UNO GAME API DESIGN & SYSTEM EXPLANATION

1. Introduction

This report presents a comprehensive explanation of the backend API architecture for a real-time, multiplayer UNO game. The goal of the system is to provide a robust, scalable, and fair environment where authenticated users can create or join game lobbies, start matches, play UNO in real time, and interact through chat. The backend design incorporates RESTful API endpoints for game state management and Socket.io WebSocket communication for real-time interactions.

UNO is a turn-based card game in which players attempt to discard all their cards by matching color, number, or symbol with the top card of the discard pile. This simple gameplay creates a complex set of state transitions when implemented digitally requiring synchronized turns, controlled access to actions, hidden card information, and real-time feedback to all players. The API design addresses these requirements through strong validation, strict turn enforcement, private and public communication channels, and a clear separation of gameplay and lobby interactions.

2. System Overview and Architectural Philosophy

The UNO multiplayer system is fundamentally composed of three major subsystems:

1. **Lobby Management** where players gather, prepare, and initiate games.
2. **Game Execution Engine** which enforces rules, manages turns, validates actions, and updates game state.
3. **Real-Time Communication (Socket.io)** which keeps players synchronized instantly, ensuring fairness and responsiveness.

The architecture follows a hybrid model:

- **REST API endpoints** handle *intent-based* actions (“draw a card,” “play a card,” “send a message”).
- **Socket.io** handles all *event broadcasting*, ensuring that all players’ screens update in real time.

This division ensures reliability: REST APIs provide authoritative game logic, while WebSockets provide instant feedback.

The system also enforces strict authorization, ensuring that:

- Only authenticated users can participate.
- Only players in a given game can interact with it.
- Only the current player may execute turn-based actions.
- Players can only see game information relevant to them (i.e they cannot see other players' cards).

Together, these mechanisms create a secure, fair, and interactive game environment.

3. Game Flow and Internal Mechanics

A typical game session proceeds through several phases:

A. Lobby Phase

A user creates a lobby and serves as the host. Other players join the lobby until the host decides the game is ready to begin. During this time, players can chat, toggle readiness, and observe who else has joined. No gameplay occurs here; the lobby is purely organizational.

B. Game Initialization Phase

Once the host starts the game:

- A deck is generated and shuffled.
- Each player receives a predefined number of cards (typically 7).
- Turn order is established.
- The first card of the discard pile is drawn (ensuring it is not a special card that breaks game flow).

The system constructs a complete snapshot of game state and publishes it to all players.

C. Gameplay Phase

Players take turns performing one or more actions depending on game rules:

- Playing a card that matches by color, number, or type.
- Using special cards to alter turn direction, skip opponents, or force draws.
- Drawing a card when no valid moves exist.
- Declaring UNO (if implemented).

Each action triggers validations, card movement between piles, and turn progression, which are broadcast in real time.

D. Endgame Phase

The game ends when a player’s hand is reduced to zero cards. The system finalizes the game state, declares the winner, and optionally archives the match for history.

E. Public vs Private State

Public State (broadcast to all players)

- Top discard card
- Current turn player ID
- Turn direction
- Number of cards each player has
- Discard pile history (optional)
- Game phase/status

Private State (sent only to a specific player)

- Player’s hand
- Drawn card
- Wild color selection
- Special event resolutions (Draw 4 target, etc.)

4. Detailed Explanation of Major API Endpoints

The API is structured into categories based on their role in the game life-cycle.

4.0 List of All API Endpoints (High-Level Overview)

Game Management Endpoints

Name	Method	Route
Create Lobby	POST	/api/lobbies
Join Lobby	POST	/api/lobbies/:lobby_id/join
Start Game	POST	/api/games/:game_id/start

Name	Method	Route
Get Game State	GET	/api/games/:game_id

Game Action Endpoints

Name	Method	Route
Draw Card	POST	/api/games/:game_id/draw
Play Card	POST	/api/games/:game_id/cards/:card_id/play
End Turn	POST	/api/games/:game_id/end-turn
Send Chat Message	POST	/api/games/:game_id/chat

4.1 Game Management Endpoints

These endpoints orchestrate the creation and control of game sessions.

A. Create Lobby Endpoint

This endpoint creates a new lobby where players can gather before the game begins. The server assigns the user who created the lobby as the lobby host. This host has special privileges, such as starting the game. The lobby system ensures that players have a coordinated and organized place to assemble and prepare. It also establishes the foundation for communication channels that will later transition into the game room.

B. Join Lobby Endpoint

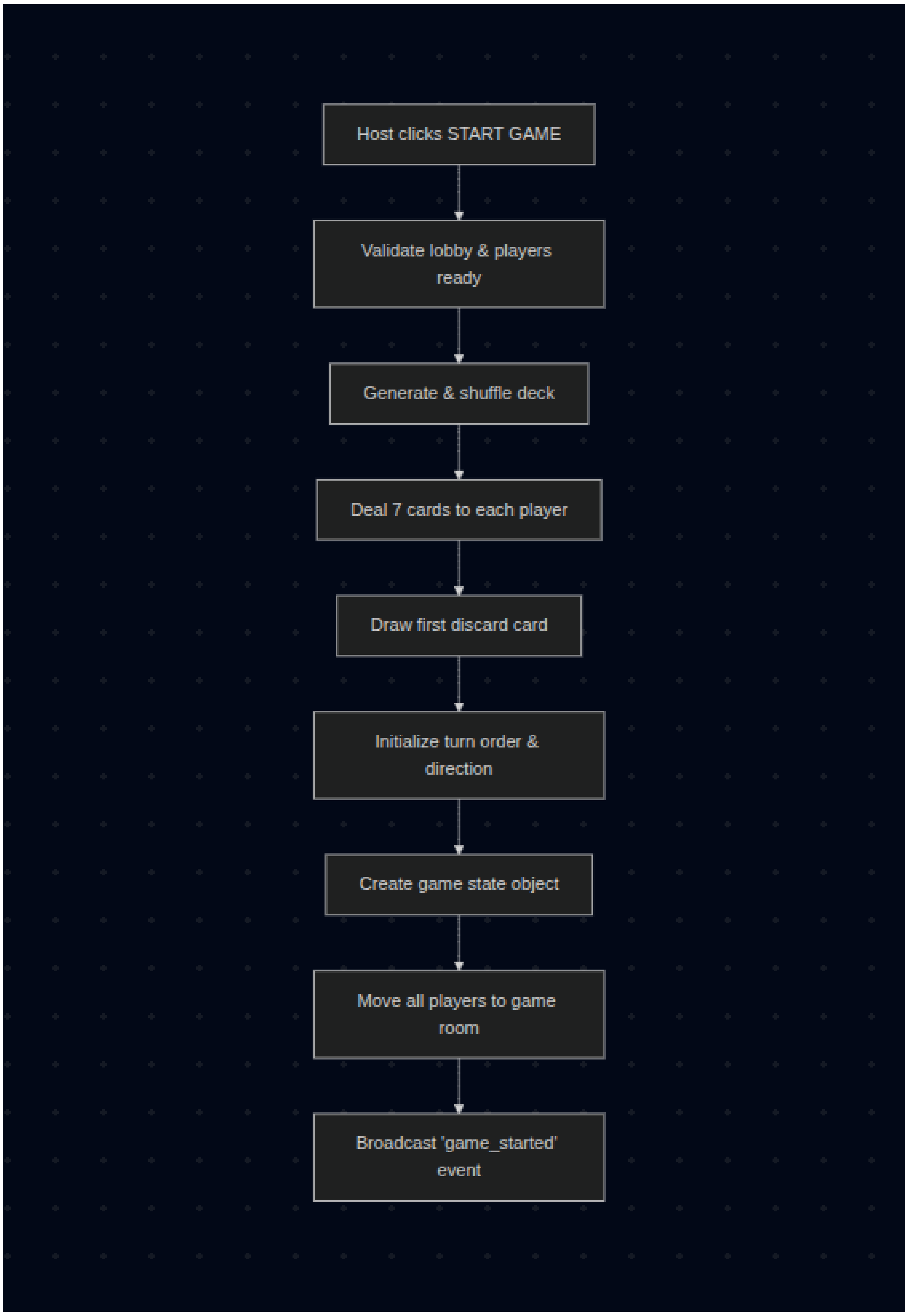
When a player attempts to join an existing lobby, the system validates that:

- The lobby exists.
- The player is authenticated.
- The lobby is not full.

Once validated, the player's information is added to the lobby's participant list, and a Socket.io event notifies all existing players. This ensures transparency and real-time awareness of who is present in the lobby.

C. Start Game Endpoint

Only the lobby host can start the game. This


Host clicks START GAME
Validate lobby & players ready
Generate & shuffle deck
Deal 7 cards to each player
Draw first discard card
Initialize turn order & direction
Create game state object
Move all players to game room
Broadcast 'game_started' event

endpoint triggers several internal mechanisms:

1. Deck creation and shuffling.
2. Dealing cards to players.
3. Initializing turn order and game direction.
4. Transitioning all users into the game room name space.

This endpoint marks the boundary between lobby mode and gameplay mode.

D. Get Game State Endpoint

This endpoint enables any authenticated game participant to request the current state of the match. This is especially important for re connections, page reloads, or mobile instability. The returned state includes:

- The user's hand (private)
- The current top card
- Turn information
- Discard pile details
- Other players' card counts (but not the cards themselves)
- Game direction (clockwise or counterclockwise)

This endpoint guarantees state consistency and a seamless user experience.

Create Lobby — Endpoint Details

1. Endpoint Name
Create Lobby

2. Route
POST /api/lobbies

3. Purpose
Creates a new lobby and assigns the creator as host.

4. Authorization

- Must be logged in

5. Request Body
None (could include options later)

6. Validation Checks

- User authenticated
- User is not already in another lobby or game

7. State Updates

- Insert new lobby row
- Insert lobby_player row with host flag = true

8. Success Response

202 Accepted
{ lobby_id: number }

9. Error Cases

400 – Already in a lobby
401 – Not logged in

10. Socket.io Events

Event: lobby:player:joined
Scope: players in lobby

Join Lobby — Endpoint Details

(similar format)

Start Game — Endpoint Details

(Include: shuffle deck, deal cards, initialize direction, skip invalid first card)

Get Game State — Endpoint Details

(Include public vs private fields)

Draw Card — Endpoint Details

You already wrote a narrative version, but you must attach a **template version**.

Play Card — Endpoint Details

You need to add this entire template:

Validation:

- Does card exist?
- Does player own card?
- Does card legally match top discard?
- Reverse/Skip/Draw2/Wild behavior
- If Wild, body must include chosen_color

State Updates:

- Move card from hand → discard pile
- Apply special effects
- Advance turn

Events:

- game:card:played (public)
- game:hand:update (private)
- game:turn:changed (public)

End Turn — Endpoint Details

Validation:

- Must be player turn
- No playable cards left

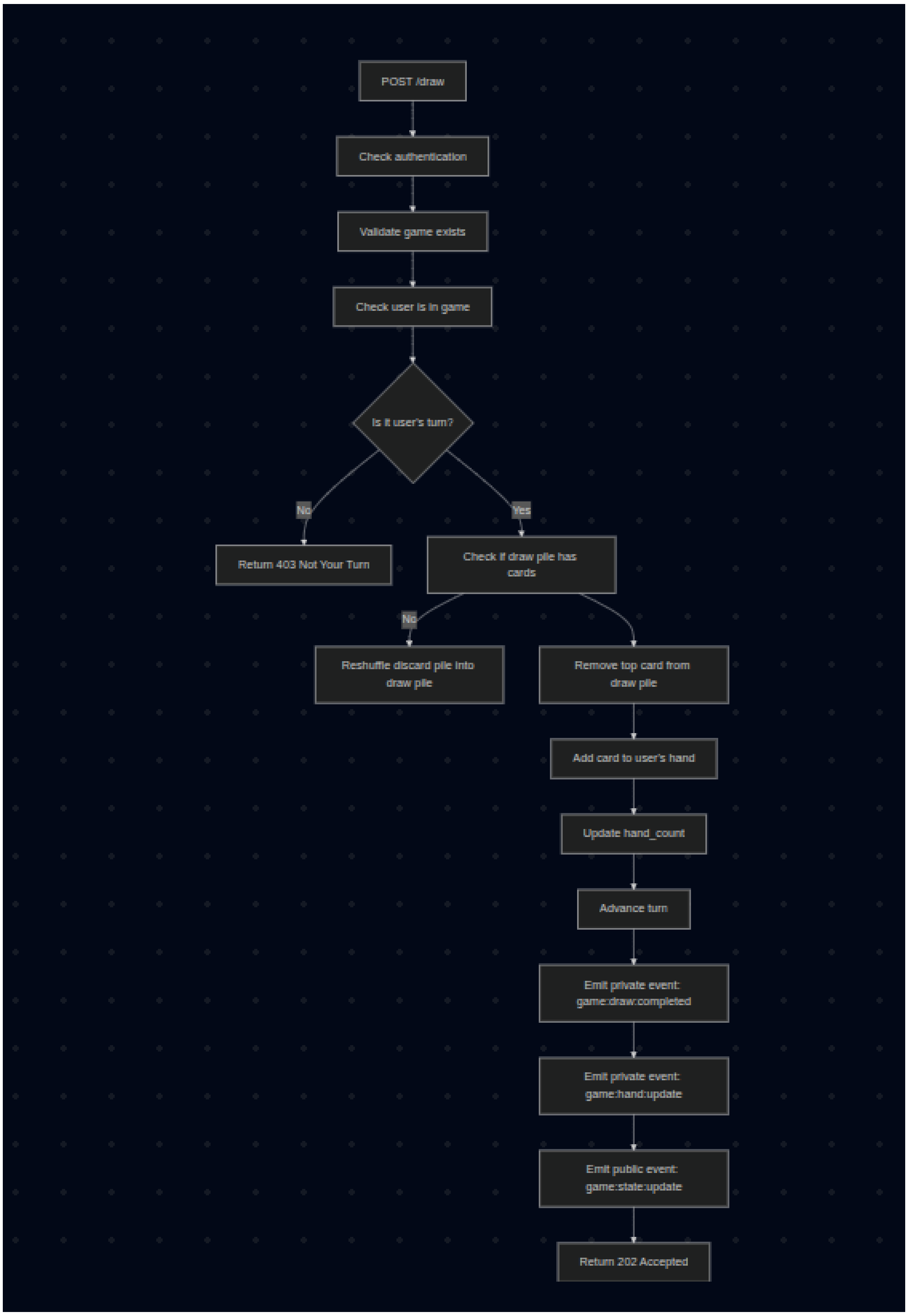
State Updates:

- Advance turn

4.2 Game Action Endpoints

Game actions are the core of gameplay. The system must validate input rigorously to maintain integrity.

4.2.1 The Draw Card Endpoint – Fully Detailed


POST /draw
Check authentication
Validate game exists
Check user is in game
Is it user's turn?
No
Yes
Return 403 Not Your Turn
Check if draw pile has cards
No
Reshuffle discard pile into draw pile
Remove top card from draw pile
Add card to user's hand
Update hand_count
Advance turn
Emit private event: game:draw:completed
Emit private event: game:hand:update
Emit public event: game:state:update
Return 202 Accepted

This is one of the most important endpoints in UNO. Its primary purpose is to allow a player to draw a card from the draw pile, but only when it is their turn. The endpoint enforces that:

- The game exists.

- The user belongs to the game.
- It is the user’s turn.
- The draw pile has cards.

Once validated, the server removes the top card from the draw pile and privately adds it to the player’s hand. The use of a private Socket.io event ensures that only the drawing player can see the card content, preventing any cheating or accidental exposure.

After the draw action is completed, the system automatically advances the turn to the next player. This design matches real UNO behavior, where drawing generally ends your turn unless special rules apply.

Several real-time events are then emitted:

1. A private event to send the drawn card to the player.
2. A private event updating the player’s hand.
3. A public event updating overall game state, such as whose turn it is or how many cards remain in the deck.

This layered event strategy ensures fairness, data privacy, and real-time synchronization without exposing hidden information.

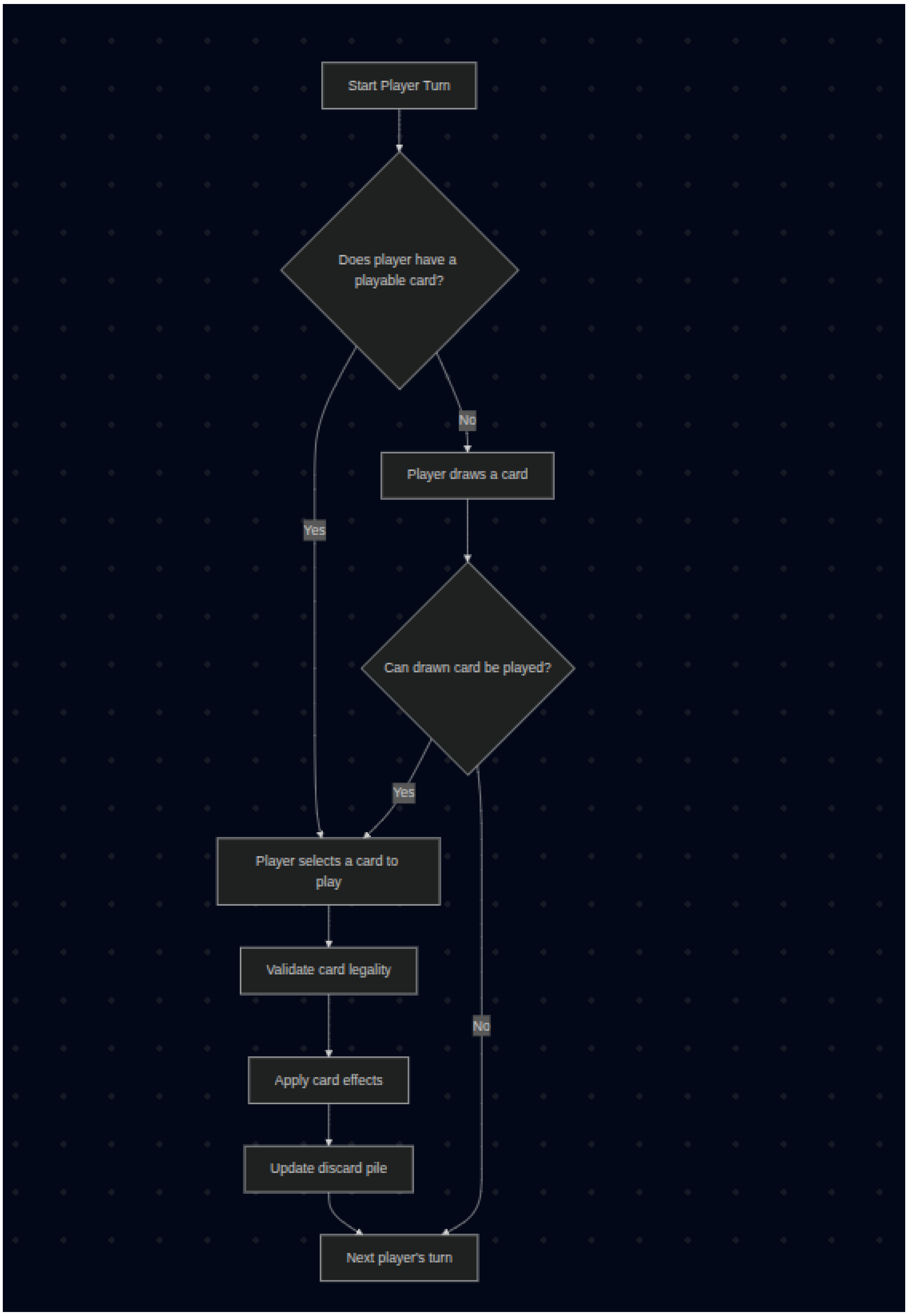
4.2.2 The Send Chat Message Endpoint — Detailed Explanation

The chat system is integrated directly into both the lobby and game room contexts. The chat endpoint is responsible for posting messages to all current participants. The server ensures:

- The user is authenticated.
- The user is currently part of the game.
- The message text is not empty and does not exceed the maximum allowed length.

Once validated, the server stores the message (for persistence and re-connection scenarios) and broadcasts it using Socket.io. This real-time communication strengthens the multiplayer feel and allows players to strategize or socialize, just like they would in an in-person card game.

5.



Real-Time System: Socket.io Domains & Responsibilities

Real-time communication is essential in a turn-based game where state depends on every action. Socket.io allows immediate broadcasting of events, preventing delays that could cause inconsistencies between players' game states.

The Socket.io system is divided into three main domains:

A. Game Domain

This is responsible for actual gameplay events:

- Card played
- Card drawn
- Turn changes
- Discard pile updates
- Special card effect activation

This domain ensures that every player sees the exact same game state at all times.

B. Lobby Domain

This domain handles the pre-game coordination. It manages:

- Players joining or leaving
- Readiness toggling
- Lobby state updates
- Game start countdown or trigger

This creates a smooth transition into gameplay and improves user experience by giving immediate feedback when players join or prepare.

C. Chat Domain

This domain manages real-time player-to-player communication. It ensures all chat messages are instantly visible to all players and are properly categorized into lobby chat or game chat.

The separation of these Socket.io name-spaces keeps the system clean, modular, and scalable. Each domain operates independently without interfering with the others, which greatly simplifies debugging and backend logic while improving performance.

5.1 List of Socket.io Events (Required for Milestone)

Public Events

game:state:update

Scope: all players
Trigger: any game state change

Data:
{ game_id, current_player, top_discard, turn_direction, player_card_counts }

game:card:played

Scope: all players
Trigger: player plays a card
Data:
{ player_id, card_id, updated_discard, next_player }

Private Events

game:hand:update

Scope: single player
Trigger: hand changes
Data: { hand: [...] }

game:card:drawn

Scope: single player
Trigger: draw card
Data: { card }

6. Conclusion

This system is designed to faithfully recreate the dynamic, interactive experience of a live UNO game in an online environment. The combination of RESTful endpoints and real-time WebSocket communication provides both authoritative control and instant responsiveness. Through clearly defined responsibilities, strict validation, private card communication, and intuitive turn management, the architecture ensures fairness, accuracy, and a smooth multiplayer experience.

The design presented here provides a robust and extensible foundation upon which additional features such as UNO declarations, penalties, game statistics, user profiles, matchmaking, or ranked modes can be built seamlessly.