**Functional Requirements Document (FRD): Card Clash Game**

**Project Title**

Card Clash Game - Multiplayer Android App

**Document Version**

v1.2 - June 21, 2025

**1. Overview**

Card Clash Game is an online multiplayer Android card game where players challenge each other using unique cricket-themed cards with statistical attributes. The goal is to strategically win cards and accumulate points by winning challenges based on selected stats.

**2. Objectives**

* Provide an engaging multiplayer card battle experience.
* Promote strategic thinking via stat-based card battles.
* Ensure fairness with time-bound decisions and clear scoring rules.

**3. Game Setup**

* **Players**: 2 to 6 (configurable)
* **Cards**: 50 to 100 unique cards. Each card contains Player Name, Runs, Wickets, Batting Average, Strike Rate, Matches Played, Centuries, and Five-Wicket Hauls.
* **Initial State**: Cards are shuffled and distributed equally among players. Each player sees only their top card face-up.
* **Player Identification**: Assigned automatically (Player 1, Player 2, etc.).

**3.1 Lobby and Game Start**

* The game features a lobby system where players can create or join a room using a unique room code.
* Each room has a configurable maximum number of players (between 2 and 6).
* When creating a room, the host specifies the maximum number of players.
* Players join the room using the provided room code.
* The server tracks the number of players in the room.
* Once the room is full (i.e., the number of joined players equals the maximum configured), the server initiates a 3-second countdown.
* During the countdown, the server sends countdown messages to all clients every second (e.g., "Game starts in 3...", "2...", "1...").
* After the countdown, the server sends a "gameStart" signal to all clients.
* Clients then transition from the lobby screen to the game play screen.
* The server initializes the game by:
  + Loading the card data from the database.
  + Shuffling the deck of 50 to 100 unique cards.
  + Distributing the cards equally among the players.
  + Assigning player identifiers (e.g., Player 1, Player 2, etc.).
  + Selecting the first active player.
  + Sending the initial game state to all clients, including each player's top card and other necessary information.

**4. Game States**

* **Initial State**: As described in section 3.
* **Player Identification**: As described in section 3.

**5. Game Flow**

* **Active Player Turn**:
  + The active player chooses a stat within 15 seconds. Timeout results in -2 points and card removal.
* **Opponent Response**:
  + Opponents decide to accept the challenge or give up within 15 seconds. Timeout results in -1 point and card removal.
* **Resolution**:
  + Cards are revealed, and scoring is evaluated.
  + If no challengers succeed, the active player gets the cards and points.
  + If challengers succeed, the highest becomes the next active player.

**6. Next Turn Logic**

* The next turn starts with:
  + The same active player if they won.
  + The highest successful challenger if applicable.
  + The next player in sequence if the active player lost or timed out.

**7. Winning Condition**

* The game ends when one player has all the cards or when other players' decks are depleted.

**8. Point System**

| **Action** | **Points** |
| --- | --- |
| Active Player wins turn | +1.5 |
| Successful Challenger (top) | +4 |
| Other Successful Challenger | +1 |
| Active Player loses turn | -4 |

* Negative scores are allowed.

**9. Timer Rules**

* All actions have a 15-second timer. Default actions apply on timeout.

**10. UI Requirements**

* **Lobby Screen**: Create/Join room, room code, player count, start game countdown.
* **Game Screen**: Active & Opponent UI with cards, stat selection, accept/give up buttons, timer, scores.
* **Result Screen**: Turn results, card animations, next turn information.

**11. Non-Functional Requirements**

* **Platform**: Android (API 26+)
* **Tech Stack**: Kotlin (frontend), Node.js (backend), MongoDB (database)
* **Performance**: Real-time updates with less than 1-second delay
* **Security**: Server-side validation
* **WebSocket Handling**: Ensure proper connection establishment before sending messages to avoid errors like "Uncaught InvalidStateError: Failed to execute 'send' on 'WebSocket': Still in CONNECTING state."

**12. Enhancements**

* Special cards, emotes, chat
* Card upgrades, leaderboard
* AI bot, rematch feature

**13. Architecture Flows**

The Card Clash Game uses a **client-server architecture** to facilitate real-time multiplayer gameplay. Below is a detailed description of the architecture flows, including both client-side and server-side components.

**13.1 Client-Side Architecture**

* **User Interface Screens**:
  1. **Lobby Screen**:
     + Allows players to create or join a room using a unique room code.
     + Displays the room code, current player count (e.g., "Players: 3/4"), and a countdown when the room is full.
     + Transitions to the game play screen upon receiving the "gameStart" signal from the server.
  2. **Game Screen**:
     + For the **active player**: Shows their top card with buttons to select a stat (e.g., Runs, Wickets).
     + For **opponents**: Displays their top card and options to accept or give up the challenge.
     + Includes a 15-second timer, current scores, and the number of cards each player has.
  3. **Result Screen**:
     + Shows the outcome of each turn: revealed cards, the winner, points awarded, and the next active player.
     + Features animations for card movements (e.g., cards moving to the winner’s deck).
* **Communication**:
  1. **HTTP**: Used for initial setup, such as creating or joining a room.
  2. **WebSocket**: Handles real-time gameplay updates (e.g., stat selections, challenge outcomes) with less than 1-second latency.
* **Local State**:
  1. Stores minimal data, such as the player’s own top card and scores.
  2. Relies on the server for sensitive data (e.g., other players’ cards), which is only revealed when necessary.

**13.2 Server-Side Architecture**

* **Game Room Management**:
  + **Room Creation**: Generates a unique room code and sets the maximum number of players (e.g., 2 to 6).
  + **Player Joining**: Allows players to join the room using the room code. Each join increments the player count. If the player count equals the maximum, the server initiates the game start sequence.
  + **Game Start Sequence**:
    - Sends countdown messages to all clients in the room every second for 3 seconds: { type: "countdown", value: 3 }, { value: 2 }, { value: 1 }.
    - After the countdown, sends { type: "gameStart" } to instruct clients to switch to the game play screen.
    - Initializes the game by shuffling and distributing cards, setting the first active player, and sending the initial game state: { type: "gameState", data: { activePlayer, playerCards, etc. } }.
* **Game Logic**:
  + **Initialization**: Shuffles 50–100 unique cards and distributes them equally to players.
  + **Turn Management**:
    - Designates the active player.
    - Processes stat selections and opponent responses.
    - Resolves challenges by comparing stats and updating scores/cards.
  + **Timeouts**: Enforces 15-second limits, applying penalties (-2 points and card removal for active player, -1 point and card removal for opponents).
  + **Winning Condition**: Ends the game when one player collects all cards or all others are depleted.
* **Real-Time Communication**:
  + Uses WebSockets (e.g., with Socket.IO) to push updates to clients instantly, such as turn results or state changes.
* **Validation and Security**:
  + Executes all game logic server-side to prevent cheating.
  + Validates player actions (e.g., ensuring a selected stat is valid for the top card).

**13.3 Database (MongoDB)**

* **Card Data**: Stores attributes for each unique card (e.g., Player Name, Runs, Wickets).
* **Future Use**: Can store game history or player stats if features like user accounts are added later.
* **In-Memory State**: During gameplay, the server keeps active game states in memory for fast access, given the short duration of games.

**14. Detailed Game Flow**

Here’s how the client and server interact step-by-step:

**14.1 Room Setup**

* A player creates a room via an HTTP request.
* The server generates a unique room code and creates a game room instance.
* Other players join using the code via HTTP requests.
* When the room is full, the server initiates the countdown and game start sequence.

**14.2 Game Initialization**

* The server:
  + Loads card data from MongoDB.
  + Shuffles the deck and distributes cards equally to players.
  + Assigns temporary player IDs (e.g., Player 1, Player 2).
  + Selects the first active player.
* Sends initial state (e.g., each player’s top card) to clients via WebSocket.

**14.3 Active Player’s Turn**

* The active player sees their top card and selects a stat within 15 seconds.
* The selection is sent to the server via WebSocket.
* **Timeout**: If the active player doesn’t act, the server deducts 2 points, removes their top card, and moves to the next player in sequence.

**14.4 Opponent Response**

* The server broadcasts the selected stat (e.g., “Runs”) to opponents.
* Opponents decide to accept or give up within 15 seconds, based on their top card’s stat.
* Decisions are sent to the server.
* **Timeout**: If an opponent doesn’t respond, the server deducts 1 point, removes their top card, and treats it as “give up.”

**14.5 Challenge Resolution**

* The server:
  + Reveals cards from the active player and challengers.
  + Compares the selected stat:
    - **Active Player Wins**: If their stat is ≥ all challengers’ stats (ties favor the active player), they gain +1.5 points and collect all cards involved.
    - **Challenger Wins**: If any challenger’s stat exceeds the active player’s, the highest stat wins; the top challenger gains +4 points, others +1, and the active player loses 4 points. The winner takes all cards.
  + Updates scores and card decks.
* Sends results to clients via WebSocket for display on the Result Screen.

**14.6 Next Turn**

* **Next Active Player**:
  + Same player if they won.
  + Top successful challenger if they won.
  + Next in sequence if the active player timed out or lost all cards.
* The game continues until one player has all cards.

**15. Key Features and Considerations**

* **Timers**: The server enforces 15-second limits and syncs clients with countdowns.
* **Data Efficiency**: Clients receive only their top card and revealed data during resolution, minimizing unnecessary exposure.
* **Scalability**: The server supports multiple concurrent game rooms.
* **Disconnections**: The server can pause briefly for reconnects or remove disconnected players after a timeout.
* **Security**: Server-side validation ensures fair play (e.g., verifying stat selections).

**16. Technology Stack**

* **Client**: Kotlin (Android API 26+).
* **Server**: Node.js with Express.js (HTTP) and Socket.IO (WebSocket).
* **Database**: MongoDB.

This document now includes the architecture flows, detailing the client-server interactions and system design for the Card Clash Game.