# Create Game

**1. Endpoint Name**

Create Game

**2. HTTP Method & Route**

`POST /api/games`

**3. Purpose**

Create a new game lobby and make the current user the host. Returns the initial lobby state and join codes (if we use join code or only invite)

**4. Authorization**

- Must be authenticated (valid session/JWT).
- Users must not already be host of another active game in `"lobby"` or `"playing"` state (to avoid multi-host).

# 5. Request body example

**5. Request Body example;**

```
{
  "name": "Miguel's Word Room",
  "max_players": 5,
  "is_private": true,
  "difficulty": "normal, hard",
  "round_limit": 3          // still unsure how my many
rounds
}
```

## 6. Validation Checks (be explicit)

- The user is authenticated.
- `name` is non-empty and max length is 10 chars).
- `max_players` is an integer between 2 and 8.(for multiplayer)
- `difficulty` is in the allowed set: `["easy","normal","hard"]`.
- If `round_limit` provided, it's an integer ≥ 1.
- The user is not already hosting an active game (`games.host_id = user.id AND state IN ('lobby','playing')`).
- No DB or server error during creation.

# 7. State Updates (DB changes) & 8. Success Response

### 7. State Updates (DB changes)

- INSERT into `games`:
  - `host_id = current_user.id`
  - `state = "lobby"`
  - `max_players`, `is_private`, `difficulty`, `round_limit`, etc.
  - Optionally generate a `join_code` like 6-char alphanumeric.
- INSERT into `game_players`:
  - `game_id = new_game.id`
  - `user_id = current_user.id`
  - `is_host = true`
  - `score = 0`

### 8. Success Response

- **201 Created** (it's literally creating a resource; it doesn't need async processing)

```
{
  "game": {
    "id": 123,
    "code": "ABCD12",
    "name": "Miguel's Word Room",
    "state": "lobby",
    "max_players": 5,
    "is_private": true,
    "difficulty": "normal",
    "round_limit": 3,
    "host_id": 44,
    "players": [
      {
        "id": 42,
        "name": "Miguel",
        "is_host": true,
        "score": 0
      }
    ]
  }
}
```

# 9. Error Case

- **401 Unauthorized** – user not logged in.
- **400 Bad Request** – invalid body fields (bad `max_players`, bad `difficulty`, etc.).
- **409 Conflict** – user already hosting another active game.
- **500 Internal Server Error** – DB failure.

**Data:**

```
{
  "game_id": 123,
  "code": "ABCD12",
  "name": "Miguel's Word Room",
  "state": "lobby",
  "max_players": 6
```

    - }

# Start Game

**1. Endpoint Name**

Start Game

**2. HTTP Method & Route**

`POST /api/games/:game_id/start`

**3. Purpose**

Transition from lobby to playing: lock in the player list, initialize the first round, choose the secret word(s) randomly, and set the current player/turn order if applicable.

# Authorization & Request Body Example

**4. Authorization**

- Must be authenticated.
- Must be a player in this game.
- Must be the **host** (`game.host_id === user.id`).
- The game must be in `"lobby"` state.

**5. Request Body**

Optional, if you want to override default settings:

```
{
  "difficulty": "hard",         // overrides if host changes it last
minute
  "round_limit": 3
}
```

# Validation Checks

- The user is authenticated.
- Games exist.
- The user is in `game_players` for this game.
- User is host (`game.host_id === user.id`).
- `game.state === "lobby"`.
- `player_count >= min_players` (e.g. at least 2).
- If the body contains new settings, they are valid (difficulty/round_limit checks).
- The game is not already `"playing"` or `"ended"`.

# State Updates

- UPDATE `games`:
  - `state = "playing"`
  - lock in `difficulty`, `round_limit`.
  - set `current_round = 1`.
- Generate secret word(s) based on difficulty:
  - This is **server-side only**; never sent to clients.
  - Possibly insert the initial row in the rounds table: (`game_id`, `round_number = 1`, `secret_word = "APPLE"`, `status = "active"`, etc.).
- Initialize turn order if needed.
- (All in a transaction to avoid partial state.)

# Success Response

- **202 Accepted**

```
{
  "status": "accepted",
  "game": {
    "id": 123,
    "state": "playing",
    "current_round": 1,
    "difficulty": "hard",
    "round_limit": 5
  }
}
```

# 9. Error Cases

- **401 Unauthorized** – not logged in.
- **403 Forbidden** – user not in game OR not host.
- **404 Not Found** – game doesn't exist.
- **409 Conflict** – game already started or ended; or not enough players.

**Data:**

```
{
  "game_id": 123,
  "state": "playing",
  "current_round": 1,
  "players": [
    { "id": 42, "name": "Miguel", "score": 0 },
    { "id": 77, "name": "Luis", "score": 0 }
  ]
}
```

    - 
- **Event:** `game:state:update`
    - **Scope:** `room:game:<game_id>`
    - **Trigger:** after initial round/turn state is created.

# Socket.io Events Summary

- 4 Socket.io events
- 2 public broadcast events (to all players in a room)
- 2 private events (to a single player)

| | |
|---|---|
| `game:lobby:created` – *private*<br>- **Triggered by:** `POST /api/games` (Create Game)<br>- **Sent to:** Host only<br>- **Data:** New game id, join code, lobby settings (maxPlayers, difficulty, etc.) | `game:lobby:updated` — *public*<br>- **Triggered by:** `POST /api/games/:game_id/join` (and future join/leave/kick)<br>- **Sent to:** All players in the lobby<br>- **Data:** Game id, full player list, host, current player count/max |
| `game:state:update` – *public*<br>- **Triggered by**: `POST /api/games/:game_id/start` and major state changes<br>- **Sent to**: All players in the game<br>- **Data**: Game state (lobby/playing/finished), current round, current player id, shared board info | `game:private:update` — *private*<br>- **Triggered by**: `Join Game / Start Game / Get Game State` when a player's personal data changes<br>- **Sent to**: One specific player<br>- **Data**: That player's private info (their guesses, notes, any hidden data not visible to others) |