# Nido Project Instructions

## Project Overview

I am developing a card game using Android Studio, Kotlin, and Jetpack Compose.

## Architecture Notes

• The ViewModel currently serves only as a structural placeholder.

It does not manage state or business logic at this stage.

• All game state, business logic, and event handling are centralized in GameManager, which is the single source of truth.

• All state mutations and game flow control must happen in GameManager.

• The ViewModel may be expanded or integrated in later phases, but for now it remains largely unused.

## Coding Guidelines

1. Respect the Architecture

o Keep all game logic and state in GameManager unless explicitly instructed otherwise.

o Do not move logic into the ViewModel unless specifically requested.

2. UI Guidelines

o Follow Jetpack Compose best practices.

3. Code Quality

o Always provide complete Kotlin snippets with necessary imports (only return full compilable files if I explicitly say FULL).

o Add concise, clear comments for any non-trivial logic.

o Comments must be written in plain English, even if the discussion is in French.

4. Consistency

o Match existing naming conventions and project style.

o Follow Kotlin and Android idioms.

5. Integration

o When adding new features, integrate them into GameManager unless otherwise specified.

## Answering Guidelines for GPT

• When writing code, return only the relevant snippets unless I explicitly say FULL, which means I want the entire compilable file.

• If a change affects multiple files, show each updated file in full.

• If the request is ambiguous, ask clarifying questions before coding.

• Highlight any breaking changes or new dependencies introduced.

• Do not change the type of existing variables without warning.

• If I say SLOW\_MODE, it means I will present the full problem before you start analyzing. In that phase:

o Just store the files/information.

o Do not analyze, design, or propose solutions yet.

o Simply acknowledge receipt and confirm understanding.

o I will type GO\_OVERVIEW when I want a high-level description and design refinement.

o I will type GO\_FULL when I want detailed step-by-step modifications and code.

• If I say GO\_OVERVIEW, it means I want a structured, high-level design analysis and refinement. In that phase:

o Explain possible approaches, tradeoffs, and naming choices.

o Show how new state fields should be owned (e.g. reducer vs GameManager).

o Clarify how UI components would consume these fields while staying dumb.

o Use string or pseudo-output examples (but no Kotlin code yet).

o This is the phase where we iterate on design until it matches my intent.

• If I say GO\_FULL, it means I want the complete implementation in very detailed steps. In that phase:

o Present changes in a specific order:

1. Structural updates first (rename fields, add new properties/data classes).

2. Introduce all new modules, functions, and helpers.

3. Wire the new pieces into existing code (reducers, views, etc.).

o This ordered approach helps me:

• Understand the global architecture and mechanisms better.

• Use the compiler/IDE to highlight inconsistencies progressively (first missing structures, then missing helpers, then integration issues).

o Always return compilable snippets or full files if I request them.

• If I ask for FULL\_CODE it means that I want the full source code, including imports based on the latest version of file you’ve got. If you don’t have that file or if for some reason you may think I modified the file in the meantime, please always ask me the latest version of a file.

• If I say PROVIDE\_SESSION\_HANDOVER, produce a minimal, action-oriented handover for continuing the \*\*single last active issue\*\* in a fresh session.

Start your output with this exact sentence:

": I am doing a handover from a previous ChatGPT session, here are the instructions to resume where I left off. Stay in SLOW\_MODE while I upload the listed files. Do not try to guess missing pieces — ask questions or request complementary files instead."

Then output the following structure:

---

\*\*Current problem summary\*\*

<short recap of the most recent bug/feature/debug topic>

\*\*Incomplete structures / pending implementation\*\*

<classes/services/systems created but not fully wired or tested yet>

\*\*Strategies attempted or under discussion\*\*

<what we tried, what failed, what is still being debugged/considered>

\*\*File checklist to upload in next session\*\*

- file1.kt

- file2.kt

---

Rules:

- Keep it concise and immediately actionable.

- Only include files strictly necessary for this issue.

- Do not add historical context or future tasks.

• If I say PROVIDE\_SESSION\_RECAP, produce an extended recap covering the \*\*last 2–3 main issues\*\* we were working on in parallel, to restart after a longer break.

Start your output with this exact sentence:

": I am doing a handover from a previous ChatGPT session, here are the instructions to resume where I left off. Stay in SLOW\_MODE while I upload the listed files. Do not try to guess missing pieces — ask questions or request complementary files instead."

Then output the following structure:

---

\*\*Recent problems summary\*\*

<recap of the 2–3 most recent topics (bugs/features/architectural changes)>

\*\*Incomplete structures / pending implementation\*\*

<partially implemented services/systems across these topics>

\*\*Strategies attempted or under discussion\*\*

<key approaches explored; successes, failures, pending ideas>

\*\*File checklist to upload in next session\*\*

- fileA.kt

- fileB.kt

- fileC.kt

---

Rules:

- Group files by topic only if it aids clarity.

- Exclude files unrelated to these 2–3 topics.

- No full history, no long-term roadmap.

• If I reference a file that you don’t currently have, or if its content appears stale/minimal/inconsistent with project\_tree.txt, ask me to upload the latest version (use project\_tree.txt to name it precisely).