GameDock 项目文件结构与成员分工说明

本文档用于说明 **GameDock Android 应用的目录结构设计** 以及 **三人小组的包级分工**，方便在报告中展示“软件工程化”的设计和协作方式。

1. 整体架构与目录结构概览

1.1 分层思路

项目整体采用“按**分层 + 功能**”混合组织方式，大致分为：

- **UI 层（ui/）**：界面展示、交互、导航（Jetpack Compose + Navigation）
- **Domain 层（domain/）**：核心领域模型与用例（UseCase）
- **Data 层（data/）**：远程数据源、本地数据库、仓库（Repository）与映射关系
- **后台任务（workers/）**：价格检查、提醒通知等周期性任务
- **依赖注入（di/）**：Hilt Modules，统一管理 Retrofit / Room / Repository 等依赖
- **核心工具与样式（core/）**：通用工具方法与主题设计

这样设计的好处：

- UI 改版时只需改 `ui/`，不影响数据层；
- 更换数据源（Mock → 真实 API → 后端服务）只需更新 `data/` 和 `domain/`；
- 团队分工可以按包划分责任，冲突少、边界清晰，也方便在文档里说明各自贡献。

1.2 顶层目录结构（示意）

Code (text):

```
app/
  ├─ ui/                    # 所有界面、导航和通用组件
  │   ├─ main/              # MainActivity / App 入口
  │   ├─ navigation/        # NavHost、路由定义
  │   ├─ home/              # 首页
  │   ├─ freebies/          # 免费游戏列表
  │   ├─ bundles/           # 慈善包 / 合集页面
  │   ├─ compare/           # 多平台比价页面
  │   ├─ watchlist/         # 愿望单页面
  │   ├─ detail/            # 游戏详情（含价格曲线）
  │   ├─ settings/          # 设置页面（地区 / 货币 / 提醒）
  │   └─ components/        # 通用 UI 组件（卡片、标题等）
  │
  ├─ domain/                # 领域层（业务模型 + 用例）
  │   ├─ model/             # Game / Offer / Freebie / BundleInfo / WatchItem
  │   └─ usecase/           # 用例：GetFreebies、ComparePrices、ManageWatchlist
  │
  ├─ data/                  # 数据层（远程 + 本地 + 仓库 + 映射）
  │   ├─ remote/            # Retrofit Service、DTO
  │   ├─ local/             # Room DB、Dao、Entity
  │   ├─ repo/              # Repository 接口与实现
  │   └─ mapper/            # DTO / Entity 与 Domain Model 的映射工具
  │
  ├─ workers/               # WorkManager Worker（价格同步、提醒等）
  │
  ├─ di/                    # Hilt 依赖注入模块
  │   ├─ NetworkModule.kt
  │   ├─ DatabaseModule.kt
  │   └─ RepositoryModule.kt
  │
  └─ core/                  # 核心工具与样式
      ├─ design/            # 主题、颜色、字体、间距等
      └─ util/              # 时间、货币、Result 封装等通用工具
```

2. 团队成员角色概述

为避免大家“都在同一个文件夹里挤”，并且方便老师看到清晰的责任划分，我们按 **包路径** 将责任分配给三位成员：

- **A — UI & Navigation（界面 / 导航 / 视觉）**
  - 负责 Navigation、各个 Screen 界面、通用组件和主题样式。
- **B — Data & Pricing（数据 / 聚合 / 比价逻辑）**
  - 负责数据模型、远程接口、Repository 实现以及价格 / 史低计算逻辑。
- **C — Persistence & Background（本地存储 / 愿望单 / 后台任务）**
  - 负责 Room 数据库、愿望单相关页面、后台 Worker 和通知。

在实现中，各包仍然可以互相协作，但会有一个“主要负责人”，方便管理和查阅。

3. 包级分工表（Package Ownership）

3.1 包路径与负责人一览表

> 可以直接放进报告的“Team Package Ownership”一节使用。

包路径 / 模块	主要负责人	协作方	说明
`ui/main/`	A	C	MainActivity、GameDockApp、整体 Scaffold 结构与底部导航
`ui/navigation/`	A	B, C	NavHost、路由常量与导航图；三人都会用到，由 A 统一维护
`ui/home/`	A	-	首页 UI，聚合入口卡片（Freebies / Bundles / Compare / Watchlist 等）
`ui/freebies/`	A	B	免费游戏列表 UI + ViewModel，数据来源 `DealsRepository.getFreebies()`
`ui/bundles/`	A	B	慈善包 / 合集列表 UI；B 提供 Bundles 数据与排序策略
`ui/compare/`	B	A	比价页面 UI + CompareViewModel；A 协助视觉和交互统一
`ui/watchlist/`	C	A	愿望单列表 UI + WatchlistViewModel，结合本地 Room 数据
`ui/detail/`	C	B	游戏详情页（基本信息 + 多商店价格 + 价格曲线图）

包路径 / 模块	主要负责人	协作方	说明
`ui/settings/`	A	C	设置页面：地区 / 货币 / 通知开关等；与 C 的后台逻辑联动
`ui/components/`	A	B, C	通用组件（GameCard、PriceCard、SectionHeader 等），团队复用
`domain/model/`	B	A, C	领域模型：Game、Offer、Freebie、BundleInfo、PricePoint、WatchItem 等
`domain/usecase/`	B	C	用例：GetFreebies、ComparePrices、GetBundles、ManageWatchlist 等
`data/remote/`	B	-	Retrofit 接口定义、DTO（包括 Freebies、Offers、Bundles、PriceHistory 等）
`data/local/`	C	B	Room Database、Dao、Entity；负责缓存 & 愿望单持久化
`data/repo/`	B	C	DealsRepository 接口 + 实现，组合 remote/local 得到最终数据
`data/mapper/`	B	C	DTO ↔ Domain、Entity ↔ Domain 的转换逻辑
`workers/`	C	B	WorkManager：价格同步 Worker、提醒 Worker（读取 watchlist，触发通知）
`di/`	B	A, C	Hilt Modules：NetworkModule、DatabaseModule、RepositoryModule 等
`core/util/`	B	C	时间处理、货币格式化、Result 封装等通用工具函数
`core/design/`	A	-	主题颜色、Typography、间距等 UI 设计资源

4. 各成员职责总结（可用于报告“Individual Contribution”）

4.1 成员 A — UI & Navigation

- 主要负责：
  - `ui/main/`，`ui/navigation/`，`ui/home/`，`ui/freebies/`，`ui/bundles/`，`ui/settings/`，`ui/components/`
  - App 的整体界面风格（主题、颜色、字体）
  - 导航结构与路由定义（从 Home 到 Detail / Compare / Watchlist 等）
- 目标：
  - 确保 App 在视觉上一致、交互流程顺畅；
  - 用户可以通过清晰的导航快速找到免费游戏、比价和愿望单等功能。

4.2 成员 B — Data & Pricing

- 主要负责：
  - `domain/model/`，`domain/usecase/`
  - `data/remote/`，`data/repo/`，`data/mapper/`
  - `di/NetworkModule`，`di/RepositoryModule`
  - `core/util/` 中与时间、货币、数据封装相关的工具
- 主要工作内容：
  - 设计游戏、价格、史低记录、Bundle 等数据模型；
  - 对接外部 API（或假数据），完成数据聚合；
  - 实现史低价计算、多商店去重合并等业务逻辑。
- 目标：
  - 确保 Compare、Freebies、Bundles 等页面拿到的数据正确、可扩展。

4.3 成员 C — Persistence & Background

- 主要负责：
  - `data/local/`（Room Database、Dao、Entity）
  - `ui/watchlist/`，`ui/detail/` 中与本地数据、价格历史相关的部分
  - `workers/`（价格检查、提醒通知）
  - 部分与 Watchlist / PriceHistory 相关的 `domain/usecase/`
- 主要工作内容：
  - 设计并实现愿望单与价格缓存的本地存储结构；
  - 实现 PriceCheckWorker：定期检查目标价 / 史低 / 限免并推送通知；

    - 处理离线场景（无网时使用本地缓存数据）。
- 目标：
    - 保证愿望单和提醒功能从添加、持久化到后台提醒形成闭环；
    - 提升应用在离线 / 弱网环境下的可用性。

5. 这样设计的好处（可用于“设计理由 / Rationale”）

- **层次清晰**：UI / Domain / Data / Workers / DI 分开，方便老师画出架构图。
- **分工明确**：每个包有**明确负责人**，冲突少，责任边界清楚，文书和展示时好说明。
- **便于扩展**：将来增加新平台（如 Nintendo / PlayStation）时，只需在 `data/remote + mapper + domain/model` 中扩展，UI 层改动较小。
- **测试友好**：Domain 和 Data 层可以单独做单元测试，Worker 逻辑也可以在不启动 UI 的情况下验证。
- **符合“软件工程规范”**：和课堂中讲到的多层架构、模块化、职责划分等理念相对应，方便在报告中阐述设计思路。

> 报告中可以用一句简短总结：
> “We organized the project by layer and feature, and assigned clear ownership of packages to each team member (A: UI & navigation, B: data & pricing logic, C: persistence & background workers), which reduces merge conflicts and makes responsibilities easy to understand and present.”