01.功能概述
02.需求分析
  2.1 用户场景
  2.2 功能范围
03.技术方案
  3.1 方案一：扩展现有购物车扫码逻辑
  3.2 方案二：独立快速支付模块
04.实现规划
  4.1 技术选型
  4.2 任务拆解
  4.3 代码路径
05.兼容性设计
  5.1 设备适配
  5.2 冲突检查
06.测试方案
  6.1 核心用例
  6.2 性能指标
07.风险评估
  7.1 技术风险
  7.2 业务风险
  7.3 用户体验风险
08.文档记录
  8.1 技术文档
  8.2 用户文档
  8.3 监控埋点

01.功能概述

- **功能ID**：`FEAT-20250624-001`
- **功能名称**：购物车主页付款码提前支付逻辑
- **目标版本**：v0.2.0
- **提交人**：@panruiqi
- **状态**：
  - [x] ⏳ 设计中 /
  - [ ] ⏳ 开发中 /
  - [ ] ☑ 已完成 /
  - [ ] ❌ 已取消
- **价值评估**：
  - [x] ★★★★★ 核心业务功能
  - [ ] ★★★★ 用户体验优化
  - [ ] ★★★ 辅助功能增强
  - [ ] ★★ 技术债务清理
- **功能描述**
  - POS机扫码的时候，用户扫完商品，不需要点击确认支付按钮，直接扫收款码也可确认完成支付。也就是提前扫码不需要确认，现有流程保留并存。

02.需求分析

2.1 用户场景

- **主要场景**:
  - 用户A扫描了一些商品，购物车有商品，显示购物车页面
  - 此时，用户A希望购物效率更高，不惦记确认支付按钮，其直接扫码付款码，然后直接支付。
- **边界场景**:

2.2 功能范围

- ☑ 包含:
- ❌ 不包含:

03.技术方案

3.1 方案一：扩展现有购物车扫码逻辑

- 实现思路:
  - 在CartViewModel中增加支付状态管理
  - 扩展handleScanResult方法，增加付款码识别分支
  - 购物车页面检测到付款码时，自动触发支付流程
  - 复用现有的支付UseCase和Repository
- `购物车页面 → 扫描付款码 → 计算总价 → 生成订单 → 执行支付 → 支付成功页面`
- 状态设计:
  - CartState.ReadyForQuickPayment - 允许快速支付状态
  - CartState.QuickPaymentProcessing - 快速支付处理中
  - CartState.QuickPaymentSuccess - 快速支付成功

3.2 方案二：独立快速支付模块

- 实现思路:
  - 创建QuickPaymentfeature专门处理购物车快速支付
  - 在购物车页面注入快速支付能力
  - 保持购物车和支付逻辑相对独立
  - 便于后续扩展和维护

04.实现规划

4.1 技术选型

如果采用方案一，相对节约时间。但是方案二从项目设计角度会更好，因为购物车不应该耦合付款逻辑，付款逻辑应当是独立的，如果main中耦合了过多的逻辑，可能产生未知的错误。如果采用独立快速支付模块，那么后续有同样的需求时，会更好的复用，实现时间更短。

这是方案一的可能的实现，方案二待设计。

核心组件：

- CartViewModel - 扩展支付状态管理
- QuickPaymentUseCase - 快速支付业务逻辑
- BarcodeValidator - 复用现有付款码识别
- PaymentRepository - 复用现有支付接口

状态流设计：

- 购物车状态 + 快速支付状态的组合管理
- 使用StateFlow进行状态同步

4.2 任务拆解

- 扫码识别增强 (4h)
  - 在购物车页面添加付款码识别逻辑
  - 区分商品码和付款码的处理分支
- 快速支付流程 (8h)
  - 实现购物车商品→支付订单的直接转换
  - 集成现有的支付UseCase
  - 处理支付成功/失败状态
- UI状态管理 (4h)
  - 添加快速支付的loading状态
  - 付成功后的页面跳转逻辑
  - 错误处理和用户提示
- 兼容性测试 (4h)
  - 确保原有确认支付流程不受影响
  - 测试各种边界场景

4.3 代码路径

05.兼容性设计

5.1 设备适配

5.2 冲突检查

扫码器兼容：

- 复用现有扫码框架，无需额外适配
- 支持所有已适配的扫码设备

支付方式兼容：

- 微信付款码 - ☑ 支持
- 支付宝付款码 - ☑ 支持
- 其他付款方式 - 按现有支持范围

06.测试方案

6.1 核心用例

正常流程：

1. 购物车有商品 → 扫描付款码 → 支付成功
2. 购物车有商品 → 点击确认支付 → 跳转支付页面 → 扫码支付

异常场景：

1. 购物车为空时扫描付款码 → 提示先添加商品
2. 网络异常时快速支付 → 错误提示和重试机制
3. 付款码无效 → 提示重新扫描

边界测试：

1. 快速支付过程中继续扫描商品码 → 忽略或排队处理
2. 快速支付过程中点击确认支付按钮 → 状态保护
3. 支付成功后的页面状态清理

6.2 性能指标

响应时间：

- 付款码识别响应时间 < 500ms
- 支付请求处理时间 < 3s
- 页面状态更新延迟 < 200ms

成功率指标：

- 付款码识别准确率 > 99%
- 快速支付成功率 > 95%
- 状态同步成功率 > 99.9%

07.风险评估

7.1 技术风险

- 状态管理复杂度增加 - 通过清晰的状态设计缓解
- 支付流程冲突 - 通过状态隔离和优先级管理解决

7.2 业务风险

- 误触发快速支付 - 通过付款码格式严格校验避免
- 支付金额确认缺失 - 可考虑添加快速确认弹窗

7.3 用户体验风险

- 学习成本 - 保持原有流程，新功能作为增强
- 操作混淆 - 通过清晰的UI状态提示引导用户

08.文档记录

8.1 技术文档

8.2 用户文档

8.3 监控埋点