Game Description: Bullshit is a card game where players try to get rid of all of their cards by claiming they are playing the next ranks in the sequence. Others can call “bullshit” if they think someone lied, if the call is correct then the player takes the entire stack, else the player who called “bullshit” takes the pile.

All API Endpoints:

Endpoint Name: Create Game

Endpoint: `POST /api/games`

Authorization:

- User must be authenticated

```
Request Body:
{
     "Max_players": 4 // optional, can be 2 - 8
}
```

Validation Checks:

- User is authenticated
- Max players is between 2 and 8

State Updates:

- `INSERT INTO Games` (status='waiting', max_players) RETURNING game_id
- `INSERT INTO Game_Participants` (game_id, user_id, turn_order=1)

Success Response:

```
{
  "game_id": 123,
  "status": "waiting",
  "max_players": 4,
  "participants": [
    {
      "participant_id": 1,
      "user_id": 42,
      "username": "player1",
      "turn_order": 1
    }
  ]
}
```

Error Cases:

- `401 Unauthorized` - Not logged in
- `400 Bad Request` - Invalid max_players value

Endpoint Name: Join Game
Endpoint: `POST /api/games/:game_id/join`
Authorization:
- User must be authenticated
- Game must be in ‘waiting’ status
- Game must not be full
- User not already in game

```
Request Body: None
```

Validation Checks:
- Game exists
- Game status === ‘waiting’
- Current players count < max_players
- User not already in game_participants

State Updates:
- Get next turn order: `SELECT MAX(turn_order) FROM Game_Participants WHERE game_id = ?`
- `INSERT INTO Game_Participants` (game_id, user_id, turn_order=next_turn_order)

Success Response:

```
202 Accepted
```

Error Cases:
- `401 Unauthorized` - Not logged in
- `404 Not Found` - Game doesn't exist
- `403 Forbidden` - Game already started or full

Endpoint Name: Leave game
Endpoint: `POST /api/games/:game_id/leave`
Authorization:
- User must be authenticated
- User must be in the game
- Game must be in 'waiting' status
-

```
Request Body: None
```

Validation Checks:
- Game exists
- Game status === 'waiting'
- User is participant in game

State Updates:
- `DELETE FROM Game_Participants` WHERE game_id=? AND user_id=?
- If no participants left: `DELETE FROM Games` WHERE game_id=? AND NOT EXISTS (SELECT 1 FROM Game_Participants WHERE game_id=?)

Success Response:

```
202 Accepted
```

Error Cases:

- `401 Unauthorized` - Not logged in
- `404 Not Found` - Game doesn't exist
- `403 Forbidden` - Game already started or not in game

Endpoint Name: Start Game
Endpoint: `POST /api/games/:game_id/start`
Authorization:

- User must be authenticated
- User must be in the game
- Game must be in 'waiting' status
- Must have at least 2 players

```
Request Body: None
```

Validation Checks:

- Game exists
- Game status === 'waiting'
- User is participant in game
- Participant count >= 2

State Updates:

- `UPDATE Games` SET status='in_progress', started_at=NOW(), current_rank='A', direction='ascending' WHERE game_id=?
- Shuffle and deal cards (52-card deck, deal evenly among players)
- `INSERT INTO Player_Hands` (participant_id, card_rank, card_suit) for each dealt card
- `UPDATE Game_Participants` SET cards_in_hand=(count of dealt cards) WHERE game_id=?
- `UPDATE Games` SET current_turn=(first participant by turn_order) WHERE game_id=?

Success Response:

```
202 Accepted
```

Error Cases:

- `401 Unauthorized` - Not logged in
- `404 Not Found` - Game doesn't exist
- `403 Forbidden` - Not in game or game already started
- `400 Bad Request` - Not enough players

Endpoint Name: Play Cards
Endpoint: `POST /api/games/:game_id/play`
Authorization:
- User must be authenticated
- User must be active participant in game
- Must be user's turn
- Game must be 'in_progress'

```
Request Body:
{
  "cards": [
    { "rank": "A", "suit": "hearts" },
    { "rank": "A", "suit": "diamonds" }
  ],
  "claimed_rank": "A"
}
```

Validation Checks:
- Game exists and status === 'in_progress'
- User is participant in game
- games.current_turn === participant_id
- cards.length > 0 and cards.length <= 4
- claimed_rank === games.current_rank
- All cards exist in player's hand
- No duplicate cards in request

State Updates:
- `INSERT INTO Game_Actions` with action_type='play', claimed_rank, num_cards_claimed, and was_truthful
- `INSERT INTO Pile_Cards` for each card played
- `DELETE FROM Player_Hands` WHERE participant_id=? AND (card_rank, card_suit) IN (played cards)
- `UPDATE Game_Participants` SET cards_in_hand = cards_in_hand - num_cards WHERE participant_id=?
- `UPDATE Games` SET last_action_id=new_action_id, pile_card_count=pile_card_count+num_cards, challenge_window_expires_at=NOW()+INTERVAL '10 seconds'
- If cards_in_hand reaches 0: `UPDATE Games` SET status='completed', winner_id=user_id, ended_at=NOW()

Success Response:

```
202 Accepted
```

Error Cases:

- `401 Unauthorized` - Not logged in
- `403 Forbidden` - Not your turn, not in game, or game not in progress
- `404 Not Found` - Game doesn't exist
- `400 Bad Request` - Invalid cards, wrong rank claimed, or don't have card

Endpoint Name: Call Bullshit
Endpoint: `POST /api/games/:game_id/challenge`
Authorization:
- User must be authenticated
- User must be active participant in game
- Must NOT be the player who just played
- Must be within challenge window
- Game must be 'in_progress'

```
Request Body: None
```

Validation Checks:
- Game exists and status === 'in_progress'
- User is participant in game
- games.last_action_id is not null
- NOW() <= games.challenge_window_expires_at
- Last action's participant_id !== current user's participant_id
- User hasn't already called BS

State Updates:
- Get last action: `SELECT * FROM Game_Actions WHERE action_id = games.last_action_id`
- Determine challenge outcome: challenge_success = NOT last_action.was_truthful
- Determine loser
- `UPDATE Game_Actions` SET challenge_success WHERE action_id = last_action_id
- `INSERT INTO Game_Actions` with action_type='challenge' for challenger
- `INSERT INTO Game_Actions` with action_type='pickup' for loser
- `INSERT INTO Player_Hands` SELECT loser_participant_id, card_rank, card_suit FROM Pile_Cards WHERE game_id=?
- `UPDATE Game_Participants` SET cards_in_hand = cards_in_hand + pile_count WHERE participant_id = loser_participant_id
- `DELETE FROM Pile_Cards` WHERE game_id=?
- `UPDATE Games` SET current_rank = next_rank(current_rank, direction)
- `UPDATE Games` SET current_turn = get_next_participant(game_id, loser_participant_id)
- `UPDATE Games` SET pile_card_count=0, challenge_window_expires_at=NULL, last_action_id=NULL
- `UPDATE Game_Participants` SET has_called_bs=false WHERE game_id=? (reset for all players)
-

Success Response:

```
202 Accepted
```

Error Cases:

- `401 Unauthorized` - Not logged in
- `403 Forbidden` - Not in game, challenging own play, or already challenged
- `404 Not Found` - Game doesn't exist
- `400 Bad Request` - Nothing to challenge or challenge window expired

Endpoint Name: Pass Turn
Endpoint: `POST /api/games/:game_id/pass`
Authorization:

- User must be authenticated
- User must be a player in this game
- Must be the user's turn
- Game must be in "in_progress" state

```
Request Body: None
```

Validation Checks:

- Game with game_id exists
- User is in Game_Participants table for this game
- games.current_turn === user's participant_id
- games.status === "in_progress"
- Player has at least one card in hand

State Updates:

- `INSERT INTO Game_Actions` with action_type='pass'
- `UPDATE Games` SET current_turn = get_next_participant(game_id, current_participant_id)

Success Response:

```
202 Accepted
```

Error Cases:

- `401 Unauthorized` - Not logged in
- `403 Forbidden` - Not your turn or not in game
- `404 Not Found` - Game doesn't exist
- `400 Bad Request` - No cards left

Endpoint Name: Get Game State
Endpoint: `GET /api/games/:game_id`
Authorization:

- User must be authenticated

- User must be participant in game

```
Request Body: None
```

Validation Checks:

- Game exists
- User is participant in game

Success Response:

```
{
  "game_id": 123,
  "status": "in_progress",
  "current_turn": {
    "participant_id": 1,
    "user_id": 42,
    "username": "player1"
  },
  "current_rank": "5",
  "direction": "ascending",
  "pile_count": 8,
  "max_players": 4,
  "can_challenge": true,
  "challenge_window_expires_at": "2025-11-17T10:30:10Z",
  "participants": [
    {
      "participant_id": 1,
      "user_id": 42,
      "username": "player1",
      "display_name": "Alice",
      "turn_order": 1,
      "cards_in_hand": 8,
      "is_active": true
    },
    {
      "participant_id": 2,
      "user_id": 43,
      "username": "player2",
      "display_name": "Bob",
      "turn_order": 2,
      "cards_in_hand": 12,
      "is_active": true
    }
  ],
  "your_hand": [
    { "rank": "5", "suit": "hearts" },
    { "rank": "5", "suit": "diamonds" },
    { "rank": "7", "suit": "clubs" }
    // ... only if user is in game
  ],
  "recent_actions": [
    {
      "action_id": 456,
      "participant_id": 1,
      "username": "player1",
      "action_type": "play",
      "claimed_rank": "4",
      "num_cards": 2,
      "timestamp": "2025-11-17T10:30:00Z"
    }
  ]
}
```

Error Cases:

- `401 Unauthorized` - Not logged in
- `403 Forbidden` - Not in this game
- `404 Not Found` - Game doesn't exist

Endpoint Name: List Available Games
Endpoint: `GET /api/games`
Authorization:

- User must be authenticated

```
Request Body: None
```

Validation Checks:

- User is authenticated

Success Response:

```
{
  "games": [
    {
      "game_id": 123,
      "status": "waiting",
      "max_players": 4,
      "current_players": 2,
      "created_at": "2025-11-17T10:00:00Z",
      "participants": [
        {
          "user_id": 42,
          "username": "player1"
        },
        {
          "user_id": 43,
          "username": "player2"
        }
      ]
    }
  ],
  "total": 15,
  "limit": 20,
  "offset": 0
}
```

Error Cases:

- `401 Unauthorized` - Not logged in

Endpoint Name: Send Chat Message
Endpoint: `POST /api/games/:game_id/chat`
Authorization:

- User must be authenticated
- User must be participant in game

```
Request Body:
{
  "message": "Good luck everyone!"
}
```

Validation Checks:

- Game exists
- User is participant in game
- Message is not empty
- Message length <= 500 characters

State Updates:

- `INSERT INTO Chat_Messages (game_id, user_id, message_text)`
- `VALUES (?, ?, ?)`
- `RETURNING message_id, sent_at;`

Success Response:

```
202 Accepted
```

Error Cases:

- `401 Unauthorized -` Not logged in

- `403 Forbidden -` Not in game
- `404 Not Found -` Game doesn't exist
- `400 Bad Request` - Empty or too long message

Endpoint Name: Get Chat History
Endpoint: `GET /api/games/:game_id/chat`
Authorization:

- User must be authenticated
- User must be participant in game

```
Request Body:
{
  "message": "Good luck everyone!"
}
```

Validation Checks:

- Game exists
- User is participant in game
- Message is not empty
- Message length <= 500 characters

State Updates:

- `INSERT INTO Chat_Messages (game_id, user_id, message_text)`
- `VALUES (?, ?, ?)`
- `RETURNING message_id, sent_at;`

Success Response:

```
{
  "messages": [
    {
      "message_id": 789,
      "user_id": 42,
      "username": "player1",
      "display_name": "Alice",
      "message": "Good luck everyone!",
      "sent_at": "2025-11-17T10:30:00Z"
    }
  ],
  "has_more": false
}
```

Error Cases:

- `401 Unauthorized` - Not logged in
- `403 Forbidden` - Not in game

- `404 Not Found` - Game doesn't exist

Socket.io Events Planning

Lobby Connections			
Event	**Data flow**	**Description**	
join_game	Player to Server	Player Joins lobby	socket.join('game_{id}')
player_joined	Server to the Lobby	Lets players know someone joined	io.to(room).emit()
leave_game	Player to Server	Player Leave Lobby	socket.leave('game_{id}'}')
player_left	Server to the Lobby	Lets players know someone left	io.to(room).emit()
client:request_state	Player to Server	Allows player to resync if an issue with internet or page refresh occurs	
disconnect	In socket.io	A player leaves lobby	
game:state	Server to the Player	Syncs the match to the players	socket.emit()

Gameplay

play_card	Player to Server	Card is placed and server updates
card_played	Server to the Lobby	Once card was played and Server updates it will announce to players
bullshit_option	Server to the Lobby	Allows players to call bullshit on a player
bullshit_called	Player to Server	Player challenges the played card
bullshit_result	Server to the Lobby	the results of the call

turn_ended	Server to the Lobby	player turns ended and moves to next
game:finished	Server to the Lobby	A player Wins

Chat

chat:send	Player to Server	A message is sent by a player
chat:message	Server to the Lobby	Displays chat message

Server Setup:

```
const express = require(“express”);
const http = require (“http”);
const {Server} = require(“socket.io);
const session = require(“./sessionMiddleware”); // whatever our middleware is

const app= express();
app.use (express.json());
app.use(session):

const server = http.createServer(app);
const io = new Server(server, {
          cors:{ origin : “*”}
          });

//Share session with socket.io
io.use ((socket, next) => {
          session (socket.request, {}, next);
});

//Client Connects/ Authentication
function authSocket(socket, next) {
const user = socket.request.session?.user;
if(!user){
console.log(“unauthorized socket: ${socket.id}”);
next(new Error(“Unauthorized”));
return socket.disconnect(true);
}

//Authenticated User
socket.user = user;
console.log(‘Socket ${socket.id} authenticated as ${user.username}’);
next();
}
io.use(authSocket);



// Event Handlers
io.on(“connection”, (socket) => {
const user = socket.user;
console.log(‘${user.username} connected with socket ${socket.id}’);

//Joining a game
socket.on (“join_game”, (gameId) => {
socket.join(‘game_${gameId}’);
console.log(‘User ${socket.id} joined game_${gameId}’);
});

Leaving Lobby:
socket.on (“leave_game”,(gameId) => {
socket.leave(‘game:${gameId}’);
console.log(‘Socket ${socket.id} left game:${gameId}’);
});

socket.on(“disconnect”, () => {
console.log(‘${user.username} disconnected’);
});
});
```