Milestone 3: API Design

Game Description

UNO is a turn-based card game where players play cards from their hands by matching the last played card by color or value, or by playing wild cards that set a color. The first player to run out of cards in their hand wins. Our application will manage this game for 2-4 players in real-time.

1. Authentication & User Endpoints

Endpoint: Create User

- **Method & Route:** POST /api/users/register
- **Purpose:** Creates a new user account.
- **Authorization:** Public (no authentication required).
- **Request Body:**

```
{
  "email": "user@example.com",
  "username": "new_user",
  "password": "a_strong_password"
}
```

- **Validation Checks:**
  1. Validate email is a valid format.
  2. Validate username is not null (min 3 chars).
  3. Validate password is not null (min 8 chars).
  4. Query database to check if email already exists.
  5. Query database to check if username already exists.
- **State Updates:**
  1. Generate a salt and hash the password.
  2. INSERT a new record into the users table.
- **Success Response:** 201 Created

```
{
  "id": 1,
  "email": "user@example.com",
  "username": "new_user",
  "created_at": "...",
  "token": "your_jwt_token_here"
}
```

- **Error Cases:**
    - 400 Bad Request: Invalid email format or password too short.
    - 409 Conflict: Email or username already exists.
- **Socket.io Events:** None.

Endpoint: Login User

- **Method & Route:** POST /api/users/login
- **Purpose:** Authenticates a user and returns a session token.
- **Authorization:** Public.
- **Request Body:**
  ```
  {
    "username": "new_user",
    "password": "a_strong_password"
  }
  ```
- **Validation Checks:**
    1. Query database to find user by username.
    2. If user exists, compare the provided password with the stored hash.
- **State Updates:** None. (A session token is generated, but no DB state change).
- **Success Response:** 200 OK
  ```
  {
    "id": 1,
    "email": "user@example.com",
    "username": "new_user",
    "token": "your_jwt_token_here"
  }
  ```
- **Error Cases:**
    - 401 Unauthorized: Password does not match.
    - 404 Not Found: User with that username does not exist.
- **Socket.io Events:** None.

2. Game Management Endpoints

Endpoint: Create Game

- **Method & Route:** POST /api/games
- **Purpose:** Creates a new game lobby. The creator becomes the host.
- **Authorization:** User must be authenticated (token required).
- **Request Body:** None. (Host user_id is taken from the auth token).

- **Validation Checks:**
    1. Validate user is logged in.
- **State Updates:**
    1. INSERT a new record into games table with:
        - status: false (lobby state)
        - player_turn: null
        - is_clockwise: true
        - last_card_played: null
    2. INSERT a new record into game_users with:
        - game_id: The new game's ID.
        - user_id: The host's user ID (from token).
        - is_host: true
        - turn_order: 1
- **Success Response:** 201 Created (Returns the new game object)
  ```
  {
   "id": 123,
   "status": false,
   "is_clockwise": true,
   "player_turn": null,
   "last_card_played": null,
   "created_at": "..."
  }
  ```
- **Error Cases:**
    - 401 Unauthorized: User is not logged in.
- **Socket.io Events:** None. (The client that just created the game will be responsible for joining its room).

Endpoint: Join Game

- **Method & Route:** POST /api/games/{game_id}/join
- **Purpose:** Adds the authenticated user to an existing game lobby.
- **Authorization:** User must be authenticated.
- **Request Body:** None.
- **Validation Checks:**
    1. Check if game_id exists in games table.
    2. Check games.status is false (game is still a lobby).
    3. Check if user is already in this game (query game_users).
    4. Count players in game_users for this game_id. If >= 4, reject.
- **State Updates:**
    1. Get MAX(turn_order) for this game_id from game_users.
    2. INSERT a new record into game_users with:

        - game_id: game_id from URL.
        - user_id: User's ID (from token).
        - is_host: false
        - turn_order: MAX(turn_order) + 1
- **Success Response:** 200 OK
  ```
  {
    "game_id": 123,
    "user_id": 2,
    "turn_order": 2,
    "is_host": false
  }
  ```
- **Error Cases:**
    - 401 Unauthorized: User not logged in.
    - 403 Forbidden: Game is full (>= 4 players) or has already started (status is true).
    - 404 Not Found: game_id does not exist.
    - 409 Conflict: User is already in this game.
- **Socket.io Events:**
    - game:player:joined -> All players in the game room.
        - **Data:** { "user_id": 2, "username": "player_2", "turn_order": 2 }

Endpoint: Start Game

- **Method & Route:** POST /api/games/{game_id}/start
- **Purpose:** Initializes the game. Deals cards and sets the first turn.
- **Authorization:** User must be authenticated AND is_host in game_users.
- **Request Body:** None.
- **Validation Checks:**
    1. Check if game_id exists.
    2. Check if user is the host for this game.
    3. Check games.status is false (game is not already started).
    4. Count players in game_users. Must be between 2 and 4.
- **State Updates:**
    1. Shuffle deck (logic in server, not DB).
    2. For each player in game_users, INSERT 7 random cards into game_cards.
    3. Draw one card for the discard pile. INSERT this card into game_cards with user_id = null.
    4. UPDATE games SET last_card_played to the ID of this discard card.
    5. UPDATE games SET status = true.
    6. Get user_id from game_users where turn_order = 1.
    7. UPDATE games SET player_turn to that user_id.
- **Success Response:** 202 Accepted (Game logic is processing).

- **Error Cases:**
    - 401 Unauthorized: User not logged in.
    - 403 Forbidden: User is not the host.
    - 400 Bad Request: Not enough players (must be 2-4).
    - 404 Not Found: game_id does not exist.
- **Socket.io Events:**
    - game:state:update -> All players in room. (Sends full public state).
    - game:hand:update -> Each player privately. (Sends their 7 cards).

Endpoint: Get Game State (Public)

- **Method & Route:** GET /api/games/{game_id}
- **Purpose:** Returns the full public state of a game in progress.
- **Authorization:** User must be authenticated AND a participant in the game.
- **Request Body:** None.
- **Validation Checks:**
    1. Check if game_id exists.
    2. Check if user is in game_users for this game_id.
- **State Updates:** None.
- **Success Response:** 200 OK
```
{
 "id": 123,
 "status": true,
 "player_turn": 1, // user_id of current player
 "is_clockwise": true,
 "last_card_played": { "id": 50, "color": "blue", "symbol": "five" },
 "players": [
  { "user_id": 1, "username": "host_user", "card_count": 7 },
  { "user_id": 2, "username": "player_2", "card_count": 7 }
 ]
}
```
- **Error Cases:**
    - 401 Unauthorized: User not logged in.
    - 403 Forbidden: User is not a participant in this game.
    - 404 Not Found: game_id does not exist.
- **Socket.io Events:** None.

Endpoint: Get My Hand (Private)

- **Method & Route:** GET /api/games/{game_id}/hand
- **Purpose:** Returns the private hand for the *authenticated* user.

- **Authorization:** User must be authenticated AND a participant in the game.
- **Request Body:** None.
- **Validation Checks:**
    1. Check if game_id exists.
    2. Check if user is in game_users for this game_id.
- **State Updates:** None.
- **Success Response:** 200 OK
```
{
  "hand": [
    { "id": 10, "color": "red", "symbol": "one" },
    { "id": 25, "color": "green", "symbol": "skip" },
    { "id": 108, "color": "black", "symbol": "wild" }
  ]
}
```
- **Error Cases:**
    - 401 Unauthorized: User not logged in.
    - 403 Forbidden: User is not a participant in this game.
    - 404 Not Found: game_id does not exist.
- **Socket.io Events:** None.

3. Game Action Endpoints

Endpoint: Play Card

- **Method & Route:** POST /api/games/{game_id}/play
- **Purpose:** Allows the current player to play a card from their hand.
- **Authorization:** Must be authenticated; must be a participant; must be their turn.
- **Request Body:**
```
{
  "card_id": 25,
  "chosen_color": "blue" // Optional: only for Wild cards
}
```
- **Validation Checks:**
    1. Check if game_id exists and status is true.
    2. Check games.player_turn matches the authenticated user's ID.
    3. Check if card_id exists in game_cards for this user_id and game_id.
    4. Get games.last_card_played. Check if the card_id from request is a valid move (matches color, symbol, or is wild).
    5. If card is wild, check that chosen_color is provided.

- **State Updates:**
    1. DELETE card from game_cards (player's hand).
    2. UPDATE games SET last_card_played = card_id. (If wild, store a "virtual" card ID that also contains the chosen_color).
    3. **Handle special card logic (skip, reverse, draw).**
    4. Determine next player's user_id based on turn_order and is_clockwise.
    5. UPDATE games SET player_turn to the next player's user_id.
- **Success Response:** 202 Accepted.
- **Error Cases:**
    - 401 Unauthorized: Not logged in.
    - 403 Forbidden: Not your turn.
    - 404 Not Found: Game or card not found.
    - 400 Bad Request: Invalid move (card doesn't match), or chosen_color missing for wild.
- **Socket.io Events:**
    - game:state:update -> All players in room.
    - game:hand:update -> The player who just played (their hand is now smaller).
    - game:ended -> If this was the player's last card.

Endpoint: Draw Card

- **Method & Route:** POST /api/games/{game_id}/draw
- **Purpose:** Allows the current player to draw a card from the deck.
- **Authorization:** Must be authenticated; must be a participant; must be their turn.
- **Request Body:** None.
- **Validation Checks:**
    1. Check if game_id exists and status is true.
    2. Check games.player_turn matches the authenticated user's ID.
    3. (Optional: Check if player has any valid moves. Most Uno versions allow drawing anytime).
- **State Updates:**
    1. Get one random card not in game_cards for this game_id.
    2. INSERT this card into game_cards for this user_id and game_id.
- **Success Response:** 202 Accepted.
- **Error Cases:**
    - 401 Unauthorized: Not logged in.
    - 403 Forbidden: Not your turn.
    - 404 Not Found: Game not found.
    - 409 Conflict: Deck is empty (edge case!).
- **Socket.io Events:**
    - game:state:update -> All players (player's card count changed).
    - game:hand:update -> The player who just drew (sends their new hand).

Endpoint: End Turn (Skip)

- **Method & Route:** POST /api/games/{game_id}/end-turn
- **Purpose:** Allows a player to end their turn, (e.g., after drawing a card).
- **Authorization:** Must be authenticated; must be a participant; must be their turn.
- **Request Body:** None.
- **Validation Checks:**
  1. Check if game_id exists and status is true.
  2. Check games.player_turn matches the authenticated user's ID.
  3. (Optional: Check if the user has already drawn a card this turn).
- **State Updates:**
  1. Determine next player's user_id based on turn_order and is_clockwise.
  2. UPDATE games SET player_turn to the next player's user_id.
- **Success Response:** 202 Accepted.
- **Error Cases:**
  - 401 Unauthorized: Not logged in.
  - 403 Forbidden: Not your turn.
  - 404 Not Found: Game not found.
- **Socket.io Events:**
  - game:state:update -> All players in room.

4. Socket.io Events

This list defines the real-time events our server will emit.

Event: game:state:update

- **Scope:** Public (All players in the game room).
- **Trigger:** After any action that changes the public game state (card played, card drawn, turn ended, game started).
- **Data:** The same JSON object as GET /api/games/{game_id}.

Event: game:hand:update

- **Scope:** Private (Specific player only).
- **Trigger:** After a player draws a card, plays a card, or when the game starts (initial deal).
- **Data:** The player's complete, new hand.

```
{
  "hand": [
    { "id": 10, "color": "red", "symbol": "one" },
    ...
  ]
}
```

Event: game:player:joined

- **Scope:** Public (All players in the game room).
- **Trigger:** After a user successfully joins a lobby (POST .../join).
- **Data:** The new player's public information.
  ```
  {
    "user_id": 2,
    "username": "player_2",
    "turn_order": 2
  }
  ```

Event: game:ended

- **Scope:** Public (All players in the game room).
- **Trigger:** When a player plays their last card.
- **Data:** The user_id and username of the winner.
  ```
  {
    "winner_id": 1,
    "winner_username": "host_user"
  }
  ```

Event: error:message

- **Scope:** Private (Specific player only).
- **Trigger:** When a user's API call fails validation in a way that needs to be communicated (e.g., "Invalid move").
- **Data:** A simple error message.
  ```
  {
    "message": "That is not a valid move. The last card was a Blue 5."
  }
  ```