电商平台测试用例

1. 需求输入阶段

1.1 用户输入初始需求

用户通过系统界面输入以下需求：

```
我想开发一个电商平台，主要功能包括：
1. 用户注册和登录
2. 商品展示和搜索
3. 购物车功能
4. 订单管理
5. 支付功能
```

1.2 AI 需求分析

系统调用 DeepSeek API 分析需求，生成以下分析结果：

```json
{
  "requirement_analysis": {
    "completeness": {
      "score": 7,
      "missing_points": [
        "用户角色和权限管理",
        "商品分类管理",
        "商品评价系统",
        "物流配送管理",
        "售后服务"
      ]
    },
    "technical_challenges": [
      "高并发订单处理",
      "库存管理",
      "支付安全",
      "数据一致性"
    ],
    "suggested_features": [
      "用户角色：普通用户、商家、管理员",
      "商品管理：分类、属性、库存",
      "订单状态流转",
      "支付集成：支付宝、微信支付",
      "物流跟踪"
    ]
  }
}
```

2. 需求完善阶段

2.1 用户确认和补充需求

用户根据 AI 分析结果，补充以下需求：

```
补充以下功能：
1. 用户分为普通用户、商家和管理员三种角色
2. 商品需要支持分类和属性管理
3. 添加商品评价功能
4. 集成支付宝和微信支付
5. 添加物流跟踪功能
```

2.2 数据库设计建议

系统生成数据库设计建议：

```sql
-- 用户表
CREATE TABLE users (
    id SERIAL PRIMARY KEY,
    username VARCHAR(50) UNIQUE NOT NULL,
    password VARCHAR(255) NOT NULL,
    email VARCHAR(100) UNIQUE NOT NULL,
    role VARCHAR(20) NOT NULL,  -- 'user', 'merchant', 'admin'
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- 商品表
CREATE TABLE products (
    id SERIAL PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    description TEXT,
    price DECIMAL(10,2) NOT NULL,
    stock INTEGER NOT NULL,
    category_id INTEGER REFERENCES categories(id),
    merchant_id INTEGER REFERENCES users(id),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- 订单表
CREATE TABLE orders (
    id SERIAL PRIMARY KEY,
    user_id INTEGER REFERENCES users(id),
    total_amount DECIMAL(10,2) NOT NULL,
    status VARCHAR(20) NOT NULL,  -- 'pending', 'paid', 'shipped', 'completed'
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

3. 代码生成阶段

3.1 后端 API 生成

系统生成 Django REST framework 的 API 代码：

```python
# views.py
from rest_framework import viewsets
from rest_framework.permissions import IsAuthenticated
from .models import Product, Order
from .serializers import ProductSerializer, OrderSerializer

class ProductViewSet(viewsets.ModelViewSet):
    queryset = Product.objects.all()
    serializer_class = ProductSerializer

    def get_queryset(self):
        queryset = Product.objects.all()
        category = self.request.query_params.get('category', None)
        if category:
            queryset = queryset.filter(category_id=category)
        return queryset

class OrderViewSet(viewsets.ModelViewSet):
    serializer_class = OrderSerializer
    permission_classes = [IsAuthenticated]

    def get_queryset(self):
        return Order.objects.filter(user=self.request.user)

    def perform_create(self, serializer):
        serializer.save(user=self.request.user)
```

3.2 前端组件生成

系统生成 Vue 3 组件代码：

```
<!-- ProductList.vue -->
<template>
  <div class="product-list">
    <el-row :gutter="20">
      <el-col :span="6" v-for="product in products" :key="product.id">
        <el-card :body-style="{ padding: '0px' }">
          <img :src="product.image" class="image" />
          <div style="padding: 14px;">
            <span>{{ product.name }}</span>
            <div class="bottom">
              <span class="price">¥{{ product.price }}</span>
              <el-button type="primary" @click="addToCart(product)">
                加入购物车
              </el-button>
            </div>
          </div>
        </el-card>
      </el-col>
    </el-row>
  </div>
</template>

<script setup>
import { ref, onMounted } from "vue";
import { useStore } from "vuex";
import { ElMessage } from "element-plus";

const store = useStore();
const products = ref([]);

onMounted(async () => {
  try {
    const response = await store.dispatch("fetchProducts");
    products.value = response.data;
  } catch (error) {
    ElMessage.error("获取商品列表失败");
  }
});

const addToCart = async (product) => {
  try {
    await store.dispatch("addToCart", product);
    ElMessage.success("添加成功");
  } catch (error) {
    ElMessage.error("添加失败");
  }
};
</script>
```

4. 测试用例生成

4.1 API 测试用例

系统生成 API 测试用例:

```python
# tests.py
from django.test import TestCase
from rest_framework.test import APIClient
from django.contrib.auth import get_user_model
from .models import Product, Order

class EcommerceAPITests(TestCase):
    def setUp(self):
        self.client = APIClient()
        self.user = get_user_model().objects.create_user(
            username='testuser',
            password='testpass123'
        )
        self.client.force_authenticate(user=self.user)

    def test_create_order(self):
        product = Product.objects.create(
            name='Test Product',
            price=99.99,
            stock=10
        )
        response = self.client.post('/api/orders/', {
            'product_id': product.id,
            'quantity': 1
        })
        self.assertEqual(response.status_code, 201)
        self.assertEqual(Order.objects.count(), 1)
```

4.2 前端测试用例

系统生成前端测试用例：

```javascript
// ProductList.test.js
import { mount } from "@vue/test-utils";
import { createStore } from "vuex";
import ProductList from "./ProductList.vue";

describe("ProductList.vue", () => {
  let store;

  beforeEach(() => {
    store = createStore({
      state: {
        products: [],
      },
      actions: {
        fetchProducts: jest.fn(),
      },
    });
  });

  it("renders product list correctly", () => {
    const wrapper = mount(ProductList, {
      global: {
        plugins: [store],
      },
    });
    expect(wrapper.find(".product-list").exists()).toBe(true);
  });
});
```

5. 系统工作流程

1. **需求输入**：
   - 用户通过系统界面输入初始需求
   - 系统调用 AI 分析需求完整性
   - 系统提供需求完善建议
2. **需求完善**：
   - 用户根据 AI 建议补充需求

    - 系统生成数据库设计建议
    - 用户确认数据库设计
3. **代码生成**：
    - 系统生成后端 API 代码
    - 系统生成前端组件代码
    - 系统生成测试用例
4. **代码部署**：
    - 系统提供代码部署指南
    - 用户按照指南部署代码
    - 系统提供部署验证

6. 预期输出

1. **需求文档**：
    - 完整的需求规格说明书
    - 数据库设计文档
    - API 接口文档
2. **代码文件**：
    - 后端 API 代码
    - 前端组件代码
    - 测试用例代码
3. **部署文档**：
    - 环境配置指南
    - 部署步骤说明
    - 验证测试指南

7. 需求变更处理

7.1 用户补充新需求

用户在使用过程中发现需要添加新功能，通过系统界面输入：

```
我需要添加以下功能：
1. 商品推荐系统
2. 用户积分系统
3. 优惠券管理
4. 商品评论的点赞功能
5. 商品收藏功能
```

7.2 AI 需求分析

系统分析新需求与现有系统的关系：

```json
{
  "requirement_analysis": {
    "impact_analysis": {
      "affected_modules": ["用户系统", "商品系统", "订单系统"],
      "new_modules": ["推荐系统", "积分系统", "优惠券系统"],
      "database_changes": [
        "添加积分表",
        "添加优惠券表",
        "添加收藏表",
        "修改评论表"
      ]
    },
    "suggested_implementation": {
      "priority": "medium",
      "estimated_effort": "3 days",
      "dependencies": ["用户系统", "商品系统"]
    }
  }
}
```

7.3 数据库更新

系统生成数据库更新脚本：

```sql
-- 积分表
CREATE TABLE user_points (
    id SERIAL PRIMARY KEY,
    user_id INTEGER REFERENCES users(id),
    points INTEGER NOT NULL DEFAULT 0,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- 积分历史表
CREATE TABLE point_history (
    id SERIAL PRIMARY KEY,
    user_id INTEGER REFERENCES users(id),
    points INTEGER NOT NULL,
    type VARCHAR(20) NOT NULL,  -- 'earn', 'spend'
    description TEXT,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- 优惠券表
CREATE TABLE coupons (
    id SERIAL PRIMARY KEY,
    code VARCHAR(50) UNIQUE NOT NULL,
    discount_type VARCHAR(20) NOT NULL,  -- 'percentage', 'fixed'
    discount_value DECIMAL(10,2) NOT NULL,
    min_purchase DECIMAL(10,2),
    start_date TIMESTAMP NOT NULL,
    end_date TIMESTAMP NOT NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- 用户优惠券表
CREATE TABLE user_coupons (
    id SERIAL PRIMARY KEY,
    user_id INTEGER REFERENCES users(id),
    coupon_id INTEGER REFERENCES coupons(id),
    is_used BOOLEAN DEFAULT FALSE,
    used_at TIMESTAMP,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- 商品收藏表
CREATE TABLE favorites (
    id SERIAL PRIMARY KEY,
    user_id INTEGER REFERENCES users(id),
    product_id INTEGER REFERENCES products(id),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    UNIQUE(user_id, product_id)
);

-- 修改评论表，添加点赞字段
ALTER TABLE comments ADD COLUMN likes INTEGER DEFAULT 0;
```

7.4 代码更新

系统生成新的 API 和组件代码：

```python
# views.py
class UserPointsViewSet(viewsets.ModelViewSet):
    serializer_class = UserPointsSerializer
    permission_classes = [IsAuthenticated]

    def get_queryset(self):
        return UserPoints.objects.filter(user=self.request.user)

    @action(detail=False, methods=['get'])
    def history(self, request):
        history = PointHistory.objects.filter(user=request.user)
        serializer = PointHistorySerializer(history, many=True)
        return Response(serializer.data)

class CouponViewSet(viewsets.ModelViewSet):
    queryset = Coupon.objects.all()
    serializer_class = CouponSerializer

    @action(detail=True, methods=['post'])
    def claim(self, request, pk=None):
        coupon = self.get_object()
        if not coupon.is_valid():
            return Response(
                {"detail": "优惠券已过期"},
                status=status.HTTP_400_BAD_REQUEST
            )
        UserCoupon.objects.create(
            user=request.user,
            coupon=coupon
        )
        return Response({"detail": "领取成功"})
```

```
<!-- UserPoints.vue -->
<template>
  <div class="user-points">
    <el-card>
      <template #header>
        <div class="card-header">
          <span>我的积分</span>
        </div>
      </template>
      <div class="points-display">
        <h2>{{ points }} 积分</h2>
      </div>
      <el-table :data="pointHistory" style="width: 100%">
        <el-table-column prop="points" label="积分变动" />
        <el-table-column prop="type" label="类型" />
        <el-table-column prop="description" label="说明" />
        <el-table-column prop="created_at" label="时间" />
      </el-table>
    </el-card>
  </div>
</template>

<script setup>
import { ref, onMounted } from "vue";
import { useStore } from "vuex";
import { ElMessage } from "element-plus";

const store = useStore();
const points = ref(0);
const pointHistory = ref([]);

onMounted(async () => {
  try {
    const response = await store.dispatch("fetchUserPoints");
    points.value = response.data.points;
    const historyResponse = await store.dispatch("fetchPointHistory");
    pointHistory.value = historyResponse.data;
  } catch (error) {
    ElMessage.error("获取积分信息失败");
  }
});
</script>
```

7.5 测试用例更新

系统生成新的测试用例：

```python
# tests.py
class UserPointsTests(TestCase):
    def setUp(self):
        self.client = APIClient()
        self.user = get_user_model().objects.create_user(
            username='testuser',
            password='testpass123'
        )
        self.client.force_authenticate(user=self.user)

    def test_earn_points(self):
        response = self.client.post('/api/points/earn/', {
            'points': 100,
            'description': '购物奖励'
        })
        self.assertEqual(response.status_code, 201)
        self.assertEqual(UserPoints.objects.get(user=self.user).points, 100)

    def test_point_history(self):
        UserPoints.objects.create(user=self.user, points=100)
        PointHistory.objects.create(
            user=self.user,
            points=100,
            type='earn',
            description='购物奖励'
        )
        response = self.client.get('/api/points/history/')
        self.assertEqual(response.status_code, 200)
        self.assertEqual(len(response.data), 1)
```

```
// UserPoints.test.js
describe("UserPoints.vue", () => {
  let store;

  beforeEach(() => {
    store = createStore({
      state: {
        points: 100,
        pointHistory: [
          {
            points: 100,
            type: "earn",
            description: "购物奖励",
            created_at: "2024-03-21T10:00:00Z",
          },
        ],
      },
      actions: {
        fetchUserPoints: jest.fn(),
        fetchPointHistory: jest.fn(),
      },
    });
  });

  it("displays points correctly", () => {
    const wrapper = mount(UserPoints, {
      global: {
        plugins: [store],
      },
    });
    expect(wrapper.find(".points-display h2").text()).toBe("100 积分");
  });
});
```

7.6 系统处理流程

1. **需求变更识别**：
    - 用户提交新需求
    - 系统分析需求与现有系统的关系
    - 系统评估影响范围
2. **数据库更新**：

    - 系统生成数据库更新脚本
    - 用户确认数据库变更
    - 系统执行数据库更新
3. **代码更新**：
    - 系统生成新的 API 代码
    - 系统生成新的前端组件
    - 系统更新现有代码
4. **测试用例更新**：
    - 系统生成新的测试用例
    - 系统更新现有测试用例
    - 系统执行测试验证
5. **部署更新**：
    - 系统生成部署指南
    - 用户按照指南更新系统
    - 系统验证更新结果

7.7 版本控制

系统自动记录需求变更历史：

```json
{
  "version": "1.1.0",
  "changes": [
    {
      "type": "feature",
      "description": "添加积分系统",
      "date": "2024-03-21",
      "author": "system"
    },
    {
      "type": "feature",
      "description": "添加优惠券系统",
      "date": "2024-03-21",
      "author": "system"
    },
    {
      "type": "feature",
      "description": "添加商品收藏功能",
      "date": "2024-03-21",
      "author": "system"
    }
  ],
  "database_changes": [
    "添加积分表",
    "添加优惠券表",
    "添加收藏表",
    "修改评论表"
  ],
  "api_changes": ["添加积分相关API", "添加优惠券相关API", "添加收藏相关API"]
}
```

通过这种方式，系统能够灵活地处理用户后续补充的需求，确保系统的可扩展性和可维护性。每次需求变更都会触发相应的代码更新和测试用例更新，保证系统的质量和稳定性。