Milestone 3: Game API Design

Brief game description --

- Our project is an online multiplayer UNO game where players can create or join game rooms, draw and play cards, and compete to get rid of all their cards first. Each turn, players can match a card by color or number, or play special cards such as Skip, Reverse, Draw Two, Wild, and Wild Draw Four. The game follows standard UNO rules without variations. The server handles all game logic.

List of all Socket.io events with descriptions ---

Event: conn:connected
Scope: Individual player
Trigger: When a player successfully connects to the server
Data:

```
{
 "user_id": number,
 "server_ts": number
}
```

Event: lobby:player:joined
Scope: All users in the lobby room
Trigger: When a player joins the lobby
Data:

```
{
 "lobby_id": number,
 "player": { "id": number, "display_name": string },
 "players": [{ "id": number, "display_name": string }],
 "server_ts": number
}
```

Event: lobby:player:left
Scope: All users in the lobby room
Trigger: When a player leaves the lobby
Data:

```
{
 "lobby_id": number,
 "player": { "id": number, "display_name": string },
 "players": [{ "id": number, "display_name": string }],
 "server_ts": number
}
```

Event: lobby:player:ready
Scope: All users in the lobby room
Trigger: When a player clicks ready
Data:

```
{
  "lobby_id": number,
  "player": { "id": number, "display_name": string },
  "ready": boolean,
  "server_ts": number
}
```

Event: lobby:state:update
Scope: All users in the lobby room
Trigger: When a game state changes
Data:

```
{
  "lobby_id": number,
  "name": string,
  "capacity": number,
  "players": [{ "id": number, "display_name": string, "ready": boolean }],
  "is_ready": boolean,
  "server_ts": number
}
```

Event: lobby:game:started
Scope: All users in the lobby room
Trigger: When host starts a game
Data:

```
{
  "lobby_id": number,
  "game_id": number,
  "server_ts": number
}
```

Event: lobby:error
Scope: Specific player only
Trigger: When a lobby related error occurs
Data:

```
{
  "code": "LOBBY_FULL" | "LOBBY_NOT_FOUND" | "FORBIDDEN",
  "message": string,
  "server_ts": number
}
```

Event: game:card:played
Scope: All players in the game room
Trigger: When a player plays a valid card
Data:

```
{
  "game_id": number,
  "player_id": number,
  "card": { "id": number, "color": string, "value": string },
  "chosen_color": string | null,
  "effects": { "skip"?: boolean, "reverse"?: boolean, "draw"?: number },
  "next_player_id": number,
  "server_ts": number
}
```

Event: game:draw:completed
Scope: Specific player only
Trigger: When a player draws one card from the deck
Data:

```
{
  "game_id": number,
  "player_id": number,
  "drawn": [{ "id": number, "color": string, "value": string }],
  "server_ts": number
}
```

Event: game:hand:update
Scope: Specific player only
Trigger: When a player hand changes after drawing or playing a card
Data:

```
{
  "game_id": number,
  "player_id": number,
  "hand": [{ "id": number, "color": string, "value": string }],
  "hand_count": number,
  "server_ts": number
```

}

Event: game:state:update
Scope: All players in the game room
Trigger: When game state changes (turn change, pile count change, top card)
Data:
```
{
  "game_id": number,
  "status": "setup" | "playing" | "ended",
  "turn_number": number,
  "current_player_id": number,
  "direction": "cw" | "ccw",
  "top_discard": { "color": string, "value": string },
  "draw_pile_count": number,
  "hand_counts": { "player_id": number },
  "server_ts": number
}
```

Event: game:turn:changed
Scope: All players in the game room
Trigger: When server advances to next player turn
Data:
```
{
  "game_id": number,
  "next_player_id": number,
  "turn_number": number,
  "server_ts": number
}
```

Event: game:reverse
Scope: All players in the game room
Trigger: When reverse card is played
Data:
```
{
  "game_id": number,
  "direction": "cw" | "ccw",
  "server_ts": number
}
```

Event: game:skip

Scope: All players in the game room
Trigger: When skip card is played skipping next player
Data:

```
{
  "game_id": number,
  "skipped_player_id": number,
  "server_ts": number
}
```

Event: game:color:chosen
Scope: All players in the game room
Trigger: When player plays a Wild or Wild Draw Four card and selects color
Data:

```
{
  "game_id": number,
  "player_id": number,
  "color": "red" | "yellow" | "green" | "blue",
  "server_ts": number
}
```

Event: game:ended
Scope: All players in the game room
Trigger: When a player wins or the game is finished
Data:

```
{
  "game_id": number,
  "winner_id": number,
  "scoreboard": [{ "player_id": number, "rank": number, "hand_count": number }],
  "ended_at": string,
  "server_ts": number
}
```

Event: game:error
Scope: Specific player only (private)
Trigger: When a player performs an invalid or unauthorized action
Data:

```
{
  "code": "INVALID_MOVE" | "FORBIDDEN"
  "message": string,
  "server_ts": number
```

}

Event: chat:message:new
Scope: All players in the game room
Trigger: When a player sends a chat in chat room
Data:

```
{
  "game_id": number,
  "message": {
    "id": number,
    "user_id": number,
    "display_name": string,
    "text": string,
    "time_sent": string
  },
  "server_ts": number
}
```

Event: chat:thread:update
Scope: All players in the game room
Trigger: When a chat is edited
Data:

```
{
  "game_id": number,
  "messages": [
    {
      "id": number,
      "user_id": number,
      "display_name": string,
      "text": string,
      "time_sent": string,
      "edited": boolean,
      "deleted": boolean
    }
  ],
  "server_ts": number
}
```

List of all API endpoints with details ---

Endpoint Name: Create Lobby

1. **Endpoint Name:** Create Lobby
2. **HTTP Method & Route:** POST /api/lobbies
3. **Purpose:** Create a new lobby so friends can join.
4. **Authorization:**
   - User must be logged in
5. **Request Body:**
   { "name": "string", "capacity": 2 }
6. **Validation Checks:**
   - capacity bounds (2–10)
7. **State Updates:**
   - Insert into lobbies (host_id = caller, status=open, capacity, name)
8. **Success Response:** 201 Created { "lobby_id": number }
9. **Error Cases:**
   - 400 invalid capacity/name
   - 401 not logged in
10. **Socket.io Events:**
    - lobby:player:joined → lobby room (host appears)
    - lobby:state:update → lobby room

Endpoint Name: Join Lobby

1. **Endpoint Name:** Join Lobby
2. **HTTP Method & Route:** POST /api/lobbies/:lobby_id/join
3. **Purpose:** Add players to an existing lobby.
4. **Authorization:**
   - Users must be logged in.
5. **Request Body:** —
6. **Validation Checks:**
   - Lobby exists and is open
   - Lobby not full
   - Player is not already in lobby
7. **State Updates:**
   - Insert player user in lobby participants
8. **Success Response:** 202 Accepted
9. **Error Cases:**
   - 404 lobby not found
   - 403 lobby full /closed
   - 409 already joined

10. **Socket.io Events:**
- lobby:player:joined → lobby room
- lobby:state:update → lobby room

Endpoint Name: Start Game

1. **Endpoint Name:** Start Game
2. **HTTP Method & Route:** POST /api/games
3. **Purpose:** Create a game from a lobby, shuffle/deal hands, pick starter card, set first turn.
4. **Authorization:**
    - User must be logged in and lobby host
    - Lobby must have ≥2 players
5. **Request Body:**
   { "lobby_id": number, "seed": number }
6. **Validation Checks:**
    - Lobby exists
    - Player is lobby host
    - Player count ≥ 2
    - Lobby not already started/closed
7. **State Updates:**
    - Insert games (status=playing, direction=cw, turn_number=1, set current_player_id)
    - Copy lobby members → gameParticipants (set player_order)
    - Build/shuffle deck (seed if provided); deal 7 to each; flip valid starter card to discard
8. **Success Response:** 201 Created { "game_id": number }
9. **Error Cases:**
    - 404 lobby not found
    - 403 not host / not enough players / lobby not ready
    - 409 already started
10. **Socket.io Events:**
    - lobby:game:started → lobby room
    - game:hand:update → private to each player (initial hand)
    - game:state:update → game room

Endpoint Name: Draw Card

1. **Endpoint Name:** Draw Card
2. **HTTP Method & Route:** POST /api/games/:game_id/draw
3. **Purpose:** Current player draws 1 card from the draw pile
4. **Authorization:**
    - User must be authenticated

- User must be a participant in the game
- User must be the current player

5. **Request Body:** —
6. **Validation Checks:**
    - Game exists and user is playing
    - Caller is current_player_id
    - Draw pile available
    - No pending required color choice
7. **State Updates:**
    - Move top card to caller's hand
    - Update player's hand_count
    - Advance turn
8. **Success Response:** 202 Accepted
9. **Error Cases:**
    - 403 not your turn
    - 409 conflict/state changed
10. **Socket.io Events:**
    - game:draw:completed → private to drawer (cards drawn)
    - game:hand:update → private to drawer (new hand)
    - game:state:update → game room (counts/pile)

--

Endpoint Name: Play Card

1. **Endpoint Name:** Play Card
2. **HTTP Method & Route:** POST /api/games/:game_id/play
3. **Purpose:** Play a card from hand to discard; update color if wild; apply effects (Skip, Reverse, +2, Wild/+4).
4. **Authorization:**
    a. User must be Logged in
    b. User must be a game participant;
    c. **User must be playing and be the current player**
5. **Request Body:**
   { "card_id": number, "chosen_color": "red"|"yellow"|"green"|"blue" }
6. **Validation Checks:**
    - Game exists and player status is playing
    - Player is current_player_id
    - Card is in caller's hand and belongs to this game
    - Legal play: match color/number/symbol OR is Wild/Wild+4
    - Wild+4 only if caller has **no card** matching current color
    - If wild, chosen_color is provided
7. **State Updates:**

- Move card to discard
- Set top color (if wild)
- Apply effects: reverse/skip/draw penalties for next player
- Update hand_count

8. **Success Response:** 202 Accepted
9. **Error Cases:**
   - 400 illegal move / missing color
   - 403 not your turn
   - 404 card not found
   - 409 conflict/race
10. **Socket.io Events:**
    - game:card:played → game room
    - game:color:chosen → game room (if wild)
    - game:reverse / game:skip → game room (as applicable)
    - game:hand:update → private to player who played
    - game:state:update → game room
    - game:turn:changed → game room
    - game:ended → game room (if win)

Endpoint Name: End Turn

1. **Endpoint Name:** End Turn
2. **HTTP Method & Route:** POST /api/games/:game_id/end-turn
3. **Purpose:** Pass the turn after drawing if no card is played.
4. **Authorization:**
   a. User must be logged in
   b. User must be a participant and current player
5. **Request Body:** —
6. **Validation Checks:**
   - Game exists user is playing
   - User is current_player_id
   - No required action (wild color card forces play)
7. **State Updates:**
   - Increment turn_number
   - set current_player_id to next (respect direction/skip/penalties)
8. **Success Response:** 202 Accepted
9. **Error Cases:**
   - 400 pending action prevents end-turn
   - 403 not your turn
   - 409 conflict/race
10. **Socket.io Events:**

- game:turn:changed → game room
- game:state:update → game room

Endpoint Name: End Game

1. **Endpoint Name:** End Game
2. **HTTP Method & Route:** server side (action in conjoinement with **check game status**)
3. **Purpose:** Automatically end the game immediately after the server detects a winner
4. **Authorization:**
    - server-side rule after game state check
5. **Request Body:**
   { "reason": "string" }
6. **Validation Checks:**
    - Game exists
7. **State Updates:**
    - Set status to ended
    - Compute scoreboard; set winner_id if applicable
8. **Success Response:** 202 Accepted
9. **Error Cases:**
    - 403 forbidden
    - 409 already ended
10. **Socket.io Events:**
    - game:ended → game room

Endpoint Name: Send Chat Message

1. **Endpoint Name:** Send Chat Message
2. **HTTP Method & Route:** POST /api/games/:game_id/chat
3. **Purpose:** Send a chat message to all players in the game room.
4. **Authorization:**
    - User must be authenticated
    - User must be a player in this game
5. **Request Body:**
   { "text": "string" }
6. **Validation Checks:**
    - Game exists
    - Caller is participant
    - text present and within max length
7. **State Updates:**
    - Insert into chat (game_id, user_id, text)
    - Update game chat for all
8. **Success Response:** 202 Accepted

9. **Error Cases:**
    - 400 missing text
    - 403 not in game
    - 404 game not found
10. **Socket.io Events:**
    - chat:message:new → game room