Game description

- An online implementation of poker. The server acts as the dealer.

Socket.io Events:

Trigger: When any player successfully plays a card

Data:

```
{
  game_id: number,
  player_id: number,
  card_id: number,
  updated_board: [...],
  next_player_id: number
}
```

Public Scope (Broadcast to all players in the game)

1. `game:state:update` - Change the round
    - Triggered when the betting part of a round is done

      Data:

      ```
      {
        game_id: number,
        game_state: text,
      }
      ```
2. `game:player:joined` - New player joins game
    - Triggered when new player joins (only during the lobby state)

      Data:

      ```
      {
        player_name: text,
      }
      ```
3. `game:player:left` - Player leaves game
    - Triggered when player leaves, either voluntarily or disconnect

      Data:

```
{
  player_name: text,
}
```

4. **`game:turn:changed`** - Turn moves to next player
   - First triggered after cards are dealt, and after a player performs an action up until the last player in the table acts.

   Data: (none)

5. **`game:community:cards:dealt`** - Community cards revealed (flop, turn, river)
   - Triggered after the betting round that following the distribution of private hand.

   Data:

   ```
   {
     card: Card[ ],
   }
   ```

6. **`Game:community:cards:add`** - Add a card to community card pile
   - Triggered at beginning of each round minus the last

   Data:

   ```
   {
     card: Card,
   }
   ```

7. **`game:bet:placed`** - Player placed a bet
   - Triggered only if a player chooses to bet money

   Data:

   ```
   {
     money_bet: number,
     pot_amount: number,
   }
   ```

8. **`game:bet:call`** - Player calls
   - Triggered when player calls

   Data:

   ```
   {
     player_name: text,
   }
   ```

9. **`game:bet:fold`** - Player folds
   - Triggered when player folds

   Data:

   ```
   {
     player_name: text,
   }
   ```

10. `game:ended` - Game finished
    - Triggered after last round when players show hands

      Data:

      ```
      {
        hands: Card[ ][ ], (array of hands)
        playerNames: text[ ],
      }
      ```

Private Scope (Sent to specific player)

1. `game:hand:update` - Player's private cards
    - Triggered once in the very start of game

      Data:

      ```
      {
        cards: Card[ ],
      }
      ```

2. `game:action:required` - Notify player it's their turn
    - Triggered after community card is delt, and after a player performs an action.

      Data: None

3. `game:balance:update` - Player's money balance changed
    - Triggered after betting/raising, or if they win a game

      Data:

      ```
      {
        money: number,
        lose_or_gain: bool,
      }
      ```

4. `game:result:personal` - Personal round results
    - Triggered at the end of a game

      Data:

      ```
      {
        lost: bool,
        money_earned_lost: number,
        Total_money: number
      }
      ```

API Endpoints

Field	Description
1. Endpoint Name	**Create Game**
2. HTTP Method and Route	`POST /api/games`
3. Purpose	Create a new poker table
4. Authorization	• Must be logged in

5. Request Body	(Persistant money)
	{
	"max_players": 6,
	"buy_in": 1000
	}
	
	(Players have the same amount of money each game)
	{
	"max_players": 6,
	(amount of money they are oblibated to bet)
	"small_blind": 5,
	"big_blind": 10
	}

6. Validation Checks	• User authenticated • valid max_players, • valid buy_in amount (might not do in final implementation)
7. State Updates	• `Insert new game into game table with current user as the owner, set the max players from user specified amount` • `Add new row in table table, set game id to the row created above. Initialize pot to 0 and set game state to lobby`
8. Success Response	`201 Created` with game ID
9. Error cases	What can go wrong? • 400 Bad Request - Invalid max players • 401 Unauthorized - Not logged in • 403 Forbidden - N/A • 404 Not Found - N/A

	• 408 Timed Out - N/A • 409 Conflict - N/A
10. Socket.io Events	None (player hasn't joined yet)

Field	Description
1. Endpoint Name	**Join Game**
2. HTTP Method and Route	`POST /api/games/:game_id/join`
3. Purpose	Add player to existing game table
4. Authorization	• Must be authenticated • Must not already be in game
5. Request Body	None
6. Validation Checks	• Game exists • Gamehas open seats • User has sufficient balance • Game not in progress
7. State Updates	• `Insert into table_player, deduct buy_in from user balance`
8. Success Response	`202 accepted`
9. Error cases	What can go wrong? • 400 Bad Request - Max players already • 401 Unauthorized - Not logged in • 403 Forbidden - Game in progress • 404 Not Found - N/A • 408 Timed Out - N/A • 409 Conflict - N/A
10. Socket.io Events	`game:player:joined` → All players in game

Field	Description
1. Endpoint Name	**Start Game**
2. HTTP Method and Route	`POST /api/games/:game_id/start`
3. Purpose	The person who created the game starts the poker game. Server will deal two cards to every player
4. Authorization	• Must be game owner, at least 2 players
5. Request Body	None
6. Validation Checks	• User is owner • Minimum players present • Game in "lobby" state
7. State Updates	• Update `table.state` to "playing" • deal initial cards • set blinds
8. Success Response	202 Accepted
9. Error cases	What can go wrong? • 400 Bad Request - >2 players • 401 Unauthorized - Not logged in • 403 Forbidden - Not game owner • 404 Not Found - N/A • 408 Timed Out - N/A • 409 Conflict - N/A
10. Socket.io Events	• `game:state:update` →Let players know the state has updated • `game:hand:update` → Each player gets their private cards • `game:turn:changed` → All players

Field	Description

1. Endpoint Name	**Get Game State**
2. HTTP Method and Route	`GET /api/games/:game_id`
3. Purpose	Retrieve current game state for a player. (lobby, flop, preflop,turn, river)
4. Authorization	• Must be in the game
5. Request Body	None
6. Validation Checks	• Game exists • User is in game_players
7. State Updates	None
8. Success Response	202 Accepted
9. Error cases	What can go wrong? • 400 Bad Request - >2 players (game no longer valid) • 401 Unauthorized - N/A • 403 Forbidden - N/A • 404 Not Found - Game does not exist • 408 Timed Out - N/A • 409 Conflict - N/A
10. Socket.io Events	• `game:state:update` → All players • `game:turn:changed` → All players • `Game:community:cards:dealt` → All players (if first round) • `Game:community:cards:add` → All players

Field	Description
• Endpoint Name	**Act**
• HTTP Method and Route	`POST /api/games/:game_id/bet`

• Purpose	Place bets
• Authorization	• Must be in the game • Must have sufficient funds

• Request Body	{
	"action": "fold|call|raise|check",
	"amount": 100
	}

• Validation Checks	• Game exists and not in “lobby” state • User's turn • Sufficient funds for bet/raise • Raise amount meets minimum requirements • Check only allowed if no bet to call
• State Updates	• Update `table_player` (bet_amount, player_money) • Update `table` (pot_money) • Advance turn to next active player
• Success Response	202 Accepted
• Error cases	What can go wrong? • 400 Bad Request - >2 players (game no longer valid) • 401 Unauthorized - N/A • 403 Forbidden - Not your turn • 404 Not Found - Game not found • 408 Timed Out - N/A • 409 Conflict - N/A
• Socket.io Events	• `game:bet:placed` → All players • `game:bet:call` - All players • `game:bet:fold` - All players • `game:turn:changed` → All players • `game:balance:update` → Acting player (private)

Field	Description
1. Endpoint Name	**Leave Game**

2. HTTP Method and Route	`POST /api/games/:game_id/leave`
3. Purpose	Player leaves the game
4. Authorization	• Must be in the game
5. Request Body	???
6. Validation Checks	• Game exists • User is in game
7. State Updates	• Remove from `table_player`, handle remaining chips (if you voluntarily leave a game don't take your money with you)
8. Success Response	202 Accepted
9. Error cases	What can go wrong? • 400 Bad Request - You're not in a game • 401 Unauthorized - Trying to leave a game you're not a part of • 403 Forbidden - Not logged in • 404 Not Found - N/A • 408 Timed Out - N/A • 409 Conflict - N/A
10. Socket.io Events	`game:player:left` → All players

Field	Description
1. Endpoint Name	**Send Message**
2. HTTP Method and Route	`POST /api/games/:game_id/chat`
3. Purpose	Send message to game chat
4. Authorization	• Must be in the game
5. Request Body	`{"message": &{message}}`
6. Validation Checks	• Message not empty • user in game • Message within max message length

7. State Updates	• Insert into `chat_messages`
8. Success Response	202 Accepted
9. Error cases	What can go wrong? • 400 Bad Request - >2 players (no longer valid game • 401 Unauthorized - Not in that game • 403 Forbidden - Not logged in • 404 Not Found - N/A • 408 Timed Out - N/A • 409 Conflict - N/A
10. Socket.io Events	`chat:message` → All players in game