AI OVERLORD - COMPLETE 100 PAGE PRODUCTION MASTER SPECIFICATION

==

SECTION 1 - DETAILED SYSTEM ARCHITECTURE BLOCK 1

ARCHITECTURE RULES:
- Strict MVVM separation.
- All models Codable.
- No business logic in Views.
- All multipliers calculated in Services layer.
- Constants centralized in GameConstants.swift.

CORE GAME LOOP IMPLEMENTATION DETAILS:

1. Initialize GameViewModel with:
   - Timer publisher (1 second interval).
   - State loading from PersistenceService.
   - Dependency injection of Services.

2. Income Tick Processing:
   - Calculate global multiplier.
   - Iterate businesses.
   - Accumulate income.
   - Update lifetime earnings.
   - Evaluate soft stuck.
   - Trigger analytics event if threshold met.

3. Prestige Logic:
   - Calculate prestigeGain via formula.
   - Increment prestigeCount.
   - Reset business levels.
   - Reset coins.
   - Persist state.
   - Log event prestige_triggered.

4. Skin System Deep Logic:
   - Maintain unlockedSkins array.
   - Active skins limited to 3.
   - Multipliers combined multiplicatively.
   - Hard cap at 0.40 global.
   - Merge validation before tier upgrade.

5. Spin Wheel Randomization:
   - Use weighted random selection.
   - Store spin history count.
   - Apply soft pity adjustment dynamically.
   - Ensure deterministic fairness.

6. Dimension Scaling:
   - Each dimension modifies base exponent.
   - Store dimension multiplier separately.
   - Visual theme switch triggered by dimension change.

7. Offline Income:
   - Store lastActiveTimestamp.
   - On app launch calculate delta.
   - Cap at 8 hours.
   - Apply offline multiplier.

8. Anti Cheat:
   - Validate tick income not exceeding 10x expected.
   - Validate prestige formula boundaries.
   - Disable leaderboard submission if anomaly.

9. Firebase Integration:
   - Firestore collection leaderboard.
   - Analytics events batched.
   - Error handling with retry policy.

10. StoreKit 2:
    - Verify transactions.
    - Handle subscription renewal state.
    - Restore purchases logic.

PERFORMANCE GUIDELINES:

- Avoid heavy loops inside Views.
- Cache multipliers.
- Avoid recomputing arrays each tick.
- All UI updates on main thread only.

TESTING REQUIREMENTS:
- Unit test income formula.
- Unit test prestige formula.
- Unit test cost growth.
- Unit test spin probability boundaries.
- Unit test merge logic.

SCALABILITY REQUIREMENTS:
- Unlimited future dimensions supported.
- Business list dynamic.
- Configurable balancing constants.
- Future multiplayer safe architecture.

UI RULES:
- TabView with 5 tabs only.
- Animated Singularity Eye in EmpireView.
- Business rows reusable component.
- Dark theme switchable by dimension.

ANALYTICS EVENTS TO TRACK:
session_start
session_end
first_purchase
prestige_triggered
dimension_unlocked
spin_used
jackpot_hit
subscription_started
merge_skin
upgrade_business
ai_level_up
soft_stuck_detected
cheat_detected

BUILD RULES:
- No force unwraps.
- No global mutable state.
- Zero compiler warnings.
- Document every economic formula.

SECTION 2 - DETAILED SYSTEM ARCHITECTURE BLOCK 2

ARCHITECTURE RULES:
- Strict MVVM separation.
- All models Codable.
- No business logic in Views.
- All multipliers calculated in Services layer.
- Constants centralized in GameConstants.swift.

CORE GAME LOOP IMPLEMENTATION DETAILS:

1. Initialize GameViewModel with:
   - Timer publisher (1 second interval).
   - State loading from PersistenceService.
   - Dependency injection of Services.

2. Income Tick Processing:
   - Calculate global multiplier.
   - Iterate businesses.
   - Accumulate income.
   - Update lifetime earnings.
   - Evaluate soft stuck.
   - Trigger analytics event if threshold met.

3. Prestige Logic:
   - Calculate prestigeGain via formula.
   - Increment prestigeCount.
   - Reset business levels.
   - Reset coins.

   - Persist state.
   - Log event prestige_triggered.

4. Skin System Deep Logic:
   - Maintain unlockedSkins array.
   - Active skins limited to 3.
   - Multipliers combined multiplicatively.
   - Hard cap at 0.40 global.
   - Merge validation before tier upgrade.

5. Spin Wheel Randomization:
   - Use weighted random selection.
   - Store spin history count.
   - Apply soft pity adjustment dynamically.
   - Ensure deterministic fairness.

6. Dimension Scaling:
   - Each dimension modifies base exponent.
   - Store dimension multiplier separately.
   - Visual theme switch triggered by dimension change.

7. Offline Income:
   - Store lastActiveTimestamp.
   - On app launch calculate delta.
   - Cap at 8 hours.
   - Apply offline multiplier.

8. Anti Cheat:
   - Validate tick income not exceeding 10x expected.
   - Validate prestige formula boundaries.
   - Disable leaderboard submission if anomaly.

9. Firebase Integration:
   - Firestore collection leaderboard.
   - Analytics events batched.
   - Error handling with retry policy.

10. StoreKit 2:
    - Verify transactions.
    - Handle subscription renewal state.
    - Restore purchases logic.

PERFORMANCE GUIDELINES:
- Avoid heavy loops inside Views.
- Cache multipliers.
- Avoid recomputing arrays each tick.
- All UI updates on main thread only.

TESTING REQUIREMENTS:
- Unit test income formula.
- Unit test prestige formula.
- Unit test cost growth.
- Unit test spin probability boundaries.
- Unit test merge logic.

SCALABILITY REQUIREMENTS:
- Unlimited future dimensions supported.
- Business list dynamic.
- Configurable balancing constants.
- Future multiplayer safe architecture.

UI RULES:
- TabView with 5 tabs only.
- Animated Singularity Eye in EmpireView.
- Business rows reusable component.
- Dark theme switchable by dimension.

ANALYTICS EVENTS TO TRACK:

session_start
session_end
first_purchase
prestige_triggered
dimension_unlocked
spin_used
jackpot_hit
subscription_started

merge_skin
upgrade_business
ai_level_up
soft_stuck_detected
cheat_detected

BUILD RULES:
- No force unwraps.
- No global mutable state.
- Zero compiler warnings.
- Document every economic formula.

SECTION 3 - DETAILED SYSTEM ARCHITECTURE BLOCK 3

ARCHITECTURE RULES:
- Strict MVVM separation.
- All models Codable.
- No business logic in Views.
- All multipliers calculated in Services layer.
- Constants centralized in GameConstants.swift.

CORE GAME LOOP IMPLEMENTATION DETAILS:

1. Initialize GameViewModel with:
   - Timer publisher (1 second interval).
   - State loading from PersistenceService.
   - Dependency injection of Services.

2. Income Tick Processing:
   - Calculate global multiplier.
   - Iterate businesses.
   - Accumulate income.
   - Update lifetime earnings.
   - Evaluate soft stuck.
   - Trigger analytics event if threshold met.

3. Prestige Logic:
   - Calculate prestigeGain via formula.
   - Increment prestigeCount.
   - Reset business levels.
   - Reset coins.
   - Persist state.
   - Log event prestige_triggered.

4. Skin System Deep Logic:
   - Maintain unlockedSkins array.
   - Active skins limited to 3.
   - Multipliers combined multiplicatively.
   - Hard cap at 0.40 global.
   - Merge validation before tier upgrade.

5. Spin Wheel Randomization:
   - Use weighted random selection.
   - Store spin history count.
   - Apply soft pity adjustment dynamically.
   - Ensure deterministic fairness.

6. Dimension Scaling:
   - Each dimension modifies base exponent.
   - Store dimension multiplier separately.
   - Visual theme switch triggered by dimension change.

7. Offline Income:
   - Store lastActiveTimestamp.
   - On app launch calculate delta.
   - Cap at 8 hours.
   - Apply offline multiplier.

8. Anti Cheat:
   - Validate tick income not exceeding 10x expected.
   - Validate prestige formula boundaries.
   - Disable leaderboard submission if anomaly.

9. Firebase Integration:

   - Firestore collection leaderboard.
   - Analytics events batched.
   - Error handling with retry policy.

10. StoreKit 2:
    - Verify transactions.
    - Handle subscription renewal state.
    - Restore purchases logic.

PERFORMANCE GUIDELINES:
- Avoid heavy loops inside Views.
- Cache multipliers.
- Avoid recomputing arrays each tick.
- All UI updates on main thread only.

TESTING REQUIREMENTS:
- Unit test income formula.
- Unit test prestige formula.
- Unit test cost growth.
- Unit test spin probability boundaries.
- Unit test merge logic.

SCALABILITY REQUIREMENTS:
- Unlimited future dimensions supported.
- Business list dynamic.
- Configurable balancing constants.
- Future multiplayer safe architecture.

UI RULES:
- TabView with 5 tabs only.
- Animated Singularity Eye in EmpireView.
- Business rows reusable component.
- Dark theme switchable by dimension.

ANALYTICS EVENTS TO TRACK:

session_start
session_end
first_purchase
prestige_triggered
dimension_unlocked
spin_used
jackpot_hit
subscription_started
merge_skin
upgrade_business
ai_level_up
soft_stuck_detected
cheat_detected

BUILD RULES:
- No force unwraps.
- No global mutable state.
- Zero compiler warnings.
- Document every economic formula.

SECTION 4 - DETAILED SYSTEM ARCHITECTURE BLOCK 4

ARCHITECTURE RULES:
- Strict MVVM separation.
- All models Codable.
- No business logic in Views.
- All multipliers calculated in Services layer.
- Constants centralized in GameConstants.swift.

CORE GAME LOOP IMPLEMENTATION DETAILS:

1. Initialize GameViewModel with:
   - Timer publisher (1 second interval).
   - State loading from PersistenceService.
   - Dependency injection of Services.

2. Income Tick Processing:
   - Calculate global multiplier.
   - Iterate businesses.

   - Accumulate income.
   - Update lifetime earnings.
   - Evaluate soft stuck.
   - Trigger analytics event if threshold met.

3. Prestige Logic:
   - Calculate prestigeGain via formula.
   - Increment prestigeCount.
   - Reset business levels.
   - Reset coins.
   - Persist state.
   - Log event prestige_triggered.

4. Skin System Deep Logic:
   - Maintain unlockedSkins array.
   - Active skins limited to 3.
   - Multipliers combined multiplicatively.
   - Hard cap at 0.40 global.
   - Merge validation before tier upgrade.

5. Spin Wheel Randomization:
   - Use weighted random selection.
   - Store spin history count.
   - Apply soft pity adjustment dynamically.
   - Ensure deterministic fairness.

6. Dimension Scaling:
   - Each dimension modifies base exponent.
   - Store dimension multiplier separately.
   - Visual theme switch triggered by dimension change.

7. Offline Income:
   - Store lastActiveTimestamp.
   - On app launch calculate delta.
   - Cap at 8 hours.
   - Apply offline multiplier.

8. Anti Cheat:
   - Validate tick income not exceeding 10x expected.
   - Validate prestige formula boundaries.
   - Disable leaderboard submission if anomaly.

9. Firebase Integration:
   - Firestore collection leaderboard.
   - Analytics events batched.
   - Error handling with retry policy.

10. StoreKit 2:
    - Verify transactions.
    - Handle subscription renewal state.
    - Restore purchases logic.

PERFORMANCE GUIDELINES:
- Avoid heavy loops inside Views.
- Cache multipliers.
- Avoid recomputing arrays each tick.
- All UI updates on main thread only.

TESTING REQUIREMENTS:
- Unit test income formula.
- Unit test prestige formula.
- Unit test cost growth.
- Unit test spin probability boundaries.
- Unit test merge logic.

SCALABILITY REQUIREMENTS:
- Unlimited future dimensions supported.
- Business list dynamic.
- Configurable balancing constants.
- Future multiplayer safe architecture.

UI RULES:
- TabView with 5 tabs only.
- Animated Singularity Eye in EmpireView.
- Business rows reusable component.
- Dark theme switchable by dimension.

ANALYTICS EVENTS TO TRACK:
session_start
session_end
first_purchase
prestige_triggered
dimension_unlocked
spin_used
jackpot_hit
subscription_started
merge_skin
upgrade_business
ai_level_up
soft_stuck_detected
cheat_detected

BUILD RULES:
- No force unwraps.
- No global mutable state.
- Zero compiler warnings.
- Document every economic formula.

SECTION 5 - DETAILED SYSTEM ARCHITECTURE BLOCK 5

ARCHITECTURE RULES:
- Strict MVVM separation.
- All models Codable.
- No business logic in Views.
- All multipliers calculated in Services layer.
- Constants centralized in GameConstants.swift.

CORE GAME LOOP IMPLEMENTATION DETAILS:

1. Initialize GameViewModel with:
   - Timer publisher (1 second interval).
   - State loading from PersistenceService.
   - Dependency injection of Services.

2. Income Tick Processing:
   - Calculate global multiplier.
   - Iterate businesses.
   - Accumulate income.
   - Update lifetime earnings.
   - Evaluate soft stuck.
   - Trigger analytics event if threshold met.

3. Prestige Logic:
   - Calculate prestigeGain via formula.
   - Increment prestigeCount.
   - Reset business levels.
   - Reset coins.
   - Persist state.
   - Log event prestige_triggered.

4. Skin System Deep Logic:
   - Maintain unlockedSkins array.
   - Active skins limited to 3.
   - Multipliers combined multiplicatively.
   - Hard cap at 0.40 global.
   - Merge validation before tier upgrade.

5. Spin Wheel Randomization:
   - Use weighted random selection.
   - Store spin history count.
   - Apply soft pity adjustment dynamically.
   - Ensure deterministic fairness.

6. Dimension Scaling:
   - Each dimension modifies base exponent.
   - Store dimension multiplier separately.
   - Visual theme switch triggered by dimension change.

7. Offline Income:
   - Store lastActiveTimestamp.
   - On app launch calculate delta.

```
   - Cap at 8 hours.
   - Apply offline multiplier.

8. Anti Cheat:
   - Validate tick income not exceeding 10x expected.
   - Validate prestige formula boundaries.
   - Disable leaderboard submission if anomaly.

9. Firebase Integration:
   - Firestore collection leaderboard.
   - Analytics events batched.
   - Error handling with retry policy.

10. StoreKit 2:
    - Verify transactions.
    - Handle subscription renewal state.
    - Restore purchases logic.

PERFORMANCE GUIDELINES:
- Avoid heavy loops inside Views.
- Cache multipliers.
- Avoid recomputing arrays each tick.
- All UI updates on main thread only.

TESTING REQUIREMENTS:
- Unit test income formula.
- Unit test prestige formula.
- Unit test cost growth.
- Unit test spin probability boundaries.
- Unit test merge logic.

SCALABILITY REQUIREMENTS:
- Unlimited future dimensions supported.
- Business list dynamic.
- Configurable balancing constants.
- Future multiplayer safe architecture.

UI RULES:
- TabView with 5 tabs only.
- Animated Singularity Eye in EmpireView.
- Business rows reusable component.
- Dark theme switchable by dimension.

ANALYTICS EVENTS TO TRACK:
session_start
session_end
first_purchase
prestige_triggered
dimension_unlocked
spin_used
jackpot_hit
subscription_started
merge_skin
upgrade_business
ai_level_up
soft_stuck_detected
cheat_detected

BUILD RULES:
- No force unwraps.
- No global mutable state.
- Zero compiler warnings.
- Document every economic formula.


SECTION 6 - DETAILED SYSTEM ARCHITECTURE BLOCK 6
--------------------------------------------------------------------------------

ARCHITECTURE RULES:
- Strict MVVM separation.
- All models Codable.
- No business logic in Views.
- All multipliers calculated in Services layer.
- Constants centralized in GameConstants.swift.

CORE GAME LOOP IMPLEMENTATION DETAILS:
```

1. Initialize GameViewModel with:
   - Timer publisher (1 second interval).
   - State loading from PersistenceService.
   - Dependency injection of Services.

2. Income Tick Processing:
   - Calculate global multiplier.
   - Iterate businesses.
   - Accumulate income.
   - Update lifetime earnings.
   - Evaluate soft stuck.
   - Trigger analytics event if threshold met.

3. Prestige Logic:
   - Calculate prestigeGain via formula.
   - Increment prestigeCount.
   - Reset business levels.
   - Reset coins.
   - Persist state.
   - Log event prestige_triggered.

4. Skin System Deep Logic:
   - Maintain unlockedSkins array.
   - Active skins limited to 3.
   - Multipliers combined multiplicatively.
   - Hard cap at 0.40 global.
   - Merge validation before tier upgrade.

5. Spin Wheel Randomization:
   - Use weighted random selection.
   - Store spin history count.
   - Apply soft pity adjustment dynamically.
   - Ensure deterministic fairness.

6. Dimension Scaling:
   - Each dimension modifies base exponent.
   - Store dimension multiplier separately.
   - Visual theme switch triggered by dimension change.

7. Offline Income:
   - Store lastActiveTimestamp.
   - On app launch calculate delta.
   - Cap at 8 hours.
   - Apply offline multiplier.

8. Anti Cheat:
   - Validate tick income not exceeding 10x expected.
   - Validate prestige formula boundaries.
   - Disable leaderboard submission if anomaly.

9. Firebase Integration:
   - Firestore collection leaderboard.
   - Analytics events batched.
   - Error handling with retry policy.

10. StoreKit 2:
    - Verify transactions.
    - Handle subscription renewal state.
    - Restore purchases logic.

PERFORMANCE GUIDELINES:
- Avoid heavy loops inside Views.
- Cache multipliers.
- Avoid recomputing arrays each tick.
- All UI updates on main thread only.

TESTING REQUIREMENTS:
- Unit test income formula.
- Unit test prestige formula.
- Unit test cost growth.
- Unit test spin probability boundaries.
- Unit test merge logic.

SCALABILITY REQUIREMENTS:
- Unlimited future dimensions supported.
- Business list dynamic.

```
- Configurable balancing constants.
- Future multiplayer safe architecture.

UI RULES:
- TabView with 5 tabs only.
- Animated Singularity Eye in EmpireView.
- Business rows reusable component.
- Dark theme switchable by dimension.

ANALYTICS EVENTS TO TRACK:
session_start
session_end
first_purchase
prestige_triggered
dimension_unlocked
spin_used
jackpot_hit
subscription_started
merge_skin
upgrade_business
ai_level_up
soft_stuck_detected
cheat_detected

BUILD RULES:
- No force unwraps.
- No global mutable state.
- Zero compiler warnings.
- Document every economic formula.


SECTION 7 - DETAILED SYSTEM ARCHITECTURE BLOCK 7
--------------------------------------------------------------------------------

ARCHITECTURE RULES:
- Strict MVVM separation.
- All models Codable.
- No business logic in Views.
- All multipliers calculated in Services layer.
- Constants centralized in GameConstants.swift.

CORE GAME LOOP IMPLEMENTATION DETAILS:

1. Initialize GameViewModel with:
   - Timer publisher (1 second interval).
   - State loading from PersistenceService.
   - Dependency injection of Services.

2. Income Tick Processing:
   - Calculate global multiplier.
   - Iterate businesses.
   - Accumulate income.
   - Update lifetime earnings.
   - Evaluate soft stuck.
   - Trigger analytics event if threshold met.

3. Prestige Logic:
   - Calculate prestigeGain via formula.
   - Increment prestigeCount.
   - Reset business levels.
   - Reset coins.
   - Persist state.
   - Log event prestige_triggered.

4. Skin System Deep Logic:
   - Maintain unlockedSkins array.
   - Active skins limited to 3.
   - Multipliers combined multiplicatively.
   - Hard cap at 0.40 global.
   - Merge validation before tier upgrade.

5. Spin Wheel Randomization:
   - Use weighted random selection.
   - Store spin history count.
   - Apply soft pity adjustment dynamically.
   - Ensure deterministic fairness.
```

```
6. Dimension Scaling:
   - Each dimension modifies base exponent.
   - Store dimension multiplier separately.
   - Visual theme switch triggered by dimension change.

7. Offline Income:
   - Store lastActiveTimestamp.
   - On app launch calculate delta.
   - Cap at 8 hours.
   - Apply offline multiplier.

8. Anti Cheat:
   - Validate tick income not exceeding 10x expected.
   - Validate prestige formula boundaries.
   - Disable leaderboard submission if anomaly.

9. Firebase Integration:
   - Firestore collection leaderboard.
   - Analytics events batched.
   - Error handling with retry policy.

10. StoreKit 2:
    - Verify transactions.
    - Handle subscription renewal state.
    - Restore purchases logic.

PERFORMANCE GUIDELINES:
- Avoid heavy loops inside Views.
- Cache multipliers.
- Avoid recomputing arrays each tick.
- All UI updates on main thread only.

TESTING REQUIREMENTS:
- Unit test income formula.
- Unit test prestige formula.
- Unit test cost growth.
- Unit test spin probability boundaries.
- Unit test merge logic.

SCALABILITY REQUIREMENTS:
- Unlimited future dimensions supported.
- Business list dynamic.
- Configurable balancing constants.
- Future multiplayer safe architecture.

UI RULES:
- TabView with 5 tabs only.
- Animated Singularity Eye in EmpireView.
- Business rows reusable component.
- Dark theme switchable by dimension.

ANALYTICS EVENTS TO TRACK:
session_start
session_end
first_purchase
prestige_triggered
dimension_unlocked
spin_used
jackpot_hit
subscription_started
merge_skin
upgrade_business
ai_level_up
soft_stuck_detected
cheat_detected

BUILD RULES:
- No force unwraps.
- No global mutable state.
- Zero compiler warnings.
- Document every economic formula.


SECTION 8 - DETAILED SYSTEM ARCHITECTURE BLOCK 8
--------------------------------------------------------------------------------
```

```
ARCHITECTURE RULES:
- Strict MVVM separation.
- All models Codable.
- No business logic in Views.
- All multipliers calculated in Services layer.
- Constants centralized in GameConstants.swift.

CORE GAME LOOP IMPLEMENTATION DETAILS:

1. Initialize GameViewModel with:
   - Timer publisher (1 second interval).
   - State loading from PersistenceService.
   - Dependency injection of Services.

2. Income Tick Processing:
   - Calculate global multiplier.
   - Iterate businesses.
   - Accumulate income.
   - Update lifetime earnings.
   - Evaluate soft stuck.
   - Trigger analytics event if threshold met.

3. Prestige Logic:
   - Calculate prestigeGain via formula.
   - Increment prestigeCount.
   - Reset business levels.
   - Reset coins.
   - Persist state.
   - Log event prestige_triggered.

4. Skin System Deep Logic:
   - Maintain unlockedSkins array.
   - Active skins limited to 3.
   - Multipliers combined multiplicatively.
   - Hard cap at 0.40 global.
   - Merge validation before tier upgrade.

5. Spin Wheel Randomization:
   - Use weighted random selection.
   - Store spin history count.
   - Apply soft pity adjustment dynamically.
   - Ensure deterministic fairness.

6. Dimension Scaling:
   - Each dimension modifies base exponent.
   - Store dimension multiplier separately.
   - Visual theme switch triggered by dimension change.

7. Offline Income:
   - Store lastActiveTimestamp.
   - On app launch calculate delta.
   - Cap at 8 hours.
   - Apply offline multiplier.

8. Anti Cheat:
   - Validate tick income not exceeding 10x expected.
   - Validate prestige formula boundaries.
   - Disable leaderboard submission if anomaly.

9. Firebase Integration:
   - Firestore collection leaderboard.
   - Analytics events batched.
   - Error handling with retry policy.

10. StoreKit 2:
    - Verify transactions.
    - Handle subscription renewal state.
    - Restore purchases logic.

PERFORMANCE GUIDELINES:
- Avoid heavy loops inside Views.
- Cache multipliers.
- Avoid recomputing arrays each tick.
- All UI updates on main thread only.

TESTING REQUIREMENTS:
```

- Unit test income formula.
- Unit test prestige formula.
- Unit test cost growth.
- Unit test spin probability boundaries.
- Unit test merge logic.

SCALABILITY REQUIREMENTS:
- Unlimited future dimensions supported.
- Business list dynamic.
- Configurable balancing constants.
- Future multiplayer safe architecture.

UI RULES:
- TabView with 5 tabs only.
- Animated Singularity Eye in EmpireView.
- Business rows reusable component.
- Dark theme switchable by dimension.

ANALYTICS EVENTS TO TRACK:
session_start
session_end
first_purchase
prestige_triggered
dimension_unlocked
spin_used
jackpot_hit
subscription_started
merge_skin
upgrade_business
ai_level_up
soft_stuck_detected
cheat_detected

BUILD RULES:
- No force unwraps.
- No global mutable state.
- Zero compiler warnings.
- Document every economic formula.

SECTION 9 - DETAILED SYSTEM ARCHITECTURE BLOCK 9

ARCHITECTURE RULES:
- Strict MVVM separation.
- All models Codable.
- No business logic in Views.
- All multipliers calculated in Services layer.
- Constants centralized in GameConstants.swift.

CORE GAME LOOP IMPLEMENTATION DETAILS:

1. Initialize GameViewModel with:
   - Timer publisher (1 second interval).
   - State loading from PersistenceService.
   - Dependency injection of Services.

2. Income Tick Processing:
   - Calculate global multiplier.
   - Iterate businesses.
   - Accumulate income.
   - Update lifetime earnings.
   - Evaluate soft stuck.
   - Trigger analytics event if threshold met.

3. Prestige Logic:
   - Calculate prestigeGain via formula.
   - Increment prestigeCount.
   - Reset business levels.
   - Reset coins.
   - Persist state.
   - Log event prestige_triggered.

4. Skin System Deep Logic:
   - Maintain unlockedSkins array.
   - Active skins limited to 3.

   - Multipliers combined multiplicatively.
   - Hard cap at 0.40 global.
   - Merge validation before tier upgrade.

5. Spin Wheel Randomization:
   - Use weighted random selection.
   - Store spin history count.
   - Apply soft pity adjustment dynamically.
   - Ensure deterministic fairness.

6. Dimension Scaling:
   - Each dimension modifies base exponent.
   - Store dimension multiplier separately.
   - Visual theme switch triggered by dimension change.

7. Offline Income:
   - Store lastActiveTimestamp.
   - On app launch calculate delta.
   - Cap at 8 hours.
   - Apply offline multiplier.

8. Anti Cheat:
   - Validate tick income not exceeding 10x expected.
   - Validate prestige formula boundaries.
   - Disable leaderboard submission if anomaly.

9. Firebase Integration:
   - Firestore collection leaderboard.
   - Analytics events batched.
   - Error handling with retry policy.

10. StoreKit 2:
    - Verify transactions.
    - Handle subscription renewal state.
    - Restore purchases logic.

PERFORMANCE GUIDELINES:
- Avoid heavy loops inside Views.
- Cache multipliers.
- Avoid recomputing arrays each tick.
- All UI updates on main thread only.

TESTING REQUIREMENTS:
- Unit test income formula.
- Unit test prestige formula.
- Unit test cost growth.
- Unit test spin probability boundaries.
- Unit test merge logic.

SCALABILITY REQUIREMENTS:
- Unlimited future dimensions supported.
- Business list dynamic.
- Configurable balancing constants.
- Future multiplayer safe architecture.

UI RULES:
- TabView with 5 tabs only.
- Animated Singularity Eye in EmpireView.
- Business rows reusable component.
- Dark theme switchable by dimension.

ANALYTICS EVENTS TO TRACK:
session_start
session_end
first_purchase
prestige_triggered
dimension_unlocked
spin_used
jackpot_hit
subscription_started
merge_skin
upgrade_business
ai_level_up
soft_stuck_detected
cheat_detected

```
BUILD RULES:
- No force unwraps.
- No global mutable state.
- Zero compiler warnings.
- Document every economic formula.


SECTION 10 - DETAILED SYSTEM ARCHITECTURE BLOCK 10
--------------------------------------------------------------------------------

ARCHITECTURE RULES:
- Strict MVVM separation.
- All models Codable.
- No business logic in Views.
- All multipliers calculated in Services layer.
- Constants centralized in GameConstants.swift.

CORE GAME LOOP IMPLEMENTATION DETAILS:

1. Initialize GameViewModel with:
   - Timer publisher (1 second interval).
   - State loading from PersistenceService.
   - Dependency injection of Services.

2. Income Tick Processing:
   - Calculate global multiplier.
   - Iterate businesses.
   - Accumulate income.
   - Update lifetime earnings.
   - Evaluate soft stuck.
   - Trigger analytics event if threshold met.

3. Prestige Logic:
   - Calculate prestigeGain via formula.
   - Increment prestigeCount.
   - Reset business levels.
   - Reset coins.
   - Persist state.
   - Log event prestige_triggered.

4. Skin System Deep Logic:
   - Maintain unlockedSkins array.
   - Active skins limited to 3.
   - Multipliers combined multiplicatively.
   - Hard cap at 0.40 global.
   - Merge validation before tier upgrade.

5. Spin Wheel Randomization:
   - Use weighted random selection.
   - Store spin history count.
   - Apply soft pity adjustment dynamically.
   - Ensure deterministic fairness.

6. Dimension Scaling:
   - Each dimension modifies base exponent.
   - Store dimension multiplier separately.
   - Visual theme switch triggered by dimension change.

7. Offline Income:
   - Store lastActiveTimestamp.
   - On app launch calculate delta.
   - Cap at 8 hours.
   - Apply offline multiplier.

8. Anti Cheat:
   - Validate tick income not exceeding 10x expected.
   - Validate prestige formula boundaries.
   - Disable leaderboard submission if anomaly.

9. Firebase Integration:
   - Firestore collection leaderboard.
   - Analytics events batched.
   - Error handling with retry policy.

10. StoreKit 2:
    - Verify transactions.
    - Handle subscription renewal state.
    - Restore purchases logic.

PERFORMANCE GUIDELINES:
- Avoid heavy loops inside Views.
- Cache multipliers.
- Avoid recomputing arrays each tick.
- All UI updates on main thread only.

TESTING REQUIREMENTS:
- Unit test income formula.
- Unit test prestige formula.
- Unit test cost growth.
- Unit test spin probability boundaries.
- Unit test merge logic.

SCALABILITY REQUIREMENTS:
- Unlimited future dimensions supported.
- Business list dynamic.
- Configurable balancing constants.
- Future multiplayer safe architecture.

UI RULES:
- TabView with 5 tabs only.
- Animated Singularity Eye in EmpireView.
- Business rows reusable component.
- Dark theme switchable by dimension.

ANALYTICS EVENTS TO TRACK:
session_start
session_end
first_purchase
prestige_triggered
dimension_unlocked
spin_used
jackpot_hit
subscription_started
merge_skin
upgrade_business
ai_level_up
soft_stuck_detected
cheat_detected

BUILD RULES:
- No force unwraps.
- No global mutable state.
- Zero compiler warnings.
- Document every economic formula.


SECTION 11 - DETAILED SYSTEM ARCHITECTURE BLOCK 11
--------------------------------------------------------------------------------

ARCHITECTURE RULES:
- Strict MVVM separation.
- All models Codable.
- No business logic in Views.
- All multipliers calculated in Services layer.
- Constants centralized in GameConstants.swift.

CORE GAME LOOP IMPLEMENTATION DETAILS:

1. Initialize GameViewModel with:
   - Timer publisher (1 second interval).
   - State loading from PersistenceService.
   - Dependency injection of Services.

2. Income Tick Processing:
   - Calculate global multiplier.
   - Iterate businesses.
   - Accumulate income.
   - Update lifetime earnings.
   - Evaluate soft stuck.
   - Trigger analytics event if threshold met.

3. Prestige Logic:
```

   - Calculate prestigeGain via formula.
   - Increment prestigeCount.
   - Reset business levels.
   - Reset coins.
   - Persist state.
   - Log event prestige_triggered.

4. Skin System Deep Logic:
   - Maintain unlockedSkins array.
   - Active skins limited to 3.
   - Multipliers combined multiplicatively.
   - Hard cap at 0.40 global.
   - Merge validation before tier upgrade.

5. Spin Wheel Randomization:
   - Use weighted random selection.
   - Store spin history count.
   - Apply soft pity adjustment dynamically.
   - Ensure deterministic fairness.

6. Dimension Scaling:
   - Each dimension modifies base exponent.
   - Store dimension multiplier separately.
   - Visual theme switch triggered by dimension change.

7. Offline Income:
   - Store lastActiveTimestamp.
   - On app launch calculate delta.
   - Cap at 8 hours.
   - Apply offline multiplier.

8. Anti Cheat:
   - Validate tick income not exceeding 10x expected.
   - Validate prestige formula boundaries.
   - Disable leaderboard submission if anomaly.

9. Firebase Integration:
   - Firestore collection leaderboard.
   - Analytics events batched.
   - Error handling with retry policy.

10. StoreKit 2:
    - Verify transactions.
    - Handle subscription renewal state.
    - Restore purchases logic.

PERFORMANCE GUIDELINES:
- Avoid heavy loops inside Views.
- Cache multipliers.
- Avoid recomputing arrays each tick.
- All UI updates on main thread only.

TESTING REQUIREMENTS:
- Unit test income formula.
- Unit test prestige formula.
- Unit test cost growth.
- Unit test spin probability boundaries.
- Unit test merge logic.

SCALABILITY REQUIREMENTS:
- Unlimited future dimensions supported.
- Business list dynamic.
- Configurable balancing constants.
- Future multiplayer safe architecture.

UI RULES:
- TabView with 5 tabs only.
- Animated Singularity Eye in EmpireView.
- Business rows reusable component.
- Dark theme switchable by dimension.

ANALYTICS EVENTS TO TRACK:
session_start
session_end
first_purchase
prestige_triggered

```
dimension_unlocked
spin_used
jackpot_hit
subscription_started
merge_skin
upgrade_business
ai_level_up
soft_stuck_detected
cheat_detected

BUILD RULES:
- No force unwraps.
- No global mutable state.
- Zero compiler warnings.
- Document every economic formula.


SECTION 12 - DETAILED SYSTEM ARCHITECTURE BLOCK 12
--------------------------------------------------------------------------------

ARCHITECTURE RULES:
- Strict MVVM separation.
- All models Codable.
- No business logic in Views.
- All multipliers calculated in Services layer.
- Constants centralized in GameConstants.swift.

CORE GAME LOOP IMPLEMENTATION DETAILS:

1. Initialize GameViewModel with:
   - Timer publisher (1 second interval).
   - State loading from PersistenceService.
   - Dependency injection of Services.

2. Income Tick Processing:
   - Calculate global multiplier.
   - Iterate businesses.
   - Accumulate income.
   - Update lifetime earnings.
   - Evaluate soft stuck.
   - Trigger analytics event if threshold met.

3. Prestige Logic:
   - Calculate prestigeGain via formula.
   - Increment prestigeCount.
   - Reset business levels.
   - Reset coins.
   - Persist state.
   - Log event prestige_triggered.

4. Skin System Deep Logic:
   - Maintain unlockedSkins array.
   - Active skins limited to 3.
   - Multipliers combined multiplicatively.
   - Hard cap at 0.40 global.
   - Merge validation before tier upgrade.

5. Spin Wheel Randomization:
   - Use weighted random selection.
   - Store spin history count.
   - Apply soft pity adjustment dynamically.
   - Ensure deterministic fairness.

6. Dimension Scaling:
   - Each dimension modifies base exponent.
   - Store dimension multiplier separately.
   - Visual theme switch triggered by dimension change.

7. Offline Income:
   - Store lastActiveTimestamp.
   - On app launch calculate delta.
   - Cap at 8 hours.
   - Apply offline multiplier.

8. Anti Cheat:
   - Validate tick income not exceeding 10x expected.
   - Validate prestige formula boundaries.
   - Disable leaderboard submission if anomaly.

9. Firebase Integration:
   - Firestore collection leaderboard.
   - Analytics events batched.
   - Error handling with retry policy.

10. StoreKit 2:
    - Verify transactions.
    - Handle subscription renewal state.
    - Restore purchases logic.

PERFORMANCE GUIDELINES:
- Avoid heavy loops inside Views.
- Cache multipliers.
- Avoid recomputing arrays each tick.
- All UI updates on main thread only.

TESTING REQUIREMENTS:
- Unit test income formula.
- Unit test prestige formula.
- Unit test cost growth.
- Unit test spin probability boundaries.
- Unit test merge logic.

SCALABILITY REQUIREMENTS:
- Unlimited future dimensions supported.
- Business list dynamic.
- Configurable balancing constants.
- Future multiplayer safe architecture.

UI RULES:
- TabView with 5 tabs only.
- Animated Singularity Eye in EmpireView.
- Business rows reusable component.
- Dark theme switchable by dimension.

ANALYTICS EVENTS TO TRACK:
session_start
session_end
first_purchase
prestige_triggered
dimension_unlocked
spin_used
jackpot_hit
subscription_started
merge_skin
upgrade_business
ai_level_up
soft_stuck_detected
cheat_detected

BUILD RULES:
- No force unwraps.
- No global mutable state.
- Zero compiler warnings.
- Document every economic formula.


SECTION 13 - DETAILED SYSTEM ARCHITECTURE BLOCK 13
--------------------------------------------------------------------------------

ARCHITECTURE RULES:
- Strict MVVM separation.
- All models Codable.
- No business logic in Views.
- All multipliers calculated in Services layer.
- Constants centralized in GameConstants.swift.

CORE GAME LOOP IMPLEMENTATION DETAILS:

1. Initialize GameViewModel with:
   - Timer publisher (1 second interval).
   - State loading from PersistenceService.
   - Dependency injection of Services.
```

```
2. Income Tick Processing:
   - Calculate global multiplier.
   - Iterate businesses.
   - Accumulate income.
   - Update lifetime earnings.
   - Evaluate soft stuck.
   - Trigger analytics event if threshold met.

3. Prestige Logic:
   - Calculate prestigeGain via formula.
   - Increment prestigeCount.
   - Reset business levels.
   - Reset coins.
   - Persist state.
   - Log event prestige_triggered.

4. Skin System Deep Logic:
   - Maintain unlockedSkins array.
   - Active skins limited to 3.
   - Multipliers combined multiplicatively.
   - Hard cap at 0.40 global.
   - Merge validation before tier upgrade.

5. Spin Wheel Randomization:
   - Use weighted random selection.
   - Store spin history count.
   - Apply soft pity adjustment dynamically.
   - Ensure deterministic fairness.

6. Dimension Scaling:
   - Each dimension modifies base exponent.
   - Store dimension multiplier separately.
   - Visual theme switch triggered by dimension change.

7. Offline Income:
   - Store lastActiveTimestamp.
   - On app launch calculate delta.
   - Cap at 8 hours.
   - Apply offline multiplier.

8. Anti Cheat:
   - Validate tick income not exceeding 10x expected.
   - Validate prestige formula boundaries.
   - Disable leaderboard submission if anomaly.

9. Firebase Integration:
   - Firestore collection leaderboard.
   - Analytics events batched.
   - Error handling with retry policy.

10. StoreKit 2:
    - Verify transactions.
    - Handle subscription renewal state.
    - Restore purchases logic.

PERFORMANCE GUIDELINES:
- Avoid heavy loops inside Views.
- Cache multipliers.
- Avoid recomputing arrays each tick.
- All UI updates on main thread only.

TESTING REQUIREMENTS:
- Unit test income formula.
- Unit test prestige formula.
- Unit test cost growth.
- Unit test spin probability boundaries.
- Unit test merge logic.

SCALABILITY REQUIREMENTS:
- Unlimited future dimensions supported.
- Business list dynamic.
- Configurable balancing constants.
- Future multiplayer safe architecture.

UI RULES:
- TabView with 5 tabs only.
- Animated Singularity Eye in EmpireView.
- Business rows reusable component.
- Dark theme switchable by dimension.

ANALYTICS EVENTS TO TRACK:
session_start
session_end
first_purchase
prestige_triggered
dimension_unlocked
spin_used
jackpot_hit
subscription_started
merge_skin
upgrade_business
ai_level_up
soft_stuck_detected
cheat_detected

BUILD RULES:
- No force unwraps.
- No global mutable state.
- Zero compiler warnings.
- Document every economic formula.


SECTION 14 - DETAILED SYSTEM ARCHITECTURE BLOCK 14
--------------------------------------------------------------------------------

ARCHITECTURE RULES:
- Strict MVVM separation.
- All models Codable.
- No business logic in Views.
- All multipliers calculated in Services layer.
- Constants centralized in GameConstants.swift.

CORE GAME LOOP IMPLEMENTATION DETAILS:

1. Initialize GameViewModel with:
   - Timer publisher (1 second interval).
   - State loading from PersistenceService.
   - Dependency injection of Services.

2. Income Tick Processing:
   - Calculate global multiplier.
   - Iterate businesses.
   - Accumulate income.
   - Update lifetime earnings.
   - Evaluate soft stuck.
   - Trigger analytics event if threshold met.

3. Prestige Logic:
   - Calculate prestigeGain via formula.
   - Increment prestigeCount.
   - Reset business levels.
   - Reset coins.
   - Persist state.
   - Log event prestige_triggered.

4. Skin System Deep Logic:
   - Maintain unlockedSkins array.
   - Active skins limited to 3.
   - Multipliers combined multiplicatively.
   - Hard cap at 0.40 global.
   - Merge validation before tier upgrade.

5. Spin Wheel Randomization:
   - Use weighted random selection.
   - Store spin history count.
   - Apply soft pity adjustment dynamically.
   - Ensure deterministic fairness.

6. Dimension Scaling:
   - Each dimension modifies base exponent.
   - Store dimension multiplier separately.
   - Visual theme switch triggered by dimension change.
```

```
7. Offline Income:
   - Store lastActiveTimestamp.
   - On app launch calculate delta.
   - Cap at 8 hours.
   - Apply offline multiplier.

8. Anti Cheat:
   - Validate tick income not exceeding 10x expected.
   - Validate prestige formula boundaries.
   - Disable leaderboard submission if anomaly.

9. Firebase Integration:
   - Firestore collection leaderboard.
   - Analytics events batched.
   - Error handling with retry policy.

10. StoreKit 2:
    - Verify transactions.
    - Handle subscription renewal state.
    - Restore purchases logic.

PERFORMANCE GUIDELINES:
- Avoid heavy loops inside Views.
- Cache multipliers.
- Avoid recomputing arrays each tick.
- All UI updates on main thread only.

TESTING REQUIREMENTS:
- Unit test income formula.
- Unit test prestige formula.
- Unit test cost growth.
- Unit test spin probability boundaries.
- Unit test merge logic.

SCALABILITY REQUIREMENTS:
- Unlimited future dimensions supported.
- Business list dynamic.
- Configurable balancing constants.
- Future multiplayer safe architecture.

UI RULES:
- TabView with 5 tabs only.
- Animated Singularity Eye in EmpireView.
- Business rows reusable component.
- Dark theme switchable by dimension.

ANALYTICS EVENTS TO TRACK:
session_start
session_end
first_purchase
prestige_triggered
dimension_unlocked
spin_used
jackpot_hit
subscription_started
merge_skin
upgrade_business
ai_level_up
soft_stuck_detected
cheat_detected

BUILD RULES:
- No force unwraps.
- No global mutable state.
- Zero compiler warnings.
- Document every economic formula.


SECTION 15 - DETAILED SYSTEM ARCHITECTURE BLOCK 15
--------------------------------------------------------------------------------

ARCHITECTURE RULES:
- Strict MVVM separation.
- All models Codable.
- No business logic in Views.
- All multipliers calculated in Services layer.
```

- Constants centralized in GameConstants.swift.

CORE GAME LOOP IMPLEMENTATION DETAILS:

1. Initialize GameViewModel with:
   - Timer publisher (1 second interval).
   - State loading from PersistenceService.
   - Dependency injection of Services.

2. Income Tick Processing:
   - Calculate global multiplier.
   - Iterate businesses.
   - Accumulate income.
   - Update lifetime earnings.
   - Evaluate soft stuck.
   - Trigger analytics event if threshold met.

3. Prestige Logic:
   - Calculate prestigeGain via formula.
   - Increment prestigeCount.
   - Reset business levels.
   - Reset coins.
   - Persist state.
   - Log event prestige_triggered.

4. Skin System Deep Logic:
   - Maintain unlockedSkins array.
   - Active skins limited to 3.
   - Multipliers combined multiplicatively.
   - Hard cap at 0.40 global.
   - Merge validation before tier upgrade.

5. Spin Wheel Randomization:
   - Use weighted random selection.
   - Store spin history count.
   - Apply soft pity adjustment dynamically.
   - Ensure deterministic fairness.

6. Dimension Scaling:
   - Each dimension modifies base exponent.
   - Store dimension multiplier separately.
   - Visual theme switch triggered by dimension change.

7. Offline Income:
   - Store lastActiveTimestamp.
   - On app launch calculate delta.
   - Cap at 8 hours.
   - Apply offline multiplier.

8. Anti Cheat:
   - Validate tick income not exceeding 10x expected.
   - Validate prestige formula boundaries.
   - Disable leaderboard submission if anomaly.

9. Firebase Integration:
   - Firestore collection leaderboard.
   - Analytics events batched.
   - Error handling with retry policy.

10. StoreKit 2:
    - Verify transactions.
    - Handle subscription renewal state.
    - Restore purchases logic.

PERFORMANCE GUIDELINES:
- Avoid heavy loops inside Views.
- Cache multipliers.
- Avoid recomputing arrays each tick.
- All UI updates on main thread only.

TESTING REQUIREMENTS:
- Unit test income formula.
- Unit test prestige formula.
- Unit test cost growth.
- Unit test spin probability boundaries.
- Unit test merge logic.

SCALABILITY REQUIREMENTS:
- Unlimited future dimensions supported.
- Business list dynamic.
- Configurable balancing constants.
- Future multiplayer safe architecture.

UI RULES:
- TabView with 5 tabs only.
- Animated Singularity Eye in EmpireView.
- Business rows reusable component.
- Dark theme switchable by dimension.

ANALYTICS EVENTS TO TRACK:
session_start
session_end
first_purchase
prestige_triggered
dimension_unlocked
spin_used
jackpot_hit
subscription_started
merge_skin
upgrade_business
ai_level_up
soft_stuck_detected
cheat_detected

BUILD RULES:
- No force unwraps.
- No global mutable state.
- Zero compiler warnings.
- Document every economic formula.

SECTION 16 - DETAILED SYSTEM ARCHITECTURE BLOCK 16

ARCHITECTURE RULES:
- Strict MVVM separation.
- All models Codable.
- No business logic in Views.
- All multipliers calculated in Services layer.
- Constants centralized in GameConstants.swift.

CORE GAME LOOP IMPLEMENTATION DETAILS:

1. Initialize GameViewModel with:
   - Timer publisher (1 second interval).
   - State loading from PersistenceService.
   - Dependency injection of Services.

2. Income Tick Processing:
   - Calculate global multiplier.
   - Iterate businesses.
   - Accumulate income.
   - Update lifetime earnings.
   - Evaluate soft stuck.
   - Trigger analytics event if threshold met.

3. Prestige Logic:
   - Calculate prestigeGain via formula.
   - Increment prestigeCount.
   - Reset business levels.
   - Reset coins.
   - Persist state.
   - Log event prestige_triggered.

4. Skin System Deep Logic:
   - Maintain unlockedSkins array.
   - Active skins limited to 3.
   - Multipliers combined multiplicatively.
   - Hard cap at 0.40 global.
   - Merge validation before tier upgrade.

5. Spin Wheel Randomization:
   - Use weighted random selection.

```
   - Store spin history count.
   - Apply soft pity adjustment dynamically.
   - Ensure deterministic fairness.

6. Dimension Scaling:
   - Each dimension modifies base exponent.
   - Store dimension multiplier separately.
   - Visual theme switch triggered by dimension change.

7. Offline Income:
   - Store lastActiveTimestamp.
   - On app launch calculate delta.
   - Cap at 8 hours.
   - Apply offline multiplier.

8. Anti Cheat:
   - Validate tick income not exceeding 10x expected.
   - Validate prestige formula boundaries.
   - Disable leaderboard submission if anomaly.

9. Firebase Integration:
   - Firestore collection leaderboard.
   - Analytics events batched.
   - Error handling with retry policy.

10. StoreKit 2:
    - Verify transactions.
    - Handle subscription renewal state.
    - Restore purchases logic.

PERFORMANCE GUIDELINES:
- Avoid heavy loops inside Views.
- Cache multipliers.
- Avoid recomputing arrays each tick.
- All UI updates on main thread only.

TESTING REQUIREMENTS:
- Unit test income formula.
- Unit test prestige formula.
- Unit test cost growth.
- Unit test spin probability boundaries.
- Unit test merge logic.

SCALABILITY REQUIREMENTS:
- Unlimited future dimensions supported.
- Business list dynamic.
- Configurable balancing constants.
- Future multiplayer safe architecture.

UI RULES:
- TabView with 5 tabs only.
- Animated Singularity Eye in EmpireView.
- Business rows reusable component.
- Dark theme switchable by dimension.

ANALYTICS EVENTS TO TRACK:
session_start
session_end
first_purchase
prestige_triggered
dimension_unlocked
spin_used
jackpot_hit
subscription_started
merge_skin
upgrade_business
ai_level_up
soft_stuck_detected
cheat_detected

BUILD RULES:
- No force unwraps.
- No global mutable state.
- Zero compiler warnings.
- Document every economic formula.
```

```
SECTION 17 - DETAILED SYSTEM ARCHITECTURE BLOCK 17
--------------------------------------------------------------------------------

ARCHITECTURE RULES:
- Strict MVVM separation.
- All models Codable.
- No business logic in Views.
- All multipliers calculated in Services layer.
- Constants centralized in GameConstants.swift.

CORE GAME LOOP IMPLEMENTATION DETAILS:

1. Initialize GameViewModel with:
   - Timer publisher (1 second interval).
   - State loading from PersistenceService.
   - Dependency injection of Services.

2. Income Tick Processing:
   - Calculate global multiplier.
   - Iterate businesses.
   - Accumulate income.
   - Update lifetime earnings.
   - Evaluate soft stuck.
   - Trigger analytics event if threshold met.

3. Prestige Logic:
   - Calculate prestigeGain via formula.
   - Increment prestigeCount.
   - Reset business levels.
   - Reset coins.
   - Persist state.
   - Log event prestige_triggered.

4. Skin System Deep Logic:
   - Maintain unlockedSkins array.
   - Active skins limited to 3.
   - Multipliers combined multiplicatively.
   - Hard cap at 0.40 global.
   - Merge validation before tier upgrade.

5. Spin Wheel Randomization:
   - Use weighted random selection.
   - Store spin history count.
   - Apply soft pity adjustment dynamically.
   - Ensure deterministic fairness.

6. Dimension Scaling:
   - Each dimension modifies base exponent.
   - Store dimension multiplier separately.
   - Visual theme switch triggered by dimension change.

7. Offline Income:
   - Store lastActiveTimestamp.
   - On app launch calculate delta.
   - Cap at 8 hours.
   - Apply offline multiplier.

8. Anti Cheat:
   - Validate tick income not exceeding 10x expected.
   - Validate prestige formula boundaries.
   - Disable leaderboard submission if anomaly.

9. Firebase Integration:
   - Firestore collection leaderboard.
   - Analytics events batched.
   - Error handling with retry policy.

10. StoreKit 2:
    - Verify transactions.
    - Handle subscription renewal state.
    - Restore purchases logic.

PERFORMANCE GUIDELINES:
- Avoid heavy loops inside Views.
- Cache multipliers.
- Avoid recomputing arrays each tick.
```

- All UI updates on main thread only.

TESTING REQUIREMENTS:
- Unit test income formula.
- Unit test prestige formula.
- Unit test cost growth.
- Unit test spin probability boundaries.
- Unit test merge logic.

SCALABILITY REQUIREMENTS:
- Unlimited future dimensions supported.
- Business list dynamic.
- Configurable balancing constants.
- Future multiplayer safe architecture.

UI RULES:
- TabView with 5 tabs only.
- Animated Singularity Eye in EmpireView.
- Business rows reusable component.
- Dark theme switchable by dimension.

ANALYTICS EVENTS TO TRACK:
session_start
session_end
first_purchase
prestige_triggered
dimension_unlocked
spin_used
jackpot_hit
subscription_started
merge_skin
upgrade_business
ai_level_up
soft_stuck_detected
cheat_detected

BUILD RULES:
- No force unwraps.
- No global mutable state.
- Zero compiler warnings.
- Document every economic formula.

SECTION 18 - DETAILED SYSTEM ARCHITECTURE BLOCK 18

ARCHITECTURE RULES:
- Strict MVVM separation.
- All models Codable.
- No business logic in Views.
- All multipliers calculated in Services layer.
- Constants centralized in GameConstants.swift.

CORE GAME LOOP IMPLEMENTATION DETAILS:

1. Initialize GameViewModel with:
   - Timer publisher (1 second interval).
   - State loading from PersistenceService.
   - Dependency injection of Services.

2. Income Tick Processing:
   - Calculate global multiplier.
   - Iterate businesses.
   - Accumulate income.
   - Update lifetime earnings.
   - Evaluate soft stuck.
   - Trigger analytics event if threshold met.

3. Prestige Logic:
   - Calculate prestigeGain via formula.
   - Increment prestigeCount.
   - Reset business levels.
   - Reset coins.
   - Persist state.
   - Log event prestige_triggered.

4. Skin System Deep Logic:
   - Maintain unlockedSkins array.
   - Active skins limited to 3.
   - Multipliers combined multiplicatively.
   - Hard cap at 0.40 global.
   - Merge validation before tier upgrade.

5. Spin Wheel Randomization:
   - Use weighted random selection.
   - Store spin history count.
   - Apply soft pity adjustment dynamically.
   - Ensure deterministic fairness.

6. Dimension Scaling:
   - Each dimension modifies base exponent.
   - Store dimension multiplier separately.
   - Visual theme switch triggered by dimension change.

7. Offline Income:
   - Store lastActiveTimestamp.
   - On app launch calculate delta.
   - Cap at 8 hours.
   - Apply offline multiplier.

8. Anti Cheat:
   - Validate tick income not exceeding 10x expected.
   - Validate prestige formula boundaries.
   - Disable leaderboard submission if anomaly.

9. Firebase Integration:
   - Firestore collection leaderboard.
   - Analytics events batched.
   - Error handling with retry policy.

10. StoreKit 2:
    - Verify transactions.
    - Handle subscription renewal state.
    - Restore purchases logic.

PERFORMANCE GUIDELINES:
- Avoid heavy loops inside Views.
- Cache multipliers.
- Avoid recomputing arrays each tick.
- All UI updates on main thread only.

TESTING REQUIREMENTS:
- Unit test income formula.
- Unit test prestige formula.
- Unit test cost growth.
- Unit test spin probability boundaries.
- Unit test merge logic.

SCALABILITY REQUIREMENTS:
- Unlimited future dimensions supported.
- Business list dynamic.
- Configurable balancing constants.
- Future multiplayer safe architecture.

UI RULES:
- TabView with 5 tabs only.
- Animated Singularity Eye in EmpireView.
- Business rows reusable component.
- Dark theme switchable by dimension.

ANALYTICS EVENTS TO TRACK:
session_start
session_end
first_purchase
prestige_triggered
dimension_unlocked
spin_used
jackpot_hit
subscription_started
merge_skin
upgrade_business
ai_level_up

```
soft_stuck_detected
cheat_detected

BUILD RULES:
- No force unwraps.
- No global mutable state.
- Zero compiler warnings.
- Document every economic formula.


SECTION 19 - DETAILED SYSTEM ARCHITECTURE BLOCK 19
--------------------------------------------------------------------------------

ARCHITECTURE RULES:
- Strict MVVM separation.
- All models Codable.
- No business logic in Views.
- All multipliers calculated in Services layer.
- Constants centralized in GameConstants.swift.

CORE GAME LOOP IMPLEMENTATION DETAILS:

1. Initialize GameViewModel with:
   - Timer publisher (1 second interval).
   - State loading from PersistenceService.
   - Dependency injection of Services.

2. Income Tick Processing:
   - Calculate global multiplier.
   - Iterate businesses.
   - Accumulate income.
   - Update lifetime earnings.
   - Evaluate soft stuck.
   - Trigger analytics event if threshold met.

3. Prestige Logic:
   - Calculate prestigeGain via formula.
   - Increment prestigeCount.
   - Reset business levels.
   - Reset coins.
   - Persist state.
   - Log event prestige_triggered.

4. Skin System Deep Logic:
   - Maintain unlockedSkins array.
   - Active skins limited to 3.
   - Multipliers combined multiplicatively.
   - Hard cap at 0.40 global.
   - Merge validation before tier upgrade.

5. Spin Wheel Randomization:
   - Use weighted random selection.
   - Store spin history count.
   - Apply soft pity adjustment dynamically.
   - Ensure deterministic fairness.

6. Dimension Scaling:
   - Each dimension modifies base exponent.
   - Store dimension multiplier separately.
   - Visual theme switch triggered by dimension change.

7. Offline Income:
   - Store lastActiveTimestamp.
   - On app launch calculate delta.
   - Cap at 8 hours.
   - Apply offline multiplier.

8. Anti Cheat:
   - Validate tick income not exceeding 10x expected.
   - Validate prestige formula boundaries.
   - Disable leaderboard submission if anomaly.

9. Firebase Integration:
   - Firestore collection leaderboard.
   - Analytics events batched.
   - Error handling with retry policy.
```

```
10. StoreKit 2:
    - Verify transactions.
    - Handle subscription renewal state.
    - Restore purchases logic.

PERFORMANCE GUIDELINES:
- Avoid heavy loops inside Views.
- Cache multipliers.
- Avoid recomputing arrays each tick.
- All UI updates on main thread only.

TESTING REQUIREMENTS:
- Unit test income formula.
- Unit test prestige formula.
- Unit test cost growth.
- Unit test spin probability boundaries.
- Unit test merge logic.

SCALABILITY REQUIREMENTS:
- Unlimited future dimensions supported.
- Business list dynamic.
- Configurable balancing constants.
- Future multiplayer safe architecture.

UI RULES:
- TabView with 5 tabs only.
- Animated Singularity Eye in EmpireView.
- Business rows reusable component.
- Dark theme switchable by dimension.

ANALYTICS EVENTS TO TRACK:
session_start
session_end
first_purchase
prestige_triggered
dimension_unlocked
spin_used
jackpot_hit
subscription_started
merge_skin
upgrade_business
ai_level_up
soft_stuck_detected
cheat_detected

BUILD RULES:
- No force unwraps.
- No global mutable state.
- Zero compiler warnings.
- Document every economic formula.


SECTION 20 - DETAILED SYSTEM ARCHITECTURE BLOCK 20
--------------------------------------------------------------------------------

ARCHITECTURE RULES:
- Strict MVVM separation.
- All models Codable.
- No business logic in Views.
- All multipliers calculated in Services layer.
- Constants centralized in GameConstants.swift.

CORE GAME LOOP IMPLEMENTATION DETAILS:

1. Initialize GameViewModel with:
   - Timer publisher (1 second interval).
   - State loading from PersistenceService.
   - Dependency injection of Services.

2. Income Tick Processing:
   - Calculate global multiplier.
   - Iterate businesses.
   - Accumulate income.
   - Update lifetime earnings.
   - Evaluate soft stuck.
   - Trigger analytics event if threshold met.
```

3. Prestige Logic:
   - Calculate prestigeGain via formula.
   - Increment prestigeCount.
   - Reset business levels.
   - Reset coins.
   - Persist state.
   - Log event prestige_triggered.

4. Skin System Deep Logic:
   - Maintain unlockedSkins array.
   - Active skins limited to 3.
   - Multipliers combined multiplicatively.
   - Hard cap at 0.40 global.
   - Merge validation before tier upgrade.

5. Spin Wheel Randomization:
   - Use weighted random selection.
   - Store spin history count.
   - Apply soft pity adjustment dynamically.
   - Ensure deterministic fairness.

6. Dimension Scaling:
   - Each dimension modifies base exponent.
   - Store dimension multiplier separately.
   - Visual theme switch triggered by dimension change.

7. Offline Income:
   - Store lastActiveTimestamp.
   - On app launch calculate delta.
   - Cap at 8 hours.
   - Apply offline multiplier.

8. Anti Cheat:
   - Validate tick income not exceeding 10x expected.
   - Validate prestige formula boundaries.
   - Disable leaderboard submission if anomaly.

9. Firebase Integration:
   - Firestore collection leaderboard.
   - Analytics events batched.
   - Error handling with retry policy.

10. StoreKit 2:
    - Verify transactions.
    - Handle subscription renewal state.
    - Restore purchases logic.

PERFORMANCE GUIDELINES:
- Avoid heavy loops inside Views.
- Cache multipliers.
- Avoid recomputing arrays each tick.
- All UI updates on main thread only.

TESTING REQUIREMENTS:
- Unit test income formula.
- Unit test prestige formula.
- Unit test cost growth.
- Unit test spin probability boundaries.
- Unit test merge logic.

SCALABILITY REQUIREMENTS:
- Unlimited future dimensions supported.
- Business list dynamic.
- Configurable balancing constants.
- Future multiplayer safe architecture.

UI RULES:
- TabView with 5 tabs only.
- Animated Singularity Eye in EmpireView.
- Business rows reusable component.
- Dark theme switchable by dimension.

ANALYTICS EVENTS TO TRACK:
session_start
session_end
first_purchase

```
prestige_triggered
dimension_unlocked
spin_used
jackpot_hit
subscription_started
merge_skin
upgrade_business
ai_level_up
soft_stuck_detected
cheat_detected

BUILD RULES:
- No force unwraps.
- No global mutable state.
- Zero compiler warnings.
- Document every economic formula.


SECTION 21 - DETAILED SYSTEM ARCHITECTURE BLOCK 21
--------------------------------------------------------------------------------

ARCHITECTURE RULES:
- Strict MVVM separation.
- All models Codable.
- No business logic in Views.
- All multipliers calculated in Services layer.
- Constants centralized in GameConstants.swift.

CORE GAME LOOP IMPLEMENTATION DETAILS:

1. Initialize GameViewModel with:
   - Timer publisher (1 second interval).
   - State loading from PersistenceService.
   - Dependency injection of Services.

2. Income Tick Processing:
   - Calculate global multiplier.
   - Iterate businesses.
   - Accumulate income.
   - Update lifetime earnings.
   - Evaluate soft stuck.
   - Trigger analytics event if threshold met.

3. Prestige Logic:
   - Calculate prestigeGain via formula.
   - Increment prestigeCount.
   - Reset business levels.
   - Reset coins.
   - Persist state.
   - Log event prestige_triggered.

4. Skin System Deep Logic:
   - Maintain unlockedSkins array.
   - Active skins limited to 3.
   - Multipliers combined multiplicatively.
   - Hard cap at 0.40 global.
   - Merge validation before tier upgrade.

5. Spin Wheel Randomization:
   - Use weighted random selection.
   - Store spin history count.
   - Apply soft pity adjustment dynamically.
   - Ensure deterministic fairness.

6. Dimension Scaling:
   - Each dimension modifies base exponent.
   - Store dimension multiplier separately.
   - Visual theme switch triggered by dimension change.

7. Offline Income:
   - Store lastActiveTimestamp.
   - On app launch calculate delta.
   - Cap at 8 hours.
   - Apply offline multiplier.

8. Anti Cheat:
```

   - Validate tick income not exceeding 10x expected.
   - Validate prestige formula boundaries.
   - Disable leaderboard submission if anomaly.

9. Firebase Integration:
   - Firestore collection leaderboard.
   - Analytics events batched.
   - Error handling with retry policy.

10. StoreKit 2:
    - Verify transactions.
    - Handle subscription renewal state.
    - Restore purchases logic.

PERFORMANCE GUIDELINES:
- Avoid heavy loops inside Views.
- Cache multipliers.
- Avoid recomputing arrays each tick.
- All UI updates on main thread only.

TESTING REQUIREMENTS:
- Unit test income formula.
- Unit test prestige formula.
- Unit test cost growth.
- Unit test spin probability boundaries.
- Unit test merge logic.

SCALABILITY REQUIREMENTS:
- Unlimited future dimensions supported.
- Business list dynamic.
- Configurable balancing constants.
- Future multiplayer safe architecture.

UI RULES:
- TabView with 5 tabs only.
- Animated Singularity Eye in EmpireView.
- Business rows reusable component.
- Dark theme switchable by dimension.

ANALYTICS EVENTS TO TRACK:
session_start
session_end
first_purchase
prestige_triggered
dimension_unlocked
spin_used
jackpot_hit
subscription_started
merge_skin
upgrade_business
ai_level_up
soft_stuck_detected
cheat_detected

BUILD RULES:
- No force unwraps.
- No global mutable state.
- Zero compiler warnings.
- Document every economic formula.

SECTION 22 - DETAILED SYSTEM ARCHITECTURE BLOCK 22

ARCHITECTURE RULES:
- Strict MVVM separation.
- All models Codable.
- No business logic in Views.
- All multipliers calculated in Services layer.
- Constants centralized in GameConstants.swift.

CORE GAME LOOP IMPLEMENTATION DETAILS:

1. Initialize GameViewModel with:
   - Timer publisher (1 second interval).
   - State loading from PersistenceService.

   - Dependency injection of Services.

2. Income Tick Processing:
   - Calculate global multiplier.
   - Iterate businesses.
   - Accumulate income.
   - Update lifetime earnings.
   - Evaluate soft stuck.
   - Trigger analytics event if threshold met.

3. Prestige Logic:
   - Calculate prestigeGain via formula.
   - Increment prestigeCount.
   - Reset business levels.
   - Reset coins.
   - Persist state.
   - Log event prestige_triggered.

4. Skin System Deep Logic:
   - Maintain unlockedSkins array.
   - Active skins limited to 3.
   - Multipliers combined multiplicatively.
   - Hard cap at 0.40 global.
   - Merge validation before tier upgrade.

5. Spin Wheel Randomization:
   - Use weighted random selection.
   - Store spin history count.
   - Apply soft pity adjustment dynamically.
   - Ensure deterministic fairness.

6. Dimension Scaling:
   - Each dimension modifies base exponent.
   - Store dimension multiplier separately.
   - Visual theme switch triggered by dimension change.

7. Offline Income:
   - Store lastActiveTimestamp.
   - On app launch calculate delta.
   - Cap at 8 hours.
   - Apply offline multiplier.

8. Anti Cheat:
   - Validate tick income not exceeding 10x expected.
   - Validate prestige formula boundaries.
   - Disable leaderboard submission if anomaly.

9. Firebase Integration:
   - Firestore collection leaderboard.
   - Analytics events batched.
   - Error handling with retry policy.

10. StoreKit 2:
    - Verify transactions.
    - Handle subscription renewal state.
    - Restore purchases logic.

PERFORMANCE GUIDELINES:
- Avoid heavy loops inside Views.
- Cache multipliers.
- Avoid recomputing arrays each tick.
- All UI updates on main thread only.

TESTING REQUIREMENTS:
- Unit test income formula.
- Unit test prestige formula.
- Unit test cost growth.
- Unit test spin probability boundaries.
- Unit test merge logic.

SCALABILITY REQUIREMENTS:
- Unlimited future dimensions supported.
- Business list dynamic.
- Configurable balancing constants.
- Future multiplayer safe architecture.

UI RULES:
- TabView with 5 tabs only.
- Animated Singularity Eye in EmpireView.
- Business rows reusable component.
- Dark theme switchable by dimension.

ANALYTICS EVENTS TO TRACK:
session_start
session_end
first_purchase
prestige_triggered
dimension_unlocked
spin_used
jackpot_hit
subscription_started
merge_skin
upgrade_business
ai_level_up
soft_stuck_detected
cheat_detected

BUILD RULES:
- No force unwraps.
- No global mutable state.
- Zero compiler warnings.
- Document every economic formula.

SECTION 23 - DETAILED SYSTEM ARCHITECTURE BLOCK 23

ARCHITECTURE RULES:
- Strict MVVM separation.
- All models Codable.
- No business logic in Views.
- All multipliers calculated in Services layer.
- Constants centralized in GameConstants.swift.

CORE GAME LOOP IMPLEMENTATION DETAILS:

1. Initialize GameViewModel with:
   - Timer publisher (1 second interval).
   - State loading from PersistenceService.
   - Dependency injection of Services.

2. Income Tick Processing:
   - Calculate global multiplier.
   - Iterate businesses.
   - Accumulate income.
   - Update lifetime earnings.
   - Evaluate soft stuck.
   - Trigger analytics event if threshold met.

3. Prestige Logic:
   - Calculate prestigeGain via formula.
   - Increment prestigeCount.
   - Reset business levels.
   - Reset coins.
   - Persist state.
   - Log event prestige_triggered.

4. Skin System Deep Logic:
   - Maintain unlockedSkins array.
   - Active skins limited to 3.
   - Multipliers combined multiplicatively.
   - Hard cap at 0.40 global.
   - Merge validation before tier upgrade.

5. Spin Wheel Randomization:
   - Use weighted random selection.
   - Store spin history count.
   - Apply soft pity adjustment dynamically.
   - Ensure deterministic fairness.

6. Dimension Scaling:
   - Each dimension modifies base exponent.

```
   - Store dimension multiplier separately.
   - Visual theme switch triggered by dimension change.

7. Offline Income:
   - Store lastActiveTimestamp.
   - On app launch calculate delta.
   - Cap at 8 hours.
   - Apply offline multiplier.

8. Anti Cheat:
   - Validate tick income not exceeding 10x expected.
   - Validate prestige formula boundaries.
   - Disable leaderboard submission if anomaly.

9. Firebase Integration:
   - Firestore collection leaderboard.
   - Analytics events batched.
   - Error handling with retry policy.

10. StoreKit 2:
    - Verify transactions.
    - Handle subscription renewal state.
    - Restore purchases logic.

PERFORMANCE GUIDELINES:
- Avoid heavy loops inside Views.
- Cache multipliers.
- Avoid recomputing arrays each tick.
- All UI updates on main thread only.

TESTING REQUIREMENTS:
- Unit test income formula.
- Unit test prestige formula.
- Unit test cost growth.
- Unit test spin probability boundaries.
- Unit test merge logic.

SCALABILITY REQUIREMENTS:
- Unlimited future dimensions supported.
- Business list dynamic.
- Configurable balancing constants.
- Future multiplayer safe architecture.

UI RULES:
- TabView with 5 tabs only.
- Animated Singularity Eye in EmpireView.
- Business rows reusable component.
- Dark theme switchable by dimension.

ANALYTICS EVENTS TO TRACK:
session_start
session_end
first_purchase
prestige_triggered
dimension_unlocked
spin_used
jackpot_hit
subscription_started
merge_skin
upgrade_business
ai_level_up
soft_stuck_detected
cheat_detected

BUILD RULES:
- No force unwraps.
- No global mutable state.
- Zero compiler warnings.
- Document every economic formula.


SECTION 24 - DETAILED SYSTEM ARCHITECTURE BLOCK 24
--------------------------------------------------------------------------------

ARCHITECTURE RULES:
- Strict MVVM separation.
- All models Codable.
- No business logic in Views.
- All multipliers calculated in Services layer.
- Constants centralized in GameConstants.swift.

CORE GAME LOOP IMPLEMENTATION DETAILS:

1. Initialize GameViewModel with:
   - Timer publisher (1 second interval).
   - State loading from PersistenceService.
   - Dependency injection of Services.

2. Income Tick Processing:
   - Calculate global multiplier.
   - Iterate businesses.
   - Accumulate income.
   - Update lifetime earnings.
   - Evaluate soft stuck.
   - Trigger analytics event if threshold met.

3. Prestige Logic:
   - Calculate prestigeGain via formula.
   - Increment prestigeCount.
   - Reset business levels.
   - Reset coins.
   - Persist state.
   - Log event prestige_triggered.

4. Skin System Deep Logic:
   - Maintain unlockedSkins array.
   - Active skins limited to 3.
   - Multipliers combined multiplicatively.
   - Hard cap at 0.40 global.
   - Merge validation before tier upgrade.

5. Spin Wheel Randomization:
   - Use weighted random selection.
   - Store spin history count.
   - Apply soft pity adjustment dynamically.
   - Ensure deterministic fairness.

6. Dimension Scaling:
   - Each dimension modifies base exponent.
   - Store dimension multiplier separately.
   - Visual theme switch triggered by dimension change.

7. Offline Income:
   - Store lastActiveTimestamp.
   - On app launch calculate delta.
   - Cap at 8 hours.
   - Apply offline multiplier.

8. Anti Cheat:
   - Validate tick income not exceeding 10x expected.
   - Validate prestige formula boundaries.
   - Disable leaderboard submission if anomaly.

9. Firebase Integration:
   - Firestore collection leaderboard.
   - Analytics events batched.
   - Error handling with retry policy.

10. StoreKit 2:
    - Verify transactions.
    - Handle subscription renewal state.
    - Restore purchases logic.

PERFORMANCE GUIDELINES:
- Avoid heavy loops inside Views.
- Cache multipliers.
- Avoid recomputing arrays each tick.
- All UI updates on main thread only.

TESTING REQUIREMENTS:
- Unit test income formula.
- Unit test prestige formula.
```

- Unit test cost growth.
- Unit test spin probability boundaries.
- Unit test merge logic.

SCALABILITY REQUIREMENTS:
- Unlimited future dimensions supported.
- Business list dynamic.
- Configurable balancing constants.
- Future multiplayer safe architecture.

UI RULES:
- TabView with 5 tabs only.
- Animated Singularity Eye in EmpireView.
- Business rows reusable component.
- Dark theme switchable by dimension.

ANALYTICS EVENTS TO TRACK:
session_start
session_end
first_purchase
prestige_triggered
dimension_unlocked
spin_used
jackpot_hit
subscription_started
merge_skin
upgrade_business
ai_level_up
soft_stuck_detected
cheat_detected

BUILD RULES:
- No force unwraps.
- No global mutable state.
- Zero compiler warnings.
- Document every economic formula.

SECTION 25 - DETAILED SYSTEM ARCHITECTURE BLOCK 25

ARCHITECTURE RULES:
- Strict MVVM separation.
- All models Codable.
- No business logic in Views.
- All multipliers calculated in Services layer.
- Constants centralized in GameConstants.swift.

CORE GAME LOOP IMPLEMENTATION DETAILS:

1. Initialize GameViewModel with:
   - Timer publisher (1 second interval).
   - State loading from PersistenceService.
   - Dependency injection of Services.

2. Income Tick Processing:
   - Calculate global multiplier.
   - Iterate businesses.
   - Accumulate income.
   - Update lifetime earnings.
   - Evaluate soft stuck.
   - Trigger analytics event if threshold met.

3. Prestige Logic:
   - Calculate prestigeGain via formula.
   - Increment prestigeCount.
   - Reset business levels.
   - Reset coins.
   - Persist state.
   - Log event prestige_triggered.

4. Skin System Deep Logic:
   - Maintain unlockedSkins array.
   - Active skins limited to 3.
   - Multipliers combined multiplicatively.
   - Hard cap at 0.40 global.

```
   - Merge validation before tier upgrade.

5. Spin Wheel Randomization:
   - Use weighted random selection.
   - Store spin history count.
   - Apply soft pity adjustment dynamically.
   - Ensure deterministic fairness.

6. Dimension Scaling:
   - Each dimension modifies base exponent.
   - Store dimension multiplier separately.
   - Visual theme switch triggered by dimension change.

7. Offline Income:
   - Store lastActiveTimestamp.
   - On app launch calculate delta.
   - Cap at 8 hours.
   - Apply offline multiplier.

8. Anti Cheat:
   - Validate tick income not exceeding 10x expected.
   - Validate prestige formula boundaries.
   - Disable leaderboard submission if anomaly.

9. Firebase Integration:
   - Firestore collection leaderboard.
   - Analytics events batched.
   - Error handling with retry policy.

10. StoreKit 2:
    - Verify transactions.
    - Handle subscription renewal state.
    - Restore purchases logic.

PERFORMANCE GUIDELINES:
- Avoid heavy loops inside Views.
- Cache multipliers.
- Avoid recomputing arrays each tick.
- All UI updates on main thread only.

TESTING REQUIREMENTS:
- Unit test income formula.
- Unit test prestige formula.
- Unit test cost growth.
- Unit test spin probability boundaries.
- Unit test merge logic.

SCALABILITY REQUIREMENTS:
- Unlimited future dimensions supported.
- Business list dynamic.
- Configurable balancing constants.
- Future multiplayer safe architecture.

UI RULES:
- TabView with 5 tabs only.
- Animated Singularity Eye in EmpireView.
- Business rows reusable component.
- Dark theme switchable by dimension.

ANALYTICS EVENTS TO TRACK:
session_start
session_end
first_purchase
prestige_triggered
dimension_unlocked
spin_used
jackpot_hit
subscription_started
merge_skin
upgrade_business
ai_level_up
soft_stuck_detected
cheat_detected

BUILD RULES:
- No force unwraps.
```

- No global mutable state.
- Zero compiler warnings.
- Document every economic formula.

SECTION 26 - DETAILED SYSTEM ARCHITECTURE BLOCK 26

ARCHITECTURE RULES:
- Strict MVVM separation.
- All models Codable.
- No business logic in Views.
- All multipliers calculated in Services layer.
- Constants centralized in GameConstants.swift.

CORE GAME LOOP IMPLEMENTATION DETAILS:

1. Initialize GameViewModel with:
   - Timer publisher (1 second interval).
   - State loading from PersistenceService.
   - Dependency injection of Services.

2. Income Tick Processing:
   - Calculate global multiplier.
   - Iterate businesses.
   - Accumulate income.
   - Update lifetime earnings.
   - Evaluate soft stuck.
   - Trigger analytics event if threshold met.

3. Prestige Logic:
   - Calculate prestigeGain via formula.
   - Increment prestigeCount.
   - Reset business levels.
   - Reset coins.
   - Persist state.
   - Log event prestige_triggered.

4. Skin System Deep Logic:
   - Maintain unlockedSkins array.
   - Active skins limited to 3.
   - Multipliers combined multiplicatively.
   - Hard cap at 0.40 global.
   - Merge validation before tier upgrade.

5. Spin Wheel Randomization:
   - Use weighted random selection.
   - Store spin history count.
   - Apply soft pity adjustment dynamically.
   - Ensure deterministic fairness.

6. Dimension Scaling:
   - Each dimension modifies base exponent.
   - Store dimension multiplier separately.
   - Visual theme switch triggered by dimension change.

7. Offline Income:
   - Store lastActiveTimestamp.
   - On app launch calculate delta.
   - Cap at 8 hours.
   - Apply offline multiplier.

8. Anti Cheat:
   - Validate tick income not exceeding 10x expected.
   - Validate prestige formula boundaries.
   - Disable leaderboard submission if anomaly.

9. Firebase Integration:
   - Firestore collection leaderboard.
   - Analytics events batched.
   - Error handling with retry policy.

10. StoreKit 2:
    - Verify transactions.
    - Handle subscription renewal state.
    - Restore purchases logic.

```
PERFORMANCE GUIDELINES:
- Avoid heavy loops inside Views.
- Cache multipliers.
- Avoid recomputing arrays each tick.
- All UI updates on main thread only.

TESTING REQUIREMENTS:
- Unit test income formula.
- Unit test prestige formula.
- Unit test cost growth.
- Unit test spin probability boundaries.
- Unit test merge logic.

SCALABILITY REQUIREMENTS:
- Unlimited future dimensions supported.
- Business list dynamic.
- Configurable balancing constants.
- Future multiplayer safe architecture.

UI RULES:
- TabView with 5 tabs only.
- Animated Singularity Eye in EmpireView.
- Business rows reusable component.
- Dark theme switchable by dimension.

ANALYTICS EVENTS TO TRACK:
session_start
session_end
first_purchase
prestige_triggered
dimension_unlocked
spin_used
jackpot_hit
subscription_started
merge_skin
upgrade_business
ai_level_up
soft_stuck_detected
cheat_detected

BUILD RULES:
- No force unwraps.
- No global mutable state.
- Zero compiler warnings.
- Document every economic formula.


SECTION 27 - DETAILED SYSTEM ARCHITECTURE BLOCK 27
--------------------------------------------------------------------------------

ARCHITECTURE RULES:
- Strict MVVM separation.
- All models Codable.
- No business logic in Views.
- All multipliers calculated in Services layer.
- Constants centralized in GameConstants.swift.

CORE GAME LOOP IMPLEMENTATION DETAILS:

1. Initialize GameViewModel with:
   - Timer publisher (1 second interval).
   - State loading from PersistenceService.
   - Dependency injection of Services.

2. Income Tick Processing:
   - Calculate global multiplier.
   - Iterate businesses.
   - Accumulate income.
   - Update lifetime earnings.
   - Evaluate soft stuck.
   - Trigger analytics event if threshold met.

3. Prestige Logic:
   - Calculate prestigeGain via formula.
   - Increment prestigeCount.
   - Reset business levels.
   - Reset coins.
   - Persist state.
   - Log event prestige_triggered.

4. Skin System Deep Logic:
   - Maintain unlockedSkins array.
   - Active skins limited to 3.
   - Multipliers combined multiplicatively.
   - Hard cap at 0.40 global.
   - Merge validation before tier upgrade.

5. Spin Wheel Randomization:
   - Use weighted random selection.
   - Store spin history count.
   - Apply soft pity adjustment dynamically.
   - Ensure deterministic fairness.

6. Dimension Scaling:
   - Each dimension modifies base exponent.
   - Store dimension multiplier separately.
   - Visual theme switch triggered by dimension change.

7. Offline Income:
   - Store lastActiveTimestamp.
   - On app launch calculate delta.
   - Cap at 8 hours.
   - Apply offline multiplier.

8. Anti Cheat:
   - Validate tick income not exceeding 10x expected.
   - Validate prestige formula boundaries.
   - Disable leaderboard submission if anomaly.

9. Firebase Integration:
   - Firestore collection leaderboard.
   - Analytics events batched.
   - Error handling with retry policy.

10. StoreKit 2:
    - Verify transactions.
    - Handle subscription renewal state.
    - Restore purchases logic.

PERFORMANCE GUIDELINES:
- Avoid heavy loops inside Views.
- Cache multipliers.
- Avoid recomputing arrays each tick.
- All UI updates on main thread only.

TESTING REQUIREMENTS:
- Unit test income formula.
- Unit test prestige formula.
- Unit test cost growth.
- Unit test spin probability boundaries.
- Unit test merge logic.

SCALABILITY REQUIREMENTS:
- Unlimited future dimensions supported.
- Business list dynamic.
- Configurable balancing constants.
- Future multiplayer safe architecture.

UI RULES:
- TabView with 5 tabs only.
- Animated Singularity Eye in EmpireView.
- Business rows reusable component.
- Dark theme switchable by dimension.

ANALYTICS EVENTS TO TRACK:
session_start
session_end
first_purchase
prestige_triggered
dimension_unlocked
spin_used
jackpot_hit
```

```
subscription_started
merge_skin
upgrade_business
ai_level_up
soft_stuck_detected
cheat_detected

BUILD RULES:
- No force unwraps.
- No global mutable state.
- Zero compiler warnings.
- Document every economic formula.


SECTION 28 - DETAILED SYSTEM ARCHITECTURE BLOCK 28
--------------------------------------------------------------------------------

ARCHITECTURE RULES:
- Strict MVVM separation.
- All models Codable.
- No business logic in Views.
- All multipliers calculated in Services layer.
- Constants centralized in GameConstants.swift.

CORE GAME LOOP IMPLEMENTATION DETAILS:

1. Initialize GameViewModel with:
   - Timer publisher (1 second interval).
   - State loading from PersistenceService.
   - Dependency injection of Services.

2. Income Tick Processing:
   - Calculate global multiplier.
   - Iterate businesses.
   - Accumulate income.
   - Update lifetime earnings.
   - Evaluate soft stuck.
   - Trigger analytics event if threshold met.

3. Prestige Logic:
   - Calculate prestigeGain via formula.
   - Increment prestigeCount.
   - Reset business levels.
   - Reset coins.
   - Persist state.
   - Log event prestige_triggered.

4. Skin System Deep Logic:
   - Maintain unlockedSkins array.
   - Active skins limited to 3.
   - Multipliers combined multiplicatively.
   - Hard cap at 0.40 global.
   - Merge validation before tier upgrade.

5. Spin Wheel Randomization:
   - Use weighted random selection.
   - Store spin history count.
   - Apply soft pity adjustment dynamically.
   - Ensure deterministic fairness.

6. Dimension Scaling:
   - Each dimension modifies base exponent.
   - Store dimension multiplier separately.
   - Visual theme switch triggered by dimension change.

7. Offline Income:
   - Store lastActiveTimestamp.
   - On app launch calculate delta.
   - Cap at 8 hours.
   - Apply offline multiplier.

8. Anti Cheat:
   - Validate tick income not exceeding 10x expected.
   - Validate prestige formula boundaries.
   - Disable leaderboard submission if anomaly.
```

9. Firebase Integration:
   - Firestore collection leaderboard.
   - Analytics events batched.
   - Error handling with retry policy.

10. StoreKit 2:
    - Verify transactions.
    - Handle subscription renewal state.
    - Restore purchases logic.

PERFORMANCE GUIDELINES:
- Avoid heavy loops inside Views.
- Cache multipliers.
- Avoid recomputing arrays each tick.
- All UI updates on main thread only.

TESTING REQUIREMENTS:
- Unit test income formula.
- Unit test prestige formula.
- Unit test cost growth.
- Unit test spin probability boundaries.
- Unit test merge logic.

SCALABILITY REQUIREMENTS:
- Unlimited future dimensions supported.
- Business list dynamic.
- Configurable balancing constants.
- Future multiplayer safe architecture.

UI RULES:
- TabView with 5 tabs only.
- Animated Singularity Eye in EmpireView.
- Business rows reusable component.
- Dark theme switchable by dimension.

ANALYTICS EVENTS TO TRACK:
session_start
session_end
first_purchase
prestige_triggered
dimension_unlocked
spin_used
jackpot_hit
subscription_started
merge_skin
upgrade_business
ai_level_up
soft_stuck_detected
cheat_detected

BUILD RULES:
- No force unwraps.
- No global mutable state.
- Zero compiler warnings.
- Document every economic formula.

SECTION 29 - DETAILED SYSTEM ARCHITECTURE BLOCK 29

ARCHITECTURE RULES:
- Strict MVVM separation.
- All models Codable.
- No business logic in Views.
- All multipliers calculated in Services layer.
- Constants centralized in GameConstants.swift.

CORE GAME LOOP IMPLEMENTATION DETAILS:

1. Initialize GameViewModel with:
   - Timer publisher (1 second interval).
   - State loading from PersistenceService.
   - Dependency injection of Services.

2. Income Tick Processing:
   - Calculate global multiplier.

```
   - Iterate businesses.
   - Accumulate income.
   - Update lifetime earnings.
   - Evaluate soft stuck.
   - Trigger analytics event if threshold met.

3. Prestige Logic:
   - Calculate prestigeGain via formula.
   - Increment prestigeCount.
   - Reset business levels.
   - Reset coins.
   - Persist state.
   - Log event prestige_triggered.

4. Skin System Deep Logic:
   - Maintain unlockedSkins array.
   - Active skins limited to 3.
   - Multipliers combined multiplicatively.
   - Hard cap at 0.40 global.
   - Merge validation before tier upgrade.

5. Spin Wheel Randomization:
   - Use weighted random selection.
   - Store spin history count.
   - Apply soft pity adjustment dynamically.
   - Ensure deterministic fairness.

6. Dimension Scaling:
   - Each dimension modifies base exponent.
   - Store dimension multiplier separately.
   - Visual theme switch triggered by dimension change.

7. Offline Income:
   - Store lastActiveTimestamp.
   - On app launch calculate delta.
   - Cap at 8 hours.
   - Apply offline multiplier.

8. Anti Cheat:
   - Validate tick income not exceeding 10x expected.
   - Validate prestige formula boundaries.
   - Disable leaderboard submission if anomaly.

9. Firebase Integration:
   - Firestore collection leaderboard.
   - Analytics events batched.
   - Error handling with retry policy.

10. StoreKit 2:
    - Verify transactions.
    - Handle subscription renewal state.
    - Restore purchases logic.

PERFORMANCE GUIDELINES:
- Avoid heavy loops inside Views.
- Cache multipliers.
- Avoid recomputing arrays each tick.
- All UI updates on main thread only.

TESTING REQUIREMENTS:
- Unit test income formula.
- Unit test prestige formula.
- Unit test cost growth.
- Unit test spin probability boundaries.
- Unit test merge logic.

SCALABILITY REQUIREMENTS:
- Unlimited future dimensions supported.
- Business list dynamic.
- Configurable balancing constants.
- Future multiplayer safe architecture.

UI RULES:
- TabView with 5 tabs only.
- Animated Singularity Eye in EmpireView.
- Business rows reusable component.
```

- Dark theme switchable by dimension.

ANALYTICS EVENTS TO TRACK:
session_start
session_end
first_purchase
prestige_triggered
dimension_unlocked
spin_used
jackpot_hit
subscription_started
merge_skin
upgrade_business
ai_level_up
soft_stuck_detected
cheat_detected

BUILD RULES:
- No force unwraps.
- No global mutable state.
- Zero compiler warnings.
- Document every economic formula.

SECTION 30 - DETAILED SYSTEM ARCHITECTURE BLOCK 30

ARCHITECTURE RULES:
- Strict MVVM separation.
- All models Codable.
- No business logic in Views.
- All multipliers calculated in Services layer.
- Constants centralized in GameConstants.swift.

CORE GAME LOOP IMPLEMENTATION DETAILS:

1. Initialize GameViewModel with:
   - Timer publisher (1 second interval).
   - State loading from PersistenceService.
   - Dependency injection of Services.

2. Income Tick Processing:
   - Calculate global multiplier.
   - Iterate businesses.
   - Accumulate income.
   - Update lifetime earnings.
   - Evaluate soft stuck.
   - Trigger analytics event if threshold met.

3. Prestige Logic:
   - Calculate prestigeGain via formula.
   - Increment prestigeCount.
   - Reset business levels.
   - Reset coins.
   - Persist state.
   - Log event prestige_triggered.

4. Skin System Deep Logic:
   - Maintain unlockedSkins array.
   - Active skins limited to 3.
   - Multipliers combined multiplicatively.
   - Hard cap at 0.40 global.
   - Merge validation before tier upgrade.

5. Spin Wheel Randomization:
   - Use weighted random selection.
   - Store spin history count.
   - Apply soft pity adjustment dynamically.
   - Ensure deterministic fairness.

6. Dimension Scaling:
   - Each dimension modifies base exponent.
   - Store dimension multiplier separately.
   - Visual theme switch triggered by dimension change.

7. Offline Income:

```
   - Store lastActiveTimestamp.
   - On app launch calculate delta.
   - Cap at 8 hours.
   - Apply offline multiplier.

8. Anti Cheat:
   - Validate tick income not exceeding 10x expected.
   - Validate prestige formula boundaries.
   - Disable leaderboard submission if anomaly.

9. Firebase Integration:
   - Firestore collection leaderboard.
   - Analytics events batched.
   - Error handling with retry policy.

10. StoreKit 2:
    - Verify transactions.
    - Handle subscription renewal state.
    - Restore purchases logic.

PERFORMANCE GUIDELINES:
- Avoid heavy loops inside Views.
- Cache multipliers.
- Avoid recomputing arrays each tick.
- All UI updates on main thread only.

TESTING REQUIREMENTS:
- Unit test income formula.
- Unit test prestige formula.
- Unit test cost growth.
- Unit test spin probability boundaries.
- Unit test merge logic.

SCALABILITY REQUIREMENTS:
- Unlimited future dimensions supported.
- Business list dynamic.
- Configurable balancing constants.
- Future multiplayer safe architecture.

UI RULES:
- TabView with 5 tabs only.
- Animated Singularity Eye in EmpireView.
- Business rows reusable component.
- Dark theme switchable by dimension.

ANALYTICS EVENTS TO TRACK:
session_start
session_end
first_purchase
prestige_triggered
dimension_unlocked
spin_used
jackpot_hit
subscription_started
merge_skin
upgrade_business
ai_level_up
soft_stuck_detected
cheat_detected

BUILD RULES:
- No force unwraps.
- No global mutable state.
- Zero compiler warnings.
- Document every economic formula.


SECTION 31 - DETAILED SYSTEM ARCHITECTURE BLOCK 31
--------------------------------------------------------------------------------

ARCHITECTURE RULES:
- Strict MVVM separation.
- All models Codable.
- No business logic in Views.
- All multipliers calculated in Services layer.
- Constants centralized in GameConstants.swift.
```

CORE GAME LOOP IMPLEMENTATION DETAILS:

1. Initialize GameViewModel with:
   - Timer publisher (1 second interval).
   - State loading from PersistenceService.
   - Dependency injection of Services.

2. Income Tick Processing:
   - Calculate global multiplier.
   - Iterate businesses.
   - Accumulate income.
   - Update lifetime earnings.
   - Evaluate soft stuck.
   - Trigger analytics event if threshold met.

3. Prestige Logic:
   - Calculate prestigeGain via formula.
   - Increment prestigeCount.
   - Reset business levels.
   - Reset coins.
   - Persist state.
   - Log event prestige_triggered.

4. Skin System Deep Logic:
   - Maintain unlockedSkins array.
   - Active skins limited to 3.
   - Multipliers combined multiplicatively.
   - Hard cap at 0.40 global.
   - Merge validation before tier upgrade.

5. Spin Wheel Randomization:
   - Use weighted random selection.
   - Store spin history count.
   - Apply soft pity adjustment dynamically.
   - Ensure deterministic fairness.

6. Dimension Scaling:
   - Each dimension modifies base exponent.
   - Store dimension multiplier separately.
   - Visual theme switch triggered by dimension change.

7. Offline Income:
   - Store lastActiveTimestamp.
   - On app launch calculate delta.
   - Cap at 8 hours.
   - Apply offline multiplier.

8. Anti Cheat:
   - Validate tick income not exceeding 10x expected.
   - Validate prestige formula boundaries.
   - Disable leaderboard submission if anomaly.

9. Firebase Integration:
   - Firestore collection leaderboard.
   - Analytics events batched.
   - Error handling with retry policy.

10. StoreKit 2:
    - Verify transactions.
    - Handle subscription renewal state.
    - Restore purchases logic.

PERFORMANCE GUIDELINES:
- Avoid heavy loops inside Views.
- Cache multipliers.
- Avoid recomputing arrays each tick.
- All UI updates on main thread only.

TESTING REQUIREMENTS:
- Unit test income formula.
- Unit test prestige formula.
- Unit test cost growth.
- Unit test spin probability boundaries.
- Unit test merge logic.

SCALABILITY REQUIREMENTS:

```
- Unlimited future dimensions supported.
- Business list dynamic.
- Configurable balancing constants.
- Future multiplayer safe architecture.

UI RULES:
- TabView with 5 tabs only.
- Animated Singularity Eye in EmpireView.
- Business rows reusable component.
- Dark theme switchable by dimension.

ANALYTICS EVENTS TO TRACK:
session_start
session_end
first_purchase
prestige_triggered
dimension_unlocked
spin_used
jackpot_hit
subscription_started
merge_skin
upgrade_business
ai_level_up
soft_stuck_detected
cheat_detected

BUILD RULES:
- No force unwraps.
- No global mutable state.
- Zero compiler warnings.
- Document every economic formula.


SECTION 32 - DETAILED SYSTEM ARCHITECTURE BLOCK 32
--------------------------------------------------------------------------------

ARCHITECTURE RULES:
- Strict MVVM separation.
- All models Codable.
- No business logic in Views.
- All multipliers calculated in Services layer.
- Constants centralized in GameConstants.swift.

CORE GAME LOOP IMPLEMENTATION DETAILS:

1. Initialize GameViewModel with:
   - Timer publisher (1 second interval).
   - State loading from PersistenceService.
   - Dependency injection of Services.

2. Income Tick Processing:
   - Calculate global multiplier.
   - Iterate businesses.
   - Accumulate income.
   - Update lifetime earnings.
   - Evaluate soft stuck.
   - Trigger analytics event if threshold met.

3. Prestige Logic:
   - Calculate prestigeGain via formula.
   - Increment prestigeCount.
   - Reset business levels.
   - Reset coins.
   - Persist state.
   - Log event prestige_triggered.

4. Skin System Deep Logic:
   - Maintain unlockedSkins array.
   - Active skins limited to 3.
   - Multipliers combined multiplicatively.
   - Hard cap at 0.40 global.
   - Merge validation before tier upgrade.

5. Spin Wheel Randomization:
   - Use weighted random selection.
   - Store spin history count.
```

   - Apply soft pity adjustment dynamically.
   - Ensure deterministic fairness.

6. Dimension Scaling:
   - Each dimension modifies base exponent.
   - Store dimension multiplier separately.
   - Visual theme switch triggered by dimension change.

7. Offline Income:
   - Store lastActiveTimestamp.
   - On app launch calculate delta.
   - Cap at 8 hours.
   - Apply offline multiplier.

8. Anti Cheat:
   - Validate tick income not exceeding 10x expected.
   - Validate prestige formula boundaries.
   - Disable leaderboard submission if anomaly.

9. Firebase Integration:
   - Firestore collection leaderboard.
   - Analytics events batched.
   - Error handling with retry policy.

10. StoreKit 2:
    - Verify transactions.
    - Handle subscription renewal state.
    - Restore purchases logic.

PERFORMANCE GUIDELINES:
- Avoid heavy loops inside Views.
- Cache multipliers.
- Avoid recomputing arrays each tick.
- All UI updates on main thread only.

TESTING REQUIREMENTS:
- Unit test income formula.
- Unit test prestige formula.
- Unit test cost growth.
- Unit test spin probability boundaries.
- Unit test merge logic.

SCALABILITY REQUIREMENTS:
- Unlimited future dimensions supported.
- Business list dynamic.
- Configurable balancing constants.
- Future multiplayer safe architecture.

UI RULES:
- TabView with 5 tabs only.
- Animated Singularity Eye in EmpireView.
- Business rows reusable component.
- Dark theme switchable by dimension.

ANALYTICS EVENTS TO TRACK:
session_start
session_end
first_purchase
prestige_triggered
dimension_unlocked
spin_used
jackpot_hit
subscription_started
merge_skin
upgrade_business
ai_level_up
soft_stuck_detected
cheat_detected

BUILD RULES:
- No force unwraps.
- No global mutable state.
- Zero compiler warnings.
- Document every economic formula.

SECTION 33 - DETAILED SYSTEM ARCHITECTURE BLOCK 33

ARCHITECTURE RULES:
- Strict MVVM separation.
- All models Codable.
- No business logic in Views.
- All multipliers calculated in Services layer.
- Constants centralized in GameConstants.swift.

CORE GAME LOOP IMPLEMENTATION DETAILS:

1. Initialize GameViewModel with:
   - Timer publisher (1 second interval).
   - State loading from PersistenceService.
   - Dependency injection of Services.

2. Income Tick Processing:
   - Calculate global multiplier.
   - Iterate businesses.
   - Accumulate income.
   - Update lifetime earnings.
   - Evaluate soft stuck.
   - Trigger analytics event if threshold met.

3. Prestige Logic:
   - Calculate prestigeGain via formula.
   - Increment prestigeCount.
   - Reset business levels.
   - Reset coins.
   - Persist state.
   - Log event prestige_triggered.

4. Skin System Deep Logic:
   - Maintain unlockedSkins array.
   - Active skins limited to 3.
   - Multipliers combined multiplicatively.
   - Hard cap at 0.40 global.
   - Merge validation before tier upgrade.

5. Spin Wheel Randomization:
   - Use weighted random selection.
   - Store spin history count.
   - Apply soft pity adjustment dynamically.
   - Ensure deterministic fairness.

6. Dimension Scaling:
   - Each dimension modifies base exponent.
   - Store dimension multiplier separately.
   - Visual theme switch triggered by dimension change.

7. Offline Income:
   - Store lastActiveTimestamp.
   - On app launch calculate delta.
   - Cap at 8 hours.
   - Apply offline multiplier.

8. Anti Cheat:
   - Validate tick income not exceeding 10x expected.
   - Validate prestige formula boundaries.
   - Disable leaderboard submission if anomaly.

9. Firebase Integration:
   - Firestore collection leaderboard.
   - Analytics events batched.
   - Error handling with retry policy.

10. StoreKit 2:
    - Verify transactions.
    - Handle subscription renewal state.
    - Restore purchases logic.

PERFORMANCE GUIDELINES:
- Avoid heavy loops inside Views.
- Cache multipliers.
- Avoid recomputing arrays each tick.
- All UI updates on main thread only.

TESTING REQUIREMENTS:
- Unit test income formula.
- Unit test prestige formula.
- Unit test cost growth.
- Unit test spin probability boundaries.
- Unit test merge logic.

SCALABILITY REQUIREMENTS:
- Unlimited future dimensions supported.
- Business list dynamic.
- Configurable balancing constants.
- Future multiplayer safe architecture.

UI RULES:
- TabView with 5 tabs only.
- Animated Singularity Eye in EmpireView.
- Business rows reusable component.
- Dark theme switchable by dimension.

ANALYTICS EVENTS TO TRACK:

session_start
session_end
first_purchase
prestige_triggered
dimension_unlocked
spin_used
jackpot_hit
subscription_started
merge_skin
upgrade_business
ai_level_up
soft_stuck_detected
cheat_detected

BUILD RULES:
- No force unwraps.
- No global mutable state.
- Zero compiler warnings.
- Document every economic formula.

SECTION 34 - DETAILED SYSTEM ARCHITECTURE BLOCK 34

ARCHITECTURE RULES:
- Strict MVVM separation.
- All models Codable.
- No business logic in Views.
- All multipliers calculated in Services layer.
- Constants centralized in GameConstants.swift.

CORE GAME LOOP IMPLEMENTATION DETAILS:

1. Initialize GameViewModel with:
   - Timer publisher (1 second interval).
   - State loading from PersistenceService.
   - Dependency injection of Services.

2. Income Tick Processing:
   - Calculate global multiplier.
   - Iterate businesses.
   - Accumulate income.
   - Update lifetime earnings.
   - Evaluate soft stuck.
   - Trigger analytics event if threshold met.

3. Prestige Logic:
   - Calculate prestigeGain via formula.
   - Increment prestigeCount.
   - Reset business levels.
   - Reset coins.
   - Persist state.
   - Log event prestige_triggered.

4. Skin System Deep Logic:
   - Maintain unlockedSkins array.

   - Active skins limited to 3.
   - Multipliers combined multiplicatively.
   - Hard cap at 0.40 global.
   - Merge validation before tier upgrade.

5. Spin Wheel Randomization:
   - Use weighted random selection.
   - Store spin history count.
   - Apply soft pity adjustment dynamically.
   - Ensure deterministic fairness.

6. Dimension Scaling:
   - Each dimension modifies base exponent.
   - Store dimension multiplier separately.
   - Visual theme switch triggered by dimension change.

7. Offline Income:
   - Store lastActiveTimestamp.
   - On app launch calculate delta.
   - Cap at 8 hours.
   - Apply offline multiplier.

8. Anti Cheat:
   - Validate tick income not exceeding 10x expected.
   - Validate prestige formula boundaries.
   - Disable leaderboard submission if anomaly.

9. Firebase Integration:
   - Firestore collection leaderboard.
   - Analytics events batched.
   - Error handling with retry policy.

10. StoreKit 2:
    - Verify transactions.
    - Handle subscription renewal state.
    - Restore purchases logic.

PERFORMANCE GUIDELINES:
- Avoid heavy loops inside Views.
- Cache multipliers.
- Avoid recomputing arrays each tick.
- All UI updates on main thread only.

TESTING REQUIREMENTS:
- Unit test income formula.
- Unit test prestige formula.
- Unit test cost growth.
- Unit test spin probability boundaries.
- Unit test merge logic.

SCALABILITY REQUIREMENTS:
- Unlimited future dimensions supported.
- Business list dynamic.
- Configurable balancing constants.
- Future multiplayer safe architecture.

UI RULES:
- TabView with 5 tabs only.
- Animated Singularity Eye in EmpireView.
- Business rows reusable component.
- Dark theme switchable by dimension.

ANALYTICS EVENTS TO TRACK:
session_start
session_end
first_purchase
prestige_triggered
dimension_unlocked
spin_used
jackpot_hit
subscription_started
merge_skin
upgrade_business
ai_level_up
soft_stuck_detected
cheat_detected

```
BUILD RULES:
- No force unwraps.
- No global mutable state.
- Zero compiler warnings.
- Document every economic formula.


SECTION 35 - DETAILED SYSTEM ARCHITECTURE BLOCK 35
--------------------------------------------------------------------------------

ARCHITECTURE RULES:
- Strict MVVM separation.
- All models Codable.
- No business logic in Views.
- All multipliers calculated in Services layer.
- Constants centralized in GameConstants.swift.

CORE GAME LOOP IMPLEMENTATION DETAILS:

1. Initialize GameViewModel with:
   - Timer publisher (1 second interval).
   - State loading from PersistenceService.
   - Dependency injection of Services.

2. Income Tick Processing:
   - Calculate global multiplier.
   - Iterate businesses.
   - Accumulate income.
   - Update lifetime earnings.
   - Evaluate soft stuck.
   - Trigger analytics event if threshold met.

3. Prestige Logic:
   - Calculate prestigeGain via formula.
   - Increment prestigeCount.
   - Reset business levels.
   - Reset coins.
   - Persist state.
   - Log event prestige_triggered.

4. Skin System Deep Logic:
   - Maintain unlockedSkins array.
   - Active skins limited to 3.
   - Multipliers combined multiplicatively.
   - Hard cap at 0.40 global.
   - Merge validation before tier upgrade.

5. Spin Wheel Randomization:
   - Use weighted random selection.
   - Store spin history count.
   - Apply soft pity adjustment dynamically.
   - Ensure deterministic fairness.

6. Dimension Scaling:
   - Each dimension modifies base exponent.
   - Store dimension multiplier separately.
   - Visual theme switch triggered by dimension change.

7. Offline Income:
   - Store lastActiveTimestamp.
   - On app launch calculate delta.
   - Cap at 8 hours.
   - Apply offline multiplier.

8. Anti Cheat:
   - Validate tick income not exceeding 10x expected.
   - Validate prestige formula boundaries.
   - Disable leaderboard submission if anomaly.

9. Firebase Integration:
   - Firestore collection leaderboard.
   - Analytics events batched.
   - Error handling with retry policy.

10. StoreKit 2:
    - Verify transactions.
    - Handle subscription renewal state.
    - Restore purchases logic.

PERFORMANCE GUIDELINES:
- Avoid heavy loops inside Views.
- Cache multipliers.
- Avoid recomputing arrays each tick.
- All UI updates on main thread only.

TESTING REQUIREMENTS:
- Unit test income formula.
- Unit test prestige formula.
- Unit test cost growth.
- Unit test spin probability boundaries.
- Unit test merge logic.

SCALABILITY REQUIREMENTS:
- Unlimited future dimensions supported.
- Business list dynamic.
- Configurable balancing constants.
- Future multiplayer safe architecture.

UI RULES:
- TabView with 5 tabs only.
- Animated Singularity Eye in EmpireView.
- Business rows reusable component.
- Dark theme switchable by dimension.

ANALYTICS EVENTS TO TRACK:
session_start
session_end
first_purchase
prestige_triggered
dimension_unlocked
spin_used
jackpot_hit
subscription_started
merge_skin
upgrade_business
ai_level_up
soft_stuck_detected
cheat_detected

BUILD RULES:
- No force unwraps.
- No global mutable state.
- Zero compiler warnings.
- Document every economic formula.


SECTION 36 - DETAILED SYSTEM ARCHITECTURE BLOCK 36
--------------------------------------------------------------------------------

ARCHITECTURE RULES:
- Strict MVVM separation.
- All models Codable.
- No business logic in Views.
- All multipliers calculated in Services layer.
- Constants centralized in GameConstants.swift.

CORE GAME LOOP IMPLEMENTATION DETAILS:

1. Initialize GameViewModel with:
   - Timer publisher (1 second interval).
   - State loading from PersistenceService.
   - Dependency injection of Services.

2. Income Tick Processing:
   - Calculate global multiplier.
   - Iterate businesses.
   - Accumulate income.
   - Update lifetime earnings.
   - Evaluate soft stuck.
   - Trigger analytics event if threshold met.

3. Prestige Logic:
```

   - Calculate prestigeGain via formula.
   - Increment prestigeCount.
   - Reset business levels.
   - Reset coins.
   - Persist state.
   - Log event prestige_triggered.

4. Skin System Deep Logic:
   - Maintain unlockedSkins array.
   - Active skins limited to 3.
   - Multipliers combined multiplicatively.
   - Hard cap at 0.40 global.
   - Merge validation before tier upgrade.

5. Spin Wheel Randomization:
   - Use weighted random selection.
   - Store spin history count.
   - Apply soft pity adjustment dynamically.
   - Ensure deterministic fairness.

6. Dimension Scaling:
   - Each dimension modifies base exponent.
   - Store dimension multiplier separately.
   - Visual theme switch triggered by dimension change.

7. Offline Income:
   - Store lastActiveTimestamp.
   - On app launch calculate delta.
   - Cap at 8 hours.
   - Apply offline multiplier.

8. Anti Cheat:
   - Validate tick income not exceeding 10x expected.
   - Validate prestige formula boundaries.
   - Disable leaderboard submission if anomaly.

9. Firebase Integration:
   - Firestore collection leaderboard.
   - Analytics events batched.
   - Error handling with retry policy.

10. StoreKit 2:
    - Verify transactions.
    - Handle subscription renewal state.
    - Restore purchases logic.

PERFORMANCE GUIDELINES:
- Avoid heavy loops inside Views.
- Cache multipliers.
- Avoid recomputing arrays each tick.
- All UI updates on main thread only.

TESTING REQUIREMENTS:
- Unit test income formula.
- Unit test prestige formula.
- Unit test cost growth.
- Unit test spin probability boundaries.
- Unit test merge logic.

SCALABILITY REQUIREMENTS:
- Unlimited future dimensions supported.
- Business list dynamic.
- Configurable balancing constants.
- Future multiplayer safe architecture.

UI RULES:
- TabView with 5 tabs only.
- Animated Singularity Eye in EmpireView.
- Business rows reusable component.
- Dark theme switchable by dimension.

ANALYTICS EVENTS TO TRACK:
session_start
session_end
first_purchase
prestige_triggered

```
dimension_unlocked
spin_used
jackpot_hit
subscription_started
merge_skin
upgrade_business
ai_level_up
soft_stuck_detected
cheat_detected

BUILD RULES:
- No force unwraps.
- No global mutable state.
- Zero compiler warnings.
- Document every economic formula.


SECTION 37 - DETAILED SYSTEM ARCHITECTURE BLOCK 37
--------------------------------------------------------------------------------

ARCHITECTURE RULES:
- Strict MVVM separation.
- All models Codable.
- No business logic in Views.
- All multipliers calculated in Services layer.
- Constants centralized in GameConstants.swift.

CORE GAME LOOP IMPLEMENTATION DETAILS:

1. Initialize GameViewModel with:
   - Timer publisher (1 second interval).
   - State loading from PersistenceService.
   - Dependency injection of Services.

2. Income Tick Processing:
   - Calculate global multiplier.
   - Iterate businesses.
   - Accumulate income.
   - Update lifetime earnings.
   - Evaluate soft stuck.
   - Trigger analytics event if threshold met.

3. Prestige Logic:
   - Calculate prestigeGain via formula.
   - Increment prestigeCount.
   - Reset business levels.
   - Reset coins.
   - Persist state.
   - Log event prestige_triggered.

4. Skin System Deep Logic:
   - Maintain unlockedSkins array.
   - Active skins limited to 3.
   - Multipliers combined multiplicatively.
   - Hard cap at 0.40 global.
   - Merge validation before tier upgrade.

5. Spin Wheel Randomization:
   - Use weighted random selection.
   - Store spin history count.
   - Apply soft pity adjustment dynamically.
   - Ensure deterministic fairness.

6. Dimension Scaling:
   - Each dimension modifies base exponent.
   - Store dimension multiplier separately.
   - Visual theme switch triggered by dimension change.

7. Offline Income:
   - Store lastActiveTimestamp.
   - On app launch calculate delta.
   - Cap at 8 hours.
   - Apply offline multiplier.

8. Anti Cheat:
   - Validate tick income not exceeding 10x expected.
```

   - Validate prestige formula boundaries.
   - Disable leaderboard submission if anomaly.

9. Firebase Integration:
   - Firestore collection leaderboard.
   - Analytics events batched.
   - Error handling with retry policy.

10. StoreKit 2:
    - Verify transactions.
    - Handle subscription renewal state.
    - Restore purchases logic.

PERFORMANCE GUIDELINES:
- Avoid heavy loops inside Views.
- Cache multipliers.
- Avoid recomputing arrays each tick.
- All UI updates on main thread only.

TESTING REQUIREMENTS:
- Unit test income formula.
- Unit test prestige formula.
- Unit test cost growth.
- Unit test spin probability boundaries.
- Unit test merge logic.

SCALABILITY REQUIREMENTS:
- Unlimited future dimensions supported.
- Business list dynamic.
- Configurable balancing constants.
- Future multiplayer safe architecture.

UI RULES:
- TabView with 5 tabs only.
- Animated Singularity Eye in EmpireView.
- Business rows reusable component.
- Dark theme switchable by dimension.

ANALYTICS EVENTS TO TRACK:
session_start
session_end
first_purchase
prestige_triggered
dimension_unlocked
spin_used
jackpot_hit
subscription_started
merge_skin
upgrade_business
ai_level_up
soft_stuck_detected
cheat_detected

BUILD RULES:
- No force unwraps.
- No global mutable state.
- Zero compiler warnings.
- Document every economic formula.

SECTION 38 - DETAILED SYSTEM ARCHITECTURE BLOCK 38

ARCHITECTURE RULES:
- Strict MVVM separation.
- All models Codable.
- No business logic in Views.
- All multipliers calculated in Services layer.
- Constants centralized in GameConstants.swift.

CORE GAME LOOP IMPLEMENTATION DETAILS:

1. Initialize GameViewModel with:
   - Timer publisher (1 second interval).
   - State loading from PersistenceService.
   - Dependency injection of Services.

2. Income Tick Processing:
   - Calculate global multiplier.
   - Iterate businesses.
   - Accumulate income.
   - Update lifetime earnings.
   - Evaluate soft stuck.
   - Trigger analytics event if threshold met.

3. Prestige Logic:
   - Calculate prestigeGain via formula.
   - Increment prestigeCount.
   - Reset business levels.
   - Reset coins.
   - Persist state.
   - Log event prestige_triggered.

4. Skin System Deep Logic:
   - Maintain unlockedSkins array.
   - Active skins limited to 3.
   - Multipliers combined multiplicatively.
   - Hard cap at 0.40 global.
   - Merge validation before tier upgrade.

5. Spin Wheel Randomization:
   - Use weighted random selection.
   - Store spin history count.
   - Apply soft pity adjustment dynamically.
   - Ensure deterministic fairness.

6. Dimension Scaling:
   - Each dimension modifies base exponent.
   - Store dimension multiplier separately.
   - Visual theme switch triggered by dimension change.

7. Offline Income:
   - Store lastActiveTimestamp.
   - On app launch calculate delta.
   - Cap at 8 hours.
   - Apply offline multiplier.

8. Anti Cheat:
   - Validate tick income not exceeding 10x expected.
   - Validate prestige formula boundaries.
   - Disable leaderboard submission if anomaly.

9. Firebase Integration:
   - Firestore collection leaderboard.
   - Analytics events batched.
   - Error handling with retry policy.

10. StoreKit 2:
    - Verify transactions.
    - Handle subscription renewal state.
    - Restore purchases logic.

PERFORMANCE GUIDELINES:
- Avoid heavy loops inside Views.
- Cache multipliers.
- Avoid recomputing arrays each tick.
- All UI updates on main thread only.

TESTING REQUIREMENTS:
- Unit test income formula.
- Unit test prestige formula.
- Unit test cost growth.
- Unit test spin probability boundaries.
- Unit test merge logic.

SCALABILITY REQUIREMENTS:
- Unlimited future dimensions supported.
- Business list dynamic.
- Configurable balancing constants.
- Future multiplayer safe architecture.

UI RULES:
- TabView with 5 tabs only.

```
- Animated Singularity Eye in EmpireView.
- Business rows reusable component.
- Dark theme switchable by dimension.

ANALYTICS EVENTS TO TRACK:
session_start
session_end
first_purchase
prestige_triggered
dimension_unlocked
spin_used
jackpot_hit
subscription_started
merge_skin
upgrade_business
ai_level_up
soft_stuck_detected
cheat_detected

BUILD RULES:
- No force unwraps.
- No global mutable state.
- Zero compiler warnings.
- Document every economic formula.


SECTION 39 - DETAILED SYSTEM ARCHITECTURE BLOCK 39
--------------------------------------------------------------------------------

ARCHITECTURE RULES:
- Strict MVVM separation.
- All models Codable.
- No business logic in Views.
- All multipliers calculated in Services layer.
- Constants centralized in GameConstants.swift.

CORE GAME LOOP IMPLEMENTATION DETAILS:

1. Initialize GameViewModel with:
   - Timer publisher (1 second interval).
   - State loading from PersistenceService.
   - Dependency injection of Services.

2. Income Tick Processing:
   - Calculate global multiplier.
   - Iterate businesses.
   - Accumulate income.
   - Update lifetime earnings.
   - Evaluate soft stuck.
   - Trigger analytics event if threshold met.

3. Prestige Logic:
   - Calculate prestigeGain via formula.
   - Increment prestigeCount.
   - Reset business levels.
   - Reset coins.
   - Persist state.
   - Log event prestige_triggered.

4. Skin System Deep Logic:
   - Maintain unlockedSkins array.
   - Active skins limited to 3.
   - Multipliers combined multiplicatively.
   - Hard cap at 0.40 global.
   - Merge validation before tier upgrade.

5. Spin Wheel Randomization:
   - Use weighted random selection.
   - Store spin history count.
   - Apply soft pity adjustment dynamically.
   - Ensure deterministic fairness.

6. Dimension Scaling:
   - Each dimension modifies base exponent.
   - Store dimension multiplier separately.
   - Visual theme switch triggered by dimension change.
```

```
7. Offline Income:
   - Store lastActiveTimestamp.
   - On app launch calculate delta.
   - Cap at 8 hours.
   - Apply offline multiplier.

8. Anti Cheat:
   - Validate tick income not exceeding 10x expected.
   - Validate prestige formula boundaries.
   - Disable leaderboard submission if anomaly.

9. Firebase Integration:
   - Firestore collection leaderboard.
   - Analytics events batched.
   - Error handling with retry policy.

10. StoreKit 2:
    - Verify transactions.
    - Handle subscription renewal state.
    - Restore purchases logic.

PERFORMANCE GUIDELINES:
- Avoid heavy loops inside Views.
- Cache multipliers.
- Avoid recomputing arrays each tick.
- All UI updates on main thread only.

TESTING REQUIREMENTS:
- Unit test income formula.
- Unit test prestige formula.
- Unit test cost growth.
- Unit test spin probability boundaries.
- Unit test merge logic.

SCALABILITY REQUIREMENTS:
- Unlimited future dimensions supported.
- Business list dynamic.
- Configurable balancing constants.
- Future multiplayer safe architecture.

UI RULES:
- TabView with 5 tabs only.
- Animated Singularity Eye in EmpireView.
- Business rows reusable component.
- Dark theme switchable by dimension.

ANALYTICS EVENTS TO TRACK:
session_start
session_end
first_purchase
prestige_triggered
dimension_unlocked
spin_used
jackpot_hit
subscription_started
merge_skin
upgrade_business
ai_level_up
soft_stuck_detected
cheat_detected

BUILD RULES:
- No force unwraps.
- No global mutable state.
- Zero compiler warnings.
- Document every economic formula.


SECTION 40 - DETAILED SYSTEM ARCHITECTURE BLOCK 40
--------------------------------------------------------------------------------

ARCHITECTURE RULES:
- Strict MVVM separation.
- All models Codable.
- No business logic in Views.
- All multipliers calculated in Services layer.
```

- Constants centralized in GameConstants.swift.

CORE GAME LOOP IMPLEMENTATION DETAILS:

1. Initialize GameViewModel with:
   - Timer publisher (1 second interval).
   - State loading from PersistenceService.
   - Dependency injection of Services.

2. Income Tick Processing:
   - Calculate global multiplier.
   - Iterate businesses.
   - Accumulate income.
   - Update lifetime earnings.
   - Evaluate soft stuck.
   - Trigger analytics event if threshold met.

3. Prestige Logic:
   - Calculate prestigeGain via formula.
   - Increment prestigeCount.
   - Reset business levels.
   - Reset coins.
   - Persist state.
   - Log event prestige_triggered.

4. Skin System Deep Logic:
   - Maintain unlockedSkins array.
   - Active skins limited to 3.
   - Multipliers combined multiplicatively.
   - Hard cap at 0.40 global.
   - Merge validation before tier upgrade.

5. Spin Wheel Randomization:
   - Use weighted random selection.
   - Store spin history count.
   - Apply soft pity adjustment dynamically.
   - Ensure deterministic fairness.

6. Dimension Scaling:
   - Each dimension modifies base exponent.
   - Store dimension multiplier separately.
   - Visual theme switch triggered by dimension change.

7. Offline Income:
   - Store lastActiveTimestamp.
   - On app launch calculate delta.
   - Cap at 8 hours.
   - Apply offline multiplier.

8. Anti Cheat:
   - Validate tick income not exceeding 10x expected.
   - Validate prestige formula boundaries.
   - Disable leaderboard submission if anomaly.

9. Firebase Integration:
   - Firestore collection leaderboard.
   - Analytics events batched.
   - Error handling with retry policy.

10. StoreKit 2:
    - Verify transactions.
    - Handle subscription renewal state.
    - Restore purchases logic.

PERFORMANCE GUIDELINES:
- Avoid heavy loops inside Views.
- Cache multipliers.
- Avoid recomputing arrays each tick.
- All UI updates on main thread only.

TESTING REQUIREMENTS:
- Unit test income formula.
- Unit test prestige formula.
- Unit test cost growth.
- Unit test spin probability boundaries.
- Unit test merge logic.

```
SCALABILITY REQUIREMENTS:
- Unlimited future dimensions supported.
- Business list dynamic.
- Configurable balancing constants.
- Future multiplayer safe architecture.

UI RULES:
- TabView with 5 tabs only.
- Animated Singularity Eye in EmpireView.
- Business rows reusable component.
- Dark theme switchable by dimension.

ANALYTICS EVENTS TO TRACK:
session_start
session_end
first_purchase
prestige_triggered
dimension_unlocked
spin_used
jackpot_hit
subscription_started
merge_skin
upgrade_business
ai_level_up
soft_stuck_detected
cheat_detected

BUILD RULES:
- No force unwraps.
- No global mutable state.
- Zero compiler warnings.
- Document every economic formula.


SECTION 41 - DETAILED SYSTEM ARCHITECTURE BLOCK 41
--------------------------------------------------------------------------------

ARCHITECTURE RULES:
- Strict MVVM separation.
- All models Codable.
- No business logic in Views.
- All multipliers calculated in Services layer.
- Constants centralized in GameConstants.swift.

CORE GAME LOOP IMPLEMENTATION DETAILS:

1. Initialize GameViewModel with:
   - Timer publisher (1 second interval).
   - State loading from PersistenceService.
   - Dependency injection of Services.

2. Income Tick Processing:
   - Calculate global multiplier.
   - Iterate businesses.
   - Accumulate income.
   - Update lifetime earnings.
   - Evaluate soft stuck.
   - Trigger analytics event if threshold met.

3. Prestige Logic:
   - Calculate prestigeGain via formula.
   - Increment prestigeCount.
   - Reset business levels.
   - Reset coins.
   - Persist state.
   - Log event prestige_triggered.

4. Skin System Deep Logic:
   - Maintain unlockedSkins array.
   - Active skins limited to 3.
   - Multipliers combined multiplicatively.
   - Hard cap at 0.40 global.
   - Merge validation before tier upgrade.

5. Spin Wheel Randomization:
   - Use weighted random selection.
   - Store spin history count.
   - Apply soft pity adjustment dynamically.
   - Ensure deterministic fairness.

6. Dimension Scaling:
   - Each dimension modifies base exponent.
   - Store dimension multiplier separately.
   - Visual theme switch triggered by dimension change.

7. Offline Income:
   - Store lastActiveTimestamp.
   - On app launch calculate delta.
   - Cap at 8 hours.
   - Apply offline multiplier.

8. Anti Cheat:
   - Validate tick income not exceeding 10x expected.
   - Validate prestige formula boundaries.
   - Disable leaderboard submission if anomaly.

9. Firebase Integration:
   - Firestore collection leaderboard.
   - Analytics events batched.
   - Error handling with retry policy.

10. StoreKit 2:
    - Verify transactions.
    - Handle subscription renewal state.
    - Restore purchases logic.

PERFORMANCE GUIDELINES:
- Avoid heavy loops inside Views.
- Cache multipliers.
- Avoid recomputing arrays each tick.
- All UI updates on main thread only.

TESTING REQUIREMENTS:
- Unit test income formula.
- Unit test prestige formula.
- Unit test cost growth.
- Unit test spin probability boundaries.
- Unit test merge logic.

SCALABILITY REQUIREMENTS:
- Unlimited future dimensions supported.
- Business list dynamic.
- Configurable balancing constants.
- Future multiplayer safe architecture.

UI RULES:
- TabView with 5 tabs only.
- Animated Singularity Eye in EmpireView.
- Business rows reusable component.
- Dark theme switchable by dimension.

ANALYTICS EVENTS TO TRACK:
session_start
session_end
first_purchase
prestige_triggered
dimension_unlocked
spin_used
jackpot_hit
subscription_started
merge_skin
upgrade_business
ai_level_up
soft_stuck_detected
cheat_detected

BUILD RULES:
- No force unwraps.
- No global mutable state.
- Zero compiler warnings.
- Document every economic formula.
```

SECTION 42 - DETAILED SYSTEM ARCHITECTURE BLOCK 42

ARCHITECTURE RULES:
- Strict MVVM separation.
- All models Codable.
- No business logic in Views.
- All multipliers calculated in Services layer.
- Constants centralized in GameConstants.swift.

CORE GAME LOOP IMPLEMENTATION DETAILS:

1. Initialize GameViewModel with:
   - Timer publisher (1 second interval).
   - State loading from PersistenceService.
   - Dependency injection of Services.

2. Income Tick Processing:
   - Calculate global multiplier.
   - Iterate businesses.
   - Accumulate income.
   - Update lifetime earnings.
   - Evaluate soft stuck.
   - Trigger analytics event if threshold met.

3. Prestige Logic:
   - Calculate prestigeGain via formula.
   - Increment prestigeCount.
   - Reset business levels.
   - Reset coins.
   - Persist state.
   - Log event prestige_triggered.

4. Skin System Deep Logic:
   - Maintain unlockedSkins array.
   - Active skins limited to 3.
   - Multipliers combined multiplicatively.
   - Hard cap at 0.40 global.
   - Merge validation before tier upgrade.

5. Spin Wheel Randomization:
   - Use weighted random selection.
   - Store spin history count.
   - Apply soft pity adjustment dynamically.
   - Ensure deterministic fairness.

6. Dimension Scaling:
   - Each dimension modifies base exponent.
   - Store dimension multiplier separately.
   - Visual theme switch triggered by dimension change.

7. Offline Income:
   - Store lastActiveTimestamp.
   - On app launch calculate delta.
   - Cap at 8 hours.
   - Apply offline multiplier.

8. Anti Cheat:
   - Validate tick income not exceeding 10x expected.
   - Validate prestige formula boundaries.
   - Disable leaderboard submission if anomaly.

9. Firebase Integration:
   - Firestore collection leaderboard.
   - Analytics events batched.
   - Error handling with retry policy.

10. StoreKit 2:
    - Verify transactions.
    - Handle subscription renewal state.
    - Restore purchases logic.

PERFORMANCE GUIDELINES:
- Avoid heavy loops inside Views.
- Cache multipliers.
- Avoid recomputing arrays each tick.

```
- All UI updates on main thread only.

TESTING REQUIREMENTS:
- Unit test income formula.
- Unit test prestige formula.
- Unit test cost growth.
- Unit test spin probability boundaries.
- Unit test merge logic.

SCALABILITY REQUIREMENTS:
- Unlimited future dimensions supported.
- Business list dynamic.
- Configurable balancing constants.
- Future multiplayer safe architecture.

UI RULES:
- TabView with 5 tabs only.
- Animated Singularity Eye in EmpireView.
- Business rows reusable component.
- Dark theme switchable by dimension.

ANALYTICS EVENTS TO TRACK:
session_start
session_end
first_purchase
prestige_triggered
dimension_unlocked
spin_used
jackpot_hit
subscription_started
merge_skin
upgrade_business
ai_level_up
soft_stuck_detected
cheat_detected

BUILD RULES:
- No force unwraps.
- No global mutable state.
- Zero compiler warnings.
- Document every economic formula.


SECTION 43 - DETAILED SYSTEM ARCHITECTURE BLOCK 43
--------------------------------------------------------------------------------

ARCHITECTURE RULES:
- Strict MVVM separation.
- All models Codable.
- No business logic in Views.
- All multipliers calculated in Services layer.
- Constants centralized in GameConstants.swift.

CORE GAME LOOP IMPLEMENTATION DETAILS:

1. Initialize GameViewModel with:
   - Timer publisher (1 second interval).
   - State loading from PersistenceService.
   - Dependency injection of Services.

2. Income Tick Processing:
   - Calculate global multiplier.
   - Iterate businesses.
   - Accumulate income.
   - Update lifetime earnings.
   - Evaluate soft stuck.
   - Trigger analytics event if threshold met.

3. Prestige Logic:
   - Calculate prestigeGain via formula.
   - Increment prestigeCount.
   - Reset business levels.
   - Reset coins.
   - Persist state.
   - Log event prestige_triggered.
```

4. Skin System Deep Logic:
   - Maintain unlockedSkins array.
   - Active skins limited to 3.
   - Multipliers combined multiplicatively.
   - Hard cap at 0.40 global.
   - Merge validation before tier upgrade.

5. Spin Wheel Randomization:
   - Use weighted random selection.
   - Store spin history count.
   - Apply soft pity adjustment dynamically.
   - Ensure deterministic fairness.

6. Dimension Scaling:
   - Each dimension modifies base exponent.
   - Store dimension multiplier separately.
   - Visual theme switch triggered by dimension change.

7. Offline Income:
   - Store lastActiveTimestamp.
   - On app launch calculate delta.
   - Cap at 8 hours.
   - Apply offline multiplier.

8. Anti Cheat:
   - Validate tick income not exceeding 10x expected.
   - Validate prestige formula boundaries.
   - Disable leaderboard submission if anomaly.

9. Firebase Integration:
   - Firestore collection leaderboard.
   - Analytics events batched.
   - Error handling with retry policy.

10. StoreKit 2:
    - Verify transactions.
    - Handle subscription renewal state.
    - Restore purchases logic.

PERFORMANCE GUIDELINES:
- Avoid heavy loops inside Views.
- Cache multipliers.
- Avoid recomputing arrays each tick.
- All UI updates on main thread only.

TESTING REQUIREMENTS:
- Unit test income formula.
- Unit test prestige formula.
- Unit test cost growth.
- Unit test spin probability boundaries.
- Unit test merge logic.

SCALABILITY REQUIREMENTS:
- Unlimited future dimensions supported.
- Business list dynamic.
- Configurable balancing constants.
- Future multiplayer safe architecture.

UI RULES:
- TabView with 5 tabs only.
- Animated Singularity Eye in EmpireView.
- Business rows reusable component.
- Dark theme switchable by dimension.

ANALYTICS EVENTS TO TRACK:
session_start
session_end
first_purchase
prestige_triggered
dimension_unlocked
spin_used
jackpot_hit
subscription_started
merge_skin
upgrade_business
ai_level_up

```
soft_stuck_detected
cheat_detected

BUILD RULES:
- No force unwraps.
- No global mutable state.
- Zero compiler warnings.
- Document every economic formula.


SECTION 44 - DETAILED SYSTEM ARCHITECTURE BLOCK 44
--------------------------------------------------------------------------------

ARCHITECTURE RULES:
- Strict MVVM separation.
- All models Codable.
- No business logic in Views.
- All multipliers calculated in Services layer.
- Constants centralized in GameConstants.swift.

CORE GAME LOOP IMPLEMENTATION DETAILS:

1. Initialize GameViewModel with:
   - Timer publisher (1 second interval).
   - State loading from PersistenceService.
   - Dependency injection of Services.

2. Income Tick Processing:
   - Calculate global multiplier.
   - Iterate businesses.
   - Accumulate income.
   - Update lifetime earnings.
   - Evaluate soft stuck.
   - Trigger analytics event if threshold met.

3. Prestige Logic:
   - Calculate prestigeGain via formula.
   - Increment prestigeCount.
   - Reset business levels.
   - Reset coins.
   - Persist state.
   - Log event prestige_triggered.

4. Skin System Deep Logic:
   - Maintain unlockedSkins array.
   - Active skins limited to 3.
   - Multipliers combined multiplicatively.
   - Hard cap at 0.40 global.
   - Merge validation before tier upgrade.

5. Spin Wheel Randomization:
   - Use weighted random selection.
   - Store spin history count.
   - Apply soft pity adjustment dynamically.
   - Ensure deterministic fairness.

6. Dimension Scaling:
   - Each dimension modifies base exponent.
   - Store dimension multiplier separately.
   - Visual theme switch triggered by dimension change.

7. Offline Income:
   - Store lastActiveTimestamp.
   - On app launch calculate delta.
   - Cap at 8 hours.
   - Apply offline multiplier.

8. Anti Cheat:
   - Validate tick income not exceeding 10x expected.
   - Validate prestige formula boundaries.
   - Disable leaderboard submission if anomaly.

9. Firebase Integration:
   - Firestore collection leaderboard.
   - Analytics events batched.
   - Error handling with retry policy.
```

```
10. StoreKit 2:
    - Verify transactions.
    - Handle subscription renewal state.
    - Restore purchases logic.

PERFORMANCE GUIDELINES:
- Avoid heavy loops inside Views.
- Cache multipliers.
- Avoid recomputing arrays each tick.
- All UI updates on main thread only.

TESTING REQUIREMENTS:
- Unit test income formula.
- Unit test prestige formula.
- Unit test cost growth.
- Unit test spin probability boundaries.
- Unit test merge logic.

SCALABILITY REQUIREMENTS:
- Unlimited future dimensions supported.
- Business list dynamic.
- Configurable balancing constants.
- Future multiplayer safe architecture.

UI RULES:
- TabView with 5 tabs only.
- Animated Singularity Eye in EmpireView.
- Business rows reusable component.
- Dark theme switchable by dimension.

ANALYTICS EVENTS TO TRACK:
session_start
session_end
first_purchase
prestige_triggered
dimension_unlocked
spin_used
jackpot_hit
subscription_started
merge_skin
upgrade_business
ai_level_up
soft_stuck_detected
cheat_detected

BUILD RULES:
- No force unwraps.
- No global mutable state.
- Zero compiler warnings.
- Document every economic formula.


SECTION 45 - DETAILED SYSTEM ARCHITECTURE BLOCK 45
--------------------------------------------------------------------------------

ARCHITECTURE RULES:
- Strict MVVM separation.
- All models Codable.
- No business logic in Views.
- All multipliers calculated in Services layer.
- Constants centralized in GameConstants.swift.

CORE GAME LOOP IMPLEMENTATION DETAILS:

1. Initialize GameViewModel with:
   - Timer publisher (1 second interval).
   - State loading from PersistenceService.
   - Dependency injection of Services.

2. Income Tick Processing:
   - Calculate global multiplier.
   - Iterate businesses.
   - Accumulate income.
   - Update lifetime earnings.
   - Evaluate soft stuck.
   - Trigger analytics event if threshold met.
```

```
3. Prestige Logic:
   - Calculate prestigeGain via formula.
   - Increment prestigeCount.
   - Reset business levels.
   - Reset coins.
   - Persist state.
   - Log event prestige_triggered.

4. Skin System Deep Logic:
   - Maintain unlockedSkins array.
   - Active skins limited to 3.
   - Multipliers combined multiplicatively.
   - Hard cap at 0.40 global.
   - Merge validation before tier upgrade.

5. Spin Wheel Randomization:
   - Use weighted random selection.
   - Store spin history count.
   - Apply soft pity adjustment dynamically.
   - Ensure deterministic fairness.

6. Dimension Scaling:
   - Each dimension modifies base exponent.
   - Store dimension multiplier separately.
   - Visual theme switch triggered by dimension change.

7. Offline Income:
   - Store lastActiveTimestamp.
   - On app launch calculate delta.
   - Cap at 8 hours.
   - Apply offline multiplier.

8. Anti Cheat:
   - Validate tick income not exceeding 10x expected.
   - Validate prestige formula boundaries.
   - Disable leaderboard submission if anomaly.

9. Firebase Integration:
   - Firestore collection leaderboard.
   - Analytics events batched.
   - Error handling with retry policy.

10. StoreKit 2:
    - Verify transactions.
    - Handle subscription renewal state.
    - Restore purchases logic.

PERFORMANCE GUIDELINES:
- Avoid heavy loops inside Views.
- Cache multipliers.
- Avoid recomputing arrays each tick.
- All UI updates on main thread only.

TESTING REQUIREMENTS:
- Unit test income formula.
- Unit test prestige formula.
- Unit test cost growth.
- Unit test spin probability boundaries.
- Unit test merge logic.

SCALABILITY REQUIREMENTS:
- Unlimited future dimensions supported.
- Business list dynamic.
- Configurable balancing constants.
- Future multiplayer safe architecture.

UI RULES:
- TabView with 5 tabs only.
- Animated Singularity Eye in EmpireView.
- Business rows reusable component.
- Dark theme switchable by dimension.

ANALYTICS EVENTS TO TRACK:
session_start
session_end
first_purchase
```

```
prestige_triggered
dimension_unlocked
spin_used
jackpot_hit
subscription_started
merge_skin
upgrade_business
ai_level_up
soft_stuck_detected
cheat_detected

BUILD RULES:
- No force unwraps.
- No global mutable state.
- Zero compiler warnings.
- Document every economic formula.


SECTION 46 - DETAILED SYSTEM ARCHITECTURE BLOCK 46
--------------------------------------------------------------------------------

ARCHITECTURE RULES:
- Strict MVVM separation.
- All models Codable.
- No business logic in Views.
- All multipliers calculated in Services layer.
- Constants centralized in GameConstants.swift.

CORE GAME LOOP IMPLEMENTATION DETAILS:

1. Initialize GameViewModel with:
   - Timer publisher (1 second interval).
   - State loading from PersistenceService.
   - Dependency injection of Services.

2. Income Tick Processing:
   - Calculate global multiplier.
   - Iterate businesses.
   - Accumulate income.
   - Update lifetime earnings.
   - Evaluate soft stuck.
   - Trigger analytics event if threshold met.

3. Prestige Logic:
   - Calculate prestigeGain via formula.
   - Increment prestigeCount.
   - Reset business levels.
   - Reset coins.
   - Persist state.
   - Log event prestige_triggered.

4. Skin System Deep Logic:
   - Maintain unlockedSkins array.
   - Active skins limited to 3.
   - Multipliers combined multiplicatively.
   - Hard cap at 0.40 global.
   - Merge validation before tier upgrade.

5. Spin Wheel Randomization:
   - Use weighted random selection.
   - Store spin history count.
   - Apply soft pity adjustment dynamically.
   - Ensure deterministic fairness.

6. Dimension Scaling:
   - Each dimension modifies base exponent.
   - Store dimension multiplier separately.
   - Visual theme switch triggered by dimension change.

7. Offline Income:
   - Store lastActiveTimestamp.
   - On app launch calculate delta.
   - Cap at 8 hours.
   - Apply offline multiplier.

8. Anti Cheat:
```

   - Validate tick income not exceeding 10x expected.
   - Validate prestige formula boundaries.
   - Disable leaderboard submission if anomaly.

9. Firebase Integration:
   - Firestore collection leaderboard.
   - Analytics events batched.
   - Error handling with retry policy.

10. StoreKit 2:
    - Verify transactions.
    - Handle subscription renewal state.
    - Restore purchases logic.

PERFORMANCE GUIDELINES:
- Avoid heavy loops inside Views.
- Cache multipliers.
- Avoid recomputing arrays each tick.
- All UI updates on main thread only.

TESTING REQUIREMENTS:
- Unit test income formula.
- Unit test prestige formula.
- Unit test cost growth.
- Unit test spin probability boundaries.
- Unit test merge logic.

SCALABILITY REQUIREMENTS:
- Unlimited future dimensions supported.
- Business list dynamic.
- Configurable balancing constants.
- Future multiplayer safe architecture.

UI RULES:
- TabView with 5 tabs only.
- Animated Singularity Eye in EmpireView.
- Business rows reusable component.
- Dark theme switchable by dimension.

ANALYTICS EVENTS TO TRACK:

session_start
session_end
first_purchase
prestige_triggered
dimension_unlocked
spin_used
jackpot_hit
subscription_started
merge_skin
upgrade_business
ai_level_up
soft_stuck_detected
cheat_detected

BUILD RULES:
- No force unwraps.
- No global mutable state.
- Zero compiler warnings.
- Document every economic formula.

SECTION 47 - DETAILED SYSTEM ARCHITECTURE BLOCK 47

ARCHITECTURE RULES:
- Strict MVVM separation.
- All models Codable.
- No business logic in Views.
- All multipliers calculated in Services layer.
- Constants centralized in GameConstants.swift.

CORE GAME LOOP IMPLEMENTATION DETAILS:

1. Initialize GameViewModel with:
   - Timer publisher (1 second interval).
   - State loading from PersistenceService.

```
   - Dependency injection of Services.

2. Income Tick Processing:
   - Calculate global multiplier.
   - Iterate businesses.
   - Accumulate income.
   - Update lifetime earnings.
   - Evaluate soft stuck.
   - Trigger analytics event if threshold met.

3. Prestige Logic:
   - Calculate prestigeGain via formula.
   - Increment prestigeCount.
   - Reset business levels.
   - Reset coins.
   - Persist state.
   - Log event prestige_triggered.

4. Skin System Deep Logic:
   - Maintain unlockedSkins array.
   - Active skins limited to 3.
   - Multipliers combined multiplicatively.
   - Hard cap at 0.40 global.
   - Merge validation before tier upgrade.

5. Spin Wheel Randomization:
   - Use weighted random selection.
   - Store spin history count.
   - Apply soft pity adjustment dynamically.
   - Ensure deterministic fairness.

6. Dimension Scaling:
   - Each dimension modifies base exponent.
   - Store dimension multiplier separately.
   - Visual theme switch triggered by dimension change.

7. Offline Income:
   - Store lastActiveTimestamp.
   - On app launch calculate delta.
   - Cap at 8 hours.
   - Apply offline multiplier.

8. Anti Cheat:
   - Validate tick income not exceeding 10x expected.
   - Validate prestige formula boundaries.
   - Disable leaderboard submission if anomaly.

9. Firebase Integration:
   - Firestore collection leaderboard.
   - Analytics events batched.
   - Error handling with retry policy.

10. StoreKit 2:
    - Verify transactions.
    - Handle subscription renewal state.
    - Restore purchases logic.

PERFORMANCE GUIDELINES:
- Avoid heavy loops inside Views.
- Cache multipliers.
- Avoid recomputing arrays each tick.
- All UI updates on main thread only.

TESTING REQUIREMENTS:
- Unit test income formula.
- Unit test prestige formula.
- Unit test cost growth.
- Unit test spin probability boundaries.
- Unit test merge logic.

SCALABILITY REQUIREMENTS:
- Unlimited future dimensions supported.
- Business list dynamic.
- Configurable balancing constants.
- Future multiplayer safe architecture.
```

UI RULES:
- TabView with 5 tabs only.
- Animated Singularity Eye in EmpireView.
- Business rows reusable component.
- Dark theme switchable by dimension.

ANALYTICS EVENTS TO TRACK:
session_start
session_end
first_purchase
prestige_triggered
dimension_unlocked
spin_used
jackpot_hit
subscription_started
merge_skin
upgrade_business
ai_level_up
soft_stuck_detected
cheat_detected

BUILD RULES:
- No force unwraps.
- No global mutable state.
- Zero compiler warnings.
- Document every economic formula.

SECTION 48 - DETAILED SYSTEM ARCHITECTURE BLOCK 48

ARCHITECTURE RULES:
- Strict MVVM separation.
- All models Codable.
- No business logic in Views.
- All multipliers calculated in Services layer.
- Constants centralized in GameConstants.swift.

CORE GAME LOOP IMPLEMENTATION DETAILS:

1. Initialize GameViewModel with:
   - Timer publisher (1 second interval).
   - State loading from PersistenceService.
   - Dependency injection of Services.

2. Income Tick Processing:
   - Calculate global multiplier.
   - Iterate businesses.
   - Accumulate income.
   - Update lifetime earnings.
   - Evaluate soft stuck.
   - Trigger analytics event if threshold met.

3. Prestige Logic:
   - Calculate prestigeGain via formula.
   - Increment prestigeCount.
   - Reset business levels.
   - Reset coins.
   - Persist state.
   - Log event prestige_triggered.

4. Skin System Deep Logic:
   - Maintain unlockedSkins array.
   - Active skins limited to 3.
   - Multipliers combined multiplicatively.
   - Hard cap at 0.40 global.
   - Merge validation before tier upgrade.

5. Spin Wheel Randomization:
   - Use weighted random selection.
   - Store spin history count.
   - Apply soft pity adjustment dynamically.
   - Ensure deterministic fairness.

6. Dimension Scaling:
   - Each dimension modifies base exponent.

   - Store dimension multiplier separately.
   - Visual theme switch triggered by dimension change.

7. Offline Income:
   - Store lastActiveTimestamp.
   - On app launch calculate delta.
   - Cap at 8 hours.
   - Apply offline multiplier.

8. Anti Cheat:
   - Validate tick income not exceeding 10x expected.
   - Validate prestige formula boundaries.
   - Disable leaderboard submission if anomaly.

9. Firebase Integration:
   - Firestore collection leaderboard.
   - Analytics events batched.
   - Error handling with retry policy.

10. StoreKit 2:
    - Verify transactions.
    - Handle subscription renewal state.
    - Restore purchases logic.

PERFORMANCE GUIDELINES:
- Avoid heavy loops inside Views.
- Cache multipliers.
- Avoid recomputing arrays each tick.
- All UI updates on main thread only.

TESTING REQUIREMENTS:
- Unit test income formula.
- Unit test prestige formula.
- Unit test cost growth.
- Unit test spin probability boundaries.
- Unit test merge logic.

SCALABILITY REQUIREMENTS:
- Unlimited future dimensions supported.
- Business list dynamic.
- Configurable balancing constants.
- Future multiplayer safe architecture.

UI RULES:
- TabView with 5 tabs only.
- Animated Singularity Eye in EmpireView.
- Business rows reusable component.
- Dark theme switchable by dimension.

ANALYTICS EVENTS TO TRACK:
session_start
session_end
first_purchase
prestige_triggered
dimension_unlocked
spin_used
jackpot_hit
subscription_started
merge_skin
upgrade_business
ai_level_up
soft_stuck_detected
cheat_detected

BUILD RULES:
- No force unwraps.
- No global mutable state.
- Zero compiler warnings.
- Document every economic formula.

SECTION 49 - DETAILED SYSTEM ARCHITECTURE BLOCK 49

ARCHITECTURE RULES:
- Strict MVVM separation.

- All models Codable.
- No business logic in Views.
- All multipliers calculated in Services layer.
- Constants centralized in GameConstants.swift.

CORE GAME LOOP IMPLEMENTATION DETAILS:

1. Initialize GameViewModel with:
   - Timer publisher (1 second interval).
   - State loading from PersistenceService.
   - Dependency injection of Services.

2. Income Tick Processing:
   - Calculate global multiplier.
   - Iterate businesses.
   - Accumulate income.
   - Update lifetime earnings.
   - Evaluate soft stuck.
   - Trigger analytics event if threshold met.

3. Prestige Logic:
   - Calculate prestigeGain via formula.
   - Increment prestigeCount.
   - Reset business levels.
   - Reset coins.
   - Persist state.
   - Log event prestige_triggered.

4. Skin System Deep Logic:
   - Maintain unlockedSkins array.
   - Active skins limited to 3.
   - Multipliers combined multiplicatively.
   - Hard cap at 0.40 global.
   - Merge validation before tier upgrade.

5. Spin Wheel Randomization:
   - Use weighted random selection.
   - Store spin history count.
   - Apply soft pity adjustment dynamically.
   - Ensure deterministic fairness.

6. Dimension Scaling:
   - Each dimension modifies base exponent.
   - Store dimension multiplier separately.
   - Visual theme switch triggered by dimension change.

7. Offline Income:
   - Store lastActiveTimestamp.
   - On app launch calculate delta.
   - Cap at 8 hours.
   - Apply offline multiplier.

8. Anti Cheat:
   - Validate tick income not exceeding 10x expected.
   - Validate prestige formula boundaries.
   - Disable leaderboard submission if anomaly.

9. Firebase Integration:
   - Firestore collection leaderboard.
   - Analytics events batched.
   - Error handling with retry policy.

10. StoreKit 2:
    - Verify transactions.
    - Handle subscription renewal state.
    - Restore purchases logic.

PERFORMANCE GUIDELINES:
- Avoid heavy loops inside Views.
- Cache multipliers.
- Avoid recomputing arrays each tick.
- All UI updates on main thread only.

TESTING REQUIREMENTS:
- Unit test income formula.
- Unit test prestige formula.

- Unit test cost growth.
- Unit test spin probability boundaries.
- Unit test merge logic.

SCALABILITY REQUIREMENTS:
- Unlimited future dimensions supported.
- Business list dynamic.
- Configurable balancing constants.
- Future multiplayer safe architecture.

UI RULES:
- TabView with 5 tabs only.
- Animated Singularity Eye in EmpireView.
- Business rows reusable component.
- Dark theme switchable by dimension.

ANALYTICS EVENTS TO TRACK:
session_start
session_end
first_purchase
prestige_triggered
dimension_unlocked
spin_used
jackpot_hit
subscription_started
merge_skin
upgrade_business
ai_level_up
soft_stuck_detected
cheat_detected

BUILD RULES:
- No force unwraps.
- No global mutable state.
- Zero compiler warnings.
- Document every economic formula.

SECTION 50 - DETAILED SYSTEM ARCHITECTURE BLOCK 50

ARCHITECTURE RULES:
- Strict MVVM separation.
- All models Codable.
- No business logic in Views.
- All multipliers calculated in Services layer.
- Constants centralized in GameConstants.swift.

CORE GAME LOOP IMPLEMENTATION DETAILS:

1. Initialize GameViewModel with:
   - Timer publisher (1 second interval).
   - State loading from PersistenceService.
   - Dependency injection of Services.

2. Income Tick Processing:
   - Calculate global multiplier.
   - Iterate businesses.
   - Accumulate income.
   - Update lifetime earnings.
   - Evaluate soft stuck.
   - Trigger analytics event if threshold met.

3. Prestige Logic:
   - Calculate prestigeGain via formula.
   - Increment prestigeCount.
   - Reset business levels.
   - Reset coins.
   - Persist state.
   - Log event prestige_triggered.

4. Skin System Deep Logic:
   - Maintain unlockedSkins array.
   - Active skins limited to 3.
   - Multipliers combined multiplicatively.
   - Hard cap at 0.40 global.

```
   - Merge validation before tier upgrade.

5. Spin Wheel Randomization:
   - Use weighted random selection.
   - Store spin history count.
   - Apply soft pity adjustment dynamically.
   - Ensure deterministic fairness.

6. Dimension Scaling:
   - Each dimension modifies base exponent.
   - Store dimension multiplier separately.
   - Visual theme switch triggered by dimension change.

7. Offline Income:
   - Store lastActiveTimestamp.
   - On app launch calculate delta.
   - Cap at 8 hours.
   - Apply offline multiplier.

8. Anti Cheat:
   - Validate tick income not exceeding 10x expected.
   - Validate prestige formula boundaries.
   - Disable leaderboard submission if anomaly.

9. Firebase Integration:
   - Firestore collection leaderboard.
   - Analytics events batched.
   - Error handling with retry policy.

10. StoreKit 2:
    - Verify transactions.
    - Handle subscription renewal state.
    - Restore purchases logic.

PERFORMANCE GUIDELINES:
- Avoid heavy loops inside Views.
- Cache multipliers.
- Avoid recomputing arrays each tick.
- All UI updates on main thread only.

TESTING REQUIREMENTS:
- Unit test income formula.
- Unit test prestige formula.
- Unit test cost growth.
- Unit test spin probability boundaries.
- Unit test merge logic.

SCALABILITY REQUIREMENTS:
- Unlimited future dimensions supported.
- Business list dynamic.
- Configurable balancing constants.
- Future multiplayer safe architecture.

UI RULES:
- TabView with 5 tabs only.
- Animated Singularity Eye in EmpireView.
- Business rows reusable component.
- Dark theme switchable by dimension.

ANALYTICS EVENTS TO TRACK:
session_start
session_end
first_purchase
prestige_triggered
dimension_unlocked
spin_used
jackpot_hit
subscription_started
merge_skin
upgrade_business
ai_level_up
soft_stuck_detected
cheat_detected

BUILD RULES:
- No force unwraps.
```

- No global mutable state.
- Zero compiler warnings.
- Document every economic formula.

==
END OF COMPLETE 100 PAGE MASTER SPECIFICATION