Claude 研发规范

研发目标：
确保这些设备在通过 USB 接入收银台后，系统能够：

1. 自动识别设备信息（名称、型号、规格等）；
2. 判断设备状态是否可用；
3. 在可用状态下与设备进行交互（例如刷卡、扫码、打印测试等）。目前部分外设仍在采购中，后续将按设备到货节奏逐步开展研发，调试与联测。

当前收银台机器硬件配置参数（最低兼容要求）：

1. 操作系统：Android 9（最低需要兼容这个版本）
2. 屏幕：主显：15.6" FHD，1920*1080

后期会使用新收银台，硬件配置（需要考虑后期到新设备上运行）如下：

1. 系统：安卓13
2. 内存：4GB RAM+64GB ROM
3. 屏幕：主显：15.6" FHD，1920*1080
4. 电容多点式触摸
5. 处理器：高通6核 2.4GHz/1.9GHz
6. 联网：Wi-Fi、
7. 网线接口:

2*USB typeA 3.0口

2*USB typeA 2.0口、1*RJ11串口、1*RJ12钱箱口

1*RJ45 LAN口、1*耳机口、1*电源口、1*HDMI口

通用研发规范（长期有效）

以下研发规范为**全局约定**，除非特别说明，后续所有功能开发均应遵守：

1. **组件化开发**
   前端代码尽可能组件化设计，提升复用性与维护性。
2. **单文件行数限制**
   任意代码文件（包括 Dart / Flutter 文件）不得超过 **600 行**。

3. **模块独立与解耦**
   功能模块需明确边界，禁止出现交叉依赖或网状结构。
4. **松耦合架构设计**
   模块之间接口清晰，依赖最小化，便于独立测试与替换。
5. **单一职责原则**
   每个类、函数或组件仅负责单一职责，避免多功能耦合。
6. **依赖变更管控**
   若需修改、替换或新增 SDK 依赖，**必须事先获得我的确认**，无非必要禁止私自变更项目中的依赖配置。
7. **依赖倒置原则**
8. **开闭原则**

Flutter 开发核心规范

📁 项目结构

```
lib/
├── app/                # 应用配置层
│   ├── routes/         # 路由配置 (router_config.dart, route_state_manager.dart)
│   └── theme/          # 主题配置 (app_theme.dart - 全局样式)
├── core/               # 核心基础层
│   ├── utils/          # 工具类 (navigation_helper.dart)
│   └── widgets/        # 全局组件 (toast/loading/dialog/drawer/common_menu/table)
├── modules/            # 业务模块层
│   └── [module_name]/  # 各业务模块 (controllers/views/widgets)
├── shared/             # 跨模块共享层
│   ├── layouts/        # 布局组件 (shell_layout.dart)
│   └── widgets/        # 共享组件
└── data/               # 数据层 (models/services
```

🎨 样式规范 (P0 - 绝对禁止硬编码)

1. 颜色 - 必须使用 AppTheme

```dart
```

// ✅ 正确

Container(color: AppTheme.cardColor)

Text('标题', style: AppTheme.textTitle)

// ❌ 禁止

Container(color: Color(0xFFFFFFFF))

Text('标题', style: TextStyle(color: Color(0xFF333333)))

```

**常用颜色：**

- 文本：`textPrimary` / `textSecondary` / `textTertiary`
- 背景：`cardColor` / `backgroundColor`
- 状态：`successColor` / `errorColor` / `warningColor`
- 边框：`borderColor`

**豁免：** `Colors.white` / `Colors.black` / `Colors.transparent`

2. 圆角 - 必须使用 AppTheme

```dart

// ✅ 正确

BorderRadius.circular(AppTheme.borderRadiusDefault)  // 8.r 默认

BorderRadius.circular(AppTheme.borderRadiusSmall)   // 4.r 按钮

BorderRadius.circular(AppTheme.borderRadiusLarge)   // 12.r 主卡片

// ❌ 禁止

BorderRadius.circular(8)

BorderRadius.circular(12.r)

```

3. 间距 - 必须使用 AppTheme

```dart

// ✅ 正确

SizedBox(height: AppTheme.spacingDefault)  // 16.w 默认

EdgeInsets.all(AppTheme.spacingM)        // 12.w

EdgeInsets.symmetric(horizontal: AppTheme.spacingL)  // 24.w

// ❌ 禁止

SizedBox(height: 16)

EdgeInsets.all(12)

```

间距规范： `spacingXS`(4) / `spacingS`(8) / `spacingM`(12) / `spacingDefault`(16) / `spacingL`(24) / `spacingXL`(32)

4. 文本样式 - 必须使用 AppTheme

```dart

// ✅ 正确

Text('标题', style: AppTheme.textSubtitle)  // 16.sp

Text('内容', style: AppTheme.textBody)     // 14.sp

Text('辅助', style: AppTheme.textCaption)   // 12.sp

// ❌ 禁止

Text('标题', style: TextStyle(fontSize: 16))

```

文本规范： `textCaption`(12) / `textBody`(14) / `textSubtitle`(16) / `textTitle`(18) / `textHeading`(20)

5. 装饰器 - 推荐使用 AppTheme 方法

```dart

// ✅ 推荐

Container(decoration: AppTheme.cardDecoration())

Container(decoration: AppTheme.greyContainerDecoration())

// 自定义参数

AppTheme.cardDecoration(

 borderRadius: AppTheme.borderRadiusXLarge,

 withShadow: false,

 borderColor: AppTheme.primaryColor,

)

```

6. 按钮样式 - 必须明确设置

```dart

```
// ✅ OutlinedButton (次要操作)
OutlinedButton(
 style: OutlinedButton.styleFrom(
  side: BorderSide(color: AppTheme.borderColor, width: 1.w),
  foregroundColor: AppTheme.textPrimary, // 防止文本变灰
  shape: RoundedRectangleBorder(
   borderRadius: BorderRadius.circular(AppTheme.borderRadiusDefault),
  ),
 ),
 child: Text('取消', style: AppTheme.textSubtitle),
)

// ✅ ElevatedButton (主要操作)
ElevatedButton(
 style: ElevatedButton.styleFrom(
  backgroundColor: AppTheme.primaryColor,
  foregroundColor: Colors.white,
  shape: RoundedRectangleBorder(
   borderRadius: BorderRadius.circular(AppTheme.borderRadiusDefault),
  ),
 ),
 child: Text('确定', style: AppTheme.textSubtitle.copyWith(
  fontWeight: FontWeight.w600, color: Colors.white,
 )),
)
```

7. 响应式适配 - 必须使用 ScreenUtil

```dart
// ✅ 必须导入
import 'package:flutter_screenutil/flutter_screenutil.dart';
// ✅ 必须使用
```

Container(width: 100.w, height: 50.h)

Text('标题', style: TextStyle(fontSize: 16.sp))

BorderRadius.circular(8.r)

// ❌ 禁止裸数字

Container(width: 100)

```

## 🛣️ 路由与状态管理

### 1. 路由导航 - 使用 NavigationHelper / AppRouter

```dart

// ✅ 推荐

NavigationHelper.loginToHome();  // 业务逻辑跳转

AppRouter.push('/cashier');      // 页面跳转

AppRouter.replace('/settings');  // 替换当前页面

AppRouter.pop();                 // 返回

// ❌ 禁止

context.go('/home');

Navigator.of(context).push(...);

```

2. Controller 生命周期 - 路由管理

```dart

// ✅ 路由进入时创建 Controller (router_config.dart)

GoRoute(

  path: '/settings',

  pageBuilder: (context, state) {

    if (Get.isRegistered<SettingsController>()) {

      Get.delete<SettingsController>();

    }

    Get.put<SettingsController>(SettingsController());

    return CustomTransitionPage(child: const SettingsPage());

  },

)

```
// ✅ 路由离开时清理 Controller (RouteStateManager)
RouteStateManager.registerCleanupConfigs([
  RouteCleanupConfig(
    routePath: '/settings',
    cleanupFunctions: [
      () => Get.delete<SettingsController>(force: true),
    ],
  ),
]);
// ❌ 禁止在页面 initState/dispose 中管理全局 Controller
```

3. Controller 定义规范

```dart
class XxxController extends GetxController {
  // 响应式状态
  final _counter = 0.obs;
  int get counter => _counter.value;
  @override
  void onInit() {
    super.onInit();
    _loadData();
  }
  @override
  void onClose() {
    // 清理资源 (Stream/Timer)
    super.onClose();
  }
  void increment() {
    _counter.value++;
  }
}
```

```

**禁止：**

- Controller 中直接操作 UI (`showDialog`)
- Controller 中使用 `BuildContext`
- Controller 循环依赖

4. 页面使用 Controller

```dart
// ✅ 方式一：GetView (推荐)
class HomePage extends GetView<HomeController> {
  @override
  Widget build(BuildContext context) {
    return Text('Count: ${controller.counter}');
  }
}
// ✅ 方式二：Obx (局部刷新)
Obx(() => Text('Count: ${controller.counter}'))
// ❌ 禁止在 build 中使用 Get.put
```

🧩 组件复用规范

提取规则

场景	提取位置	示例
全局复用 (3+ 模块)	`lib/core/widgets/`	Toast/Loading/Dialog
跨模块复用 (2 模块)	`lib/shared/widgets/`	ChangePasswordForm
模块内复用	`lib/modules/xxx/widgets/`	CashierCard
单一使用	不提取	-

命名规范

- 卡片：`XxxCard`
- 面板：`XxxPanel`
```

- 表单：`XxxForm`
- 列表项：`XxxItem`
- 布局：`XxxLayout`

🔧 全局组件使用

```dart
// Toast
Toast.success(message: '操作成功');
Toast.error(message: '操作失败');
// Loading
Loading.show(message: '加载中...');
Loading.hide();
// Dialog
AppDialog.confirm(
  context: context,
  title: '确认删除',
  content: '确定要删除吗？',
  onConfirm: () {},
);
// Drawer
CustomDrawer.show(
  context: context,
  position: DrawerPosition.right,
  width: 400.w,
  child: YourContent(),
);
```

✅ 代码检查清单

提交前必检

- [ ] 无硬编码颜色/圆角/间距/字号

- [ ] 所有尺寸使用 `.w/.h/.r/.sp`
- [ ] 按钮设置完整样式 (`foregroundColor`)
- [ ] 重复代码已提取公共组件
- [ ] Controller 生命周期正确管理
- [ ] 使用 `AppRouter/NavigationHelper` 导航
- [ ] 通过 `flutter analyze` 无警告
- [ ] 通过 `dart format .` 格式化

必备导入

```dart
import 'package:flutter/material.dart';
import 'package:flutter_screenutil/flutter_screenutil.dart';
import '../../app/theme/app_theme.dart';
```

📌 核心原则

1. **统一来源**：样式来自 `AppTheme`，路由来自 `AppRouter`，组件来自 `core/widgets`
2. **清晰分层**：app(配置) → core(基础) → modules(业务) → shared(共享)
3. **生命周期**：路由进入创建 Controller，路由离开清理 Controller
4. **禁止硬编码**：任何颜色/圆角/间距/字号都必须使用 AppTheme
5. **组件复用**：相同代码出现 2+ 次必须提取