litemall测试用例及清单

**单元测试用例设计**

| **模块** | **测试类型** | **核心测试点** | **设计思路** |
| --- | --- | --- | --- |
| **数据库初始化** | **数据层测试** | **数据库创建、表结构生成、初始数据插入、连接有效性** | **通过JdbcTemplate直接查询系统表（如information\_schema.TABLES）验证结构，查询业务表验证数据。** |
| **卖家功能** | **接口测试** | **登录验证、商品 CRUD、商品状态管理（上架 / 下架 / 冻结）、购买意向处理** | **模拟 HTTP 请求调用后端接口，通过MockMvc断言响应状态、JSON 字段值及数据库状态变化。** |
| **买家功能** | **接口测试** | **商品列表查看、购买意向提交、商品状态联动（提交后冻结）** | **先通过接口创建测试数据（如商品），再调用目标接口验证功能及数据流转正确性。** |
| **前端交互** | **页面功能测试** | **路由跳转（返回主界面）、图片加载失败处理、状态展示一致性** | **模拟用户操作（如点击按钮），验证路由变化、DOM 元素状态及错误处理逻辑。** |

**单元测试用例清单**

| **用例 ID** | **功能点** | **测试单元（类 / 方法）** | **测试对象（核心类 / 接口）** | **测试步骤** | **预期结果** | **优先级** | **实际结果** |
| --- | --- | --- | --- | --- | --- | --- | --- |
| DB-001 | 数据库创建 | 手动执行 | MySQL  数据库服务 | 1. 执行 SQL 语句 CREATE DATABASE simple\_shop CHARACTER SET utf8mb4 COLLATE utf8mb4\_general\_ci | 数据库创建成功，查询 SHOW DATABASES 包含 simple\_shop | 高 | 执行成功 |
| DB-002 | 数据表创建 | DatabaseInitializationTest#testTablesCreated | 数据库初始化脚本、Product 实体类、PurchaseIntention 实体类、Seller 实体类 | 1. 执行初始化 SQL 脚本创建表结构  2. 调用测试方法，通过 JdbcTemplate 查询 information\_schema.TABLES 验证表存在 | 测试断言通过，product、purchase\_intention、seller 表均存在（查询结果 count=1） | 高 | 执行成功 |
| DB-003 | 初始卖家数据插入 | DatabaseInitializationTest#testInitialSellerData | Seller 实体类、卖家数据初始化脚本 | 1. 执行初始化 SQL 插入卖家数据  2. 调用测试方法，通过 JdbcTemplate 查询 seller 表中 username='seller' 的记录 | 测试断言通过，查询到密码为 123 的卖家记录 | 高 | 执行成功 |
| DB-004 | 数据库配置验证 | DatabaseInitializationTest#testDatabaseConnection | application.properties 配置文件、Spring DataSource 配置 | 1. 启动 Spring 容器，自动加载 application.properties 配置  2. 调用测试方法，执行 SELECT 1 验证连接 | 测试断言通过，查询返回结果为 1，无连接异常 | 高 | 执行成功 |
| S-001 | 卖家登录 | SellerLoginTest#testSellerLogin\_Success、SellerLoginTest#testSellerLogin\_Failure | SellerController（登录接口）、SellerService（登录验证逻辑） | 1. 调用测试方法，通过 MockMvc 模拟 POST 请求 /api/seller/login  2. 分别传入正确（seller/123）和错误（seller/wrongpass）的账号密码 | 成功用例返回 200 OK 和 true，失败用例返回 200 OK 和 false | 高 | 执行成功 |
| S-002 | 发布新商品 | ProductManagementTest#testCreateProduct | ProductController（创建商品接口）、ProductService（商品保存逻辑）、Product 实体类 | 1. 调用测试方法，通过 MockMvc 模拟 POST 请求 /api/products，传入 JSON 格式商品数据（名称、价格等）  2. 验证响应 JSON 中的商品信息 | 响应状态为 200 OK，返回的商品 JSON 中 name 与输入一致，isActive 为 true | 高 | 执行成功 |
| S-003 | 编辑商品 | ProductManagementTest#testUpdateProduct | ProductController（更新商品接口）、ProductService（商品更新逻辑）、Product 实体类 | 1. 先创建商品获取 ID  2. 调用测试方法，通过 MockMvc 模拟 PUT 请求 /api/products/{id}，传入修改后的 JSON 数据  3. 验证响应及数据库中商品信息 | 响应状态为 200 OK，数据库中对应商品字段（如价格）更新为修改后的值 | 中 | 执行成功 |
| S-004 | 商品上架 / 下架 | ProductManagementTest#testActivateDeactivateProduct | ProductController（上架 / 下架接口）、ProductService（状态更新逻辑）、Product 实体类 | 1. 先创建商品获取 ID  2. 调用测试方法，通过 MockMvc 模拟 PUT 请求 /api/products/{id}/activate 验证上架  3. 模拟请求 /api/products/{id}/freeze 验证下架 | 上架后响应 JSON 中 isActive 为 true，下架后为 false | 高 | 执行成功 |
| S-005 | 冻结 / 解冻商品 | ProductManagementTest#testFreezeProduct | ProductController（冻结 / 解冻接口）、ProductService（冻结状态逻辑）、Product 实体类 | 1. 先创建商品获取 ID  2. 调用测试方法，通过 MockMvc 模拟 PUT 请求 /api/products/{id}/freeze?freeze=true 验证冻结  3. 模拟请求 /api/products/{id}/freeze?freeze=false 验证解冻  4. 调用 GET 请求 /api/products/{id} 验证状态 | 冻结后响应 JSON 中 isFrozen 为 true，解冻后为 false | 高 | 执行成功 |
| S-006 | 查看购买意向 | PurchaseIntentionTest#testViewPurchaseIntentions | BuyerController（查询购买意向接口）、PurchaseIntentionService（查询逻辑） | 1. 调用测试方法，通过 MockMvc 模拟 GET 请求 /api/buyers  2. 验证响应格式和内容 | 响应状态为 200 OK，返回 JSON 数组（为空或包含购买意向记录） | 高 | 执行成功 |
| S-007 | 处理购买意向 | PurchaseIntentionTest#testHandlePurchaseIntention | BuyerController（处理意向接口）、PurchaseIntentionService（状态更新逻辑） | 1. 先创建商品和购买意向获取 ID  2. 调用测试方法，通过 MockMvc 模拟 PUT 请求 /api/buyers/{id}/complete?success=true 验证交易成功  3. 模拟请求 ?success=false 验证交易失败 | 响应状态为 200 OK，返回 true；交易成功时商品 isActive 为 false，失败时商品 isFrozen 为 false | 高 | 执行成功 |
| S-008 | 返回主界面 | （前端测试）SellerDashboard.vue 导航测试 | 前端路由配置（router/index.js）、SellerDashboard 组件 | 1. 模拟点击卖家后台 "返回买家平台" 按钮  2. 验证路由跳转 | 路由成功跳转至 /（买家首页），URL 为 http://localhost:8080 | 中 | 执行成功 |
| B-001 | 查看商品 | BuyerFunctionTest#testViewProduct | ProductController（查询在售商品接口）、ProductService（查询在售商品逻辑） | 1. 先创建在售商品  2. 调用测试方法，通过 MockMvc 模拟 GET 请求 /api/products/active  3. 验证响应 JSON 中的商品信息 | 响应状态为 200 OK，返回的商品 JSON 中包含已创建的商品信息（如名称、价格） | 高 | 执行成功 |
| B-002 | 提交购买意向 | BuyerFunctionTest#testSubmitPurchaseIntention | BuyerController（提交意向接口）、PurchaseIntentionService（保存意向逻辑）、Buyer 实体类 | 1. 先创建商品获取 ID  2. 调用测试方法，通过 MockMvc 模拟 POST 请求 /api/buyers/product/{id}，传入买家信息 JSON  3. 调用 GET 请求 /api/products/{id} 验证商品状态 | 响应状态为 200 OK，返回买家信息 JSON；商品 isFrozen 字段变为 true | 高 | 执行成功 |
| B-003 | 图片加载失败处理 | （前端测试）HomeView.vue 图片加载测试 | 前端 ProductImage 组件、图片加载失败处理工具函数 | 1. 创建商品时传入无效图片 URL（如 http://invalid.url/img.jpg）  2. 模拟访问买家页面，检查图片元素 | 图片加载失败时显示默认占位图，控制台无 404 错误日志，页面渲染正常 | 中 | 执行成功 |

**功能测试**

**1. 卖家模块测试用例**

| 测试 ID | 测试场景 | 输入数据 | 预期结果 | 优先级 |
| --- | --- | --- | --- | --- |
| S-001 | 卖家登录（正确密码） | 用户名：seller，密码：123 | 登录成功，跳转至卖家管理页 | 高 |
| S-002 | 卖家登录（错误密码） | 用户名：seller，密码：wrong | 登录失败，显示错误提示 | 高 |
| S-003 | 发布新商品 | 商品名："测试商品"，价格：99.9，描述："测试描述" | 商品发布成功，列表中显示新商品 | 高 |
| S-004 | 商品上架 / 下架 | 选择商品，点击 "上架"/"下架" 按钮 | 商品状态更新为 "在售"/"下架" | 中 |
| S-005 | 冻结 / 解冻商品 | 选择商品，点击 "冻结"/"解冻" 按钮 | 商品状态更新为 "已冻结"/"正常" | 中 |
| S-006 | 查看购买意向列表 | 点击购买意向 | 显示所有买家提交的购买意向 | 高 |
| S-007 | 处理购买意向（成功） | 选择意向，点击 "交易成功" | 状态更新为 "交易成功"，商品状态更新 | 高 |
| S-008 | 处理购买意向（失败） | 选择意向，点击 "交易失败" | 状态更新为 "交易失败"，商品状态更新 | 高 |

**2. 买家模块测试用例**

| **测试 ID** | **测试场景** | **输入数据** | **预期结果** | **优先级** |
| --- | --- | --- | --- | --- |
| B-001 | 查看商品列表 | 无 | 显示所有在售商品 | 高 |
| B-002 | 提交购买意向 | 商品 ID：1，买家信息：姓名 "测试买家"，电话 "13800138000"，地址 "测试地址" | 提交成功，商品状态变为 "冻结" | 高 |
| B-003 | 查看商品详情 | 点击任意商品 | 显示商品完整信息（名称、价格、描述等） | 中 |