Product Requirements Document (PRD)

Project Name: Kingslayer (Regicide TUI)

Version: 1.0 **Status:** Draft **Target Platform:** PC (Windows/Linux/Mac) via Terminal
Development Strategy: Antigravity (Iterative, Low-Friction, Modular)

1. Executive Summary

The goal is to build a high-fidelity, text-based implementation of the card game *Regicide*. The product will be a Terminal User Interface (TUI) application. Development will proceed in two distinct phases:

1. **Phase 1 (MVP):** A complete, playable Solo engine against the AI logic.
2. **Phase 2 (Expansion):** LAN-based multiplayer for 2-4 players using a Host-Client architecture with manual IP discovery.

Visual Constraint: Strict ASCII/Unicode text only. No graphical assets, no copyrighted character art.

2. Technical Stack Recommendations

- **Core Logic:** Python 3.10+ (Recommended for rapid prototyping) OR Rust (for performance/safety).
- **TUI Library:** `Textual` (Python) or `Ratatui` (Rust).
- **Networking:** Standard TCP Sockets (`socket` lib in Python or `Tokio` in Rust).
- **Data Serialization:** JSON (for game state exchange over LAN).

3. Functional Requirements: Phase 1 (Solo Engine)

3.1 Game Setup & Deck Construction

- **The Castle Deck (Enemy Deck):**
  - Must be constructed in layers: 4 Kings (bottom), 4 Queens (middle), 4 Jacks (top).
  - Suits within layers must be randomized.
  - *Current Enemy* is the top card of the Castle Deck.
- **The Tavern Deck (Player Deck):**
  - Contains Standard 52-card deck (A-10, J/Q/K removed) + Jesters.
  - **Jesters Count:** 0 (Solo/Standard), 2 (Co-op/Easy Mode).
- **Hand Management:**
  - Solo Max Hand Size: 8 cards.

3.2 Card Attributes

- **Suits & Powers:**
  - **Hearts (Heal):** Shuffle discard pile into Tavern deck (Amount = Attack Value).
  - **Diamonds (Draw):** Draw cards from Tavern deck (Amount = Attack Value).
  - **Clubs (Double Damage):** Attack Value counts x2 against enemy HP.
  - **Spades (Shield):** Reduce Enemy Attack Value for the current turn.
- **Ranks:**
  - **2-10:** Face value.
  - **Ace (Animal Companion):** Value 1. Can be played with any other card to combine values + activate both suits.
  - **Jester:** Value 0. Cancels enemy immunity. Skips enemy attack phase.
- **Enemy Stats:**
  - **Jack:** 20 HP, 10 Attack.
  - **Queen:** 30 HP, 15 Attack.
  - **King:** 40 HP, 20 Attack.

3.3 The Game Loop (Turn Logic)

1. **Input Phase:** Player selects card(s).
   - *Validation:* Single card, OR Pair/Triple/Quad of same rank (sum <= 10), OR Ace + Any Card.
2. **Resolution Phase:**
   - Check Immunity (Is Enemy suit == Played suit?). If yes, Suit Power is ignored (Damage still applies).
   - Apply Suit Powers (Shields update "Shield" buffer; Hearts/Diamonds modify decks).
   - Deal Damage to Enemy HP.
3. **Check Victory/Defeat:**
   - **Exact Damage:** If damage == Enemy Current HP, Enemy is "Captured" (placed on top of Tavern Deck face down).
   - **Overkill:** If damage > Enemy Current HP, Enemy is Discarded.
4. **Enemy Attack Phase:**
   - If Enemy HP > 0: Calculate Damage = (Enemy Attack - Active Shields).
   - **Input Request:** Player must discard cards from hand where Sum(Values) >= Damage.
   - *Failure Condition:* If player cannot discard enough, Game Over (Regicide).

3.4 TUI Layout (ASCII)

The screen should be divided into four borders/panes:

- **Top Pane (The Castle):** Displays the current Enemy Card (ASCII Art representation), current HP bar (e.g., `[||||||||||]`), and Attack stat.
- **Middle Pane (The Battlefield):** Displays currently played cards, active Shield value, and damage capability.
- **Bottom Pane (Hand):** Displays player's cards. Selected cards highlighted with `> <` or inverted colors.
- **Side Pane (Log):** Scrollable text log: *"Player played 5 of Hearts. Healed 5 cards."*

4. Functional Requirements: Phase 2 (LAN Multiplayer)

4.1 Network Architecture

- **Host (Player 1):**
  - Runs the "Game Logic Authority".
  - Listens on Port `5555`.
  - Displays local IP address in the lobby header (e.g., `Hosting at 192.168.1.5`).
- **Clients (Players 2-4):**
  - Connect via TCP to Host IP.
  - Send "Input Events" (Card Index Selected, Commit Turn).
  - Receive "State Objects" (Full JSON representation of the board).

4.2 Multiplayer Rules Engine

- **Scaling Difficulty:**
  - 2 Players: Max Hand 7, Jesters 0.
  - 3 Players: Max Hand 6, Jesters 1.
  - 4 Players: Max Hand 5, Jesters 2.
- **Turn Order:**
  - Clockwise rotation. Host tracks `current_player_index`.
  - TUI must indicate "Waiting for Player X..." when it is not the local user's turn.

4.3 Communication Protocol

- **Message Types:**
  - `HANDSHAKE`: Client sends Name; Host assigns Player ID.
  - `GAME_STATE`: Host sends full board (Enemy status, active cards, whose turn it is).
  - `PLAYER_ACTION`: Client sends `{ "action": "PLAY", "cards": ["Ace of Spades", "King of Hearts"] }`.
  - `ERROR`: Host sends "Invalid Move" notification.

5. Development Roadmap (The Build Steps)

Step 1: Data Structures & Core Logic

- Create classes/structs for `Card` , `Deck` , `Player` , `Enemy` .
- Implement `Deck.shuffle()` and `Castle.construct()` .

Step 2: The TUI Skeleton

- Initialize the TUI library.
- Create the 3-pane layout.
- Implement a "Card Renderer" that turns a card object into an ASCII box.

```
.-------.
| 5     |
|   ♥   |
|     5 |
'-------'
```

Step 3: Solo Game Loop Integration

- Connect Input (Arrow Keys/Enter) to Game Logic.
- Implement the "Discard to Survive" modal calculation.

Step 4: Network Layer

- Build the `Server` class (Host) and `Client` class.
- Implement the JSON serializer for the `GameState` .

Step 5: Polish

- Add color codes (Red for Hearts/Diamonds, Blue/White for Spades/Clubs).
- Add ASCII art for Kings/Queens/Jacks (Crowns, Swords).

6. Constraints & Edge Cases

- **No Copyright:** Do not scan or use images from the official board game. Use standard Unicode characters.
- **Crash Recovery:** If a client disconnects, the Host should pause the game.
- **Window Resizing:** The TUI should handle terminal resizing gracefully without breaking the layout.