hiển thị lỗi với credentials không hợp lệ', () => {
23    cy.get('[data-testid="username-input"]').type('ab');
24    cy.get('[data-testid="password-input"]').type('123');
25    cy.get('[data-testid="login-button"]').click();
26
27    cy.get('[data-testid="username-error"]')
28      .should('be.visible');
29  });
30 });
```

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:

```
1 name: Login Tests CI
2
3 on:
4   push:
5     branches: [ main, develop ]
6   pull_request:
7     branches: [ main ]
8
9 jobs:
10  test-login:
11    runs-on: ubuntu-latest
12    steps:
13      - uses: actions/checkout@v2
14
15      - name: Setup Node.js
16        uses: actions/setup-node@v2
17        with:
18          node-version: '18'
19
20      - name: Install dependencies
21        run: |
22          cd frontend
23          npm install
24
25      - name: Run Login Unit Tests
26        run: |
27          cd frontend
28          npm test -- --testPathPattern=Login
29
30      - name: Run Login E2E Tests
31        run: |
32          cd frontend
33          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:

```
1 class ProductPage {
2   visit() {
3     cy.visit('/products');
4   }
5
6   clickAddNew() {
7     cy.get('[data-testid="add-product-btn"]').click();
8   }
9
10  fillProductForm(product) {
11    cy.get('[data-testid="product-name"]').type(product.name);
12    cy.get('[data-testid="product-price"]').type(product.price);
13    cy.get('[data-testid="product-quantity"]').type(product.quantity);
14  }
15
16  submitForm() {
17    cy.get('[data-testid="submit-btn"]').click();
18  }
19
20  getSuccessMessage() {
21    return cy.get('[data-testid="success-message"]');
22  }
23
24  getProductInList(name) {
25    return cy.contains('[data-testid="product-item"]', name);
26  }
27 }
28
29 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:

```
1 import ProductPage from '../pages/ProductPage';
2
3 describe('Product E2E Tests', () => {
4   const productPage = new ProductPage();
```




```
5
6 beforeEach(() => {
7   cy.login('testuser', 'Test123'); // Custom command
8   productPage.visit();
9 });
10
11 it('Nen tao san pham moi thanh cong', () => {
12   productPage.clickAddNew();
13   productPage.fillProductForm({
14     name: 'Laptop Dell',
15     price: '15000000',
16     quantity: '10'
17   });
18   productPage.submitForm();
19
20   productPage.getSuccessMessage()
21     .should('contain', 'thanh cong');
22   productPage.getProductInList('Laptop Dell')
23     .should('exist');
24 });
25
26 it('Nen cap nhat san pham thanh cong', () => {
27   productPage.getProductInList('Laptop Dell').click();
28   cy.get('[data-testid="edit-btn"]').click();
29   cy.get('[data-testid="product-price"]').clear().type('14000000');
30   productPage.submitForm();
31
32   cy.get('[data-testid="product-price"]')
33     .should('contain', '14,000,000');
34 });
35
36 it('Nen xoa san pham thanh cong', () => {
37   productPage.getProductInList('Laptop Dell').click();
38   cy.get('[data-testid="delete-btn"]').click();
39   cy.get('[data-testid="confirm-delete"]').click();
40
41   productPage.getProductInList('Laptop Dell')
42     .should('not.exist');
43 });
44 });
```

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:

```
1 name: Complete CI/CD Pipeline
2
3 on:
4   push:
5     branches: [ main ]
6
7 jobs:
8   test:
9     runs-on: ubuntu-latest
10
11     services:
12       postgres:
13         image: postgres:14
```



```
14     env:
15       POSTGRES_DB: testdb
16       POSTGRES_PASSWORD: password
17     ports:
18       - 5432:5432
19
20     steps:
21     - uses: actions/checkout@v2
22
23     - name: Setup Java
24       uses: actions/setup-java@v2
25       with:
26         java-version: '17'
27
28     - name: Setup Node.js
29       uses: actions/setup-node@v2
30       with:
31         node-version: '18'
32
33     - name: Backend Tests
34       run: |
35         cd backend
36         ./mvnw clean test
37
38     - name: Frontend Tests
39       run: |
40         cd frontend
41         npm install
42         npm test -- --coverage
43
44     - name: E2E Tests
45       run: |
46         cd frontend
47         npm run test:e2e
48
49     - name: Upload Coverage
50       uses: codecov/codecov-action@v2
51       with:
52         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 $\geq 80\%$
- 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