GameDock – Alpha Progress Summary

1. Project Overview

Project title: GameDock – Game Deals & Freebies Aggregator
Platform: Android (Kotlin + Jetpack Compose, single-activity)

The Alpha focus is not on full backend integration yet, but on:

- A clean and navigable UI shell,
- Stable screen structure and state handling,
- A data layer design that can later be connected to real APIs and Room.

2. Features Implemented at Alpha

2.1 Navigation & Screen Structure

- Single-activity Compose app with a `NavHost` and a bottom navigation bar.
- Routes configured for:
  - **Home** – intro and entry point,
  - **Freebies** – list of free games,
  - **Compare** – cross-store price comparison search,
  - **Watchlist** – tracked games overview (skeleton),
  - **Bundles** – bundle deals placeholder,
  - **Settings** – basic preferences UI.
- All screens are reachable and state survives simple configuration changes via ViewModels.

2.2 Freebies Screen (Working with Mock Data)

- `FreebiesScreen` built with Jetpack Compose (`LazyColumn` list).
- `FreebiesViewModel` exposes a simple `FreebiesUiState` :
  - `isLoading` , `games: List<Game>` , `errorMessage` .
- A `FakeDealsRepository` returns **mock freebies** (e.g. well-known PC titles) via a `GetFreebiesUseCase` .

- Simulated loading state (small delay) to imitate network latency.
- Ready to switch to real Retrofit + Room later by changing the repository implementation only.

Status: User can open the Freebies tab and see a functional list of free games based on mock data.

2.3 Compare Screen (Working with Mock Data)

- `CompareScreen` provides a text field for searching by game name.
- `CompareViewModel` holds `CompareUiState` and calls `ComparePricesUseCase`.
- The fake repository returns a list of `Offer` objects for several stores (e.g. Steam, Epic, GOG) with:
  - current price,
  - original price,
  - discount percentage,
  - an indicator of historical low (mocked).
- Results shown via a reusable `PriceCard` composable inside a `LazyColumn`.

Status: From the user's point of view, the Compare screen works end-to-end with fake data and demonstrates the price comparison concept.

2.4 Watchlist, Bundles, and Settings (Skeletons)

- **Watchlist**:
  - `WatchlistScreen` and `WatchlistViewModel` wired.
  - An in-memory watchlist model (`WatchItem`) and a simple `ManageWatchlistUseCase` using `StateFlow`.
  - UI shows basic information about tracked items (no Room persistence yet).
- **Bundles**:
  - `BundlesScreen` and `BundlesViewModel` created as a dedicated entry point for charity / bundle deals.
  - `BundleInfo` domain model defined for future integration (source, min price, games, expiry).
- **Settings**:
  - Basic settings UI with switches for things like deal alerts (currently local state).
  - Designed to be later backed by DataStore or system settings.

Status: These screens give a clear picture of the final feature set and provide placeholders where future logic will plug in.

3. Learning Outcomes & Next Steps

Learning Outcomes so far

- Built a **single-activity Jetpack Compose app** with Navigation and a bottom bar.
- Practiced **state hoisting and ViewModel + StateFlow** patterns for UI state.
- Designed a small but clear **domain layer** and **repository pattern**.
- Structured the project into UI / domain / data / DI packages, improving maintainability and team collaboration.

Planned Next Steps (towards Final)

- Replace `FakeDealsRepository` with:
  - Retrofit-based remote calls, and
  - Room-based local caching and watchlist persistence.
- Implement a `WorkManager` **PriceCheckWorker** for background alerts on watchlisted games.
- Populate the Bundles screen with at least one real or semi-real bundle data source.
- Polish UI and UX (empty states, error visuals, basic accessibility, dark mode).

This Alpha milestone mainly delivers a **solid UI shell and architecture** with mock data, so the next iterations can focus on connecting real data sources and completing the core “save money on games” experience.