Milestone 3: API Design

https://docs.google.com/presentation/d/1SIm-HC4W8OTE4meMLj8kmdjloe4RknUErmiu-SO3PsY/edit?slide=id.p#slide=id.p

1. Game Overview

Scrabble is a classic word game where players use letter tiles to form words on a grid. Each word formed scores points based on the value of the tile and the bonus point squares on the board. The game ends when the tiles run out, and once one player empties their hand of all tiles. The highest scoring player wins. Our online version supports real-time gameplay, chat, and synchronized state using Socket.IO.

2. API Endpoints

Endpoint Name: Create Game

HTTP Method & Route: POST/api/games

Purpose: Server creates a new game and returns the new game_id. The creating player becomes the first player in the lobby.

Authorization:

- User must be authenticated
- User must be logged in (have an active session)

Request Body:

```
{
        “maxPlayers”: number,
        “Settings”: object
}
```

Validation Checks:

1. Requesting users exists.
2. max_players is valid (must be at least 2 players)
3. Player does not have an existing game

State Updates:

- INSERT new row into games table
- INSERT creator into game_participants table as player 1

Success Response:

202 Accepted

Error Cases:

- 401 Unauthorized - Not logged in
- 400 Bad Request - Invalid max_players
- 500 Internal Server Error - Database failure

Socket.io Events Triggered:

- lobby:game:created

Endpoint Name: Join Game

HTTP Method & Route: POST/api/games/:game_id/join

Purpose: Players join an open game lobby before the game starts

Authorization:

- User must be authenticated
- User must not already be in this game
- User must not already be in a different active game

Request Body:

Validation Checks:

- Game with game_id exists
- Game with has not started
- Game is not full
- User is not already in game
- Lobby is accepting players

State Updates:

- INSERT new row into game_participant table linking user->game
- Increment game's current_players count

Success Response:

202 Accepted

Error Cases:

- 401 Unauthorized - user not logged in
- 403 Forbidden - user is already in a game or in another active game
- 404 Not Found - game doesn't exist
- 409 Conflict - game already started or full

Socket.io Events Triggered:

- game:player:joined
- lobby:game:updated

Endpoint Name: Start Game

HTTP Method & Route: POST /api/games/:game_id/start

Purpose: Transitions a lobby to an active game: initialize the tiles, deal the tiles to each player, set initial board state and scores, choose the starting player, mark the game as active

Authorization:

- User must be authenticated
- User must be the lobby owner
- Game must be in waiting state

Request Body:

Validation Checks:

- game with game_id exists
- game has at least 2 players
- game has not already started
- requesting user is the lobby owner

State Updates:

- update games table > set status to "active" and save start_time
- create and shuffle tle bag
- deal 7 random tiles to each player (insert into player_hand table)
- set current_turn_player_id
- initialize the board

Success Response:

{
    "game_id": 1,

“status”: “active”,
“current_player_id”: 1,
“players”: [
{ “player_id”: 1, “name”: “Random”, “score”: 0},
{ “player_id”: 2, “name”: “Boop”, “score”: 0}
]
}

Error Cases:

- 401 Unauthorized - user not logged in
- 403 Forbidden - user is not the lobby owner
- 404 Not Found - game does not exist
- 409 Conflict - not enough players or already started

Socket.io Events Triggered:

- game:started
- game:hand:update
- game:turn:change

Endpoint Name: Get Game State

HTTP Method & Route: GET api/games/:game_id

Purpose:
Returns the full current game information so the frontend can show the board, player scores, and whose turn it is

Authorization:

- User must be logged in
- User must be in the game

Request Body:

Validation Checks:

- Game exists
- User is part of the game

State Updates:

- None (Read only endpoint)

Success Response:

{
“game_id”: 1,
“status”: “”active”,
“players”: [
“players”: [

{ “player_id”: 1, “name”: “Random”, “score”: 2},
{ “player_id”: 2, “name”: “Boop”, “score”: 90}
],
“current_player_id”: 1,
“board”: [...],
“your_hand”: [“A”, “B”, “C”. “D”, “E”, “F”, “G”]
:}

Error Cases:

- 401 Unauthorized - not logged in
- 403 Forbidden - user not in game
- 404 Not found - game does not exist

Socket.io Events Triggered:
N/A (only returns data when requested)

Endpoint Name: Submit Word

HTTP Method & Route: POST /api/games/:game_id/submit-word

Purpose: Check word for correctness, score it if correct, place tiles on the board for everyone, refill the player’s hand, go to the next turn, and if necessary, end the game.

Authorization:

- User must be logged in
- Must be the current turn for the player

Request Body:

{
“current_hand”: string[],
“tile_placements”: {
x: number,
y: number,
letter: string}
}

Validation Checks:

- User is in this game and logged in
- Game exists and is active
- The current turn matches the requesting player
- Placement letters are a subset of the current hand
- All (x,y) coordinates in the board
- Target placements are empty

- The word is in a horizontal or vertical line with no gaps
- At least one tile is adjacent to an existing tile. On first turn, one tile must be in the center
- Word must exist in the dictionary
- Any other words formed must exist in the dictionary

State Updates:
- INSERT INTO board_tiles (game_id, row, col, letter, placed_by, placed_at)
- DELETE FROM player_tiles (Delete tiles from hand)
- INSERT INTO player_tiles (game_id, user_id, letter) (Draw from tile bag)
  DELETE FROM tile_bag (remove those tiles from tile bag)
- UPDATE game_participants
  SET score = score + points_earned (Update player score)
- UPDATE games
  SET current_turn_user_id = next_player (Go to next turn)
- UPDATE games
  SET status = 'completed' (Check if bag is empty and if hand is empty if so end game)

Success Response:

```
{
    “points”: string,
    “next_player_id”: number,
    “ended”: boolean
}
```

Error Cases:
- 401 Unauthorized - Not logged in
- 403 Forbidden - not current player’s turn
- 404 Not found - Game doesn’t exist
- 422 Unprocessed Entity - Invalid Move

Socket.io Events Triggered:
- game:board:update
- game:score:update
- game:hand:update
- game:turn:change
- game:ended (if needed)

Endpoint Name: Pass turn

HTTP Method & Route: POST/api/games/:game_id/pass

Purpose: For passing a turn without playing a word

Authorization:

- User must be logged in
- Must be the current turn for the player

Request Body:

{
}
(No data needed)

Validation Checks:

- Game exists
- It is the current player's turn

State Updates:

- Go to next player's turn
- If all players have passed and the bag is empty then end game

Success Response:

{
"next_player_id": number,
"ended": boolean
}

Error Cases:

- 401 Unauthorized - Not logged in
- 403 Forbidden - not current player's turn
- 404 Not found - Game doesn't exist

Socket.io Events Triggered:

- game:turn:pass
- game:turn:change
- game:ended (If it's over)

Endpoint Name: Swap Tiles

HTTP Method & Route: /api/games/:game_id/swap

Purpose: Swapping Tiles..

Authorization:

- User must be authenticated
- Must be the current turn for the player
- Game must be active

Request Body:

```
{
"tiles_to_swap": ["A", "B", "C"]
}
```

Validation Checks:

- Game exists and is active
- It is the current player's turn
- Tiles to swap exist in player's hand
- Tile bag has enough tiles remaining
- At least 1 tile specified for swap
- Maximum 7 tiles can be swapped

State Updates:

- DELETE FROM player_tiles (Remove swapped tiles from hand)
- INSERT INTO tile_bag (Return swapped tiles to bag)
- Shuffle tile bag
- DELETE FROM tile_bag (Draw new tiles equal to swapped count)
- INSERT INTO player_tiles (Add new tiles to player's hand)
- UPDATE games SET current_turn_user_id = next_player (Go to next turn)

Success Response:

```
{
 "swapped_count": 3,
  "next_player_id": 2,
  "ended": false
}
```

Error Cases:

- 401 Unauthorized - Not logged in
- 403 Forbidden - Not current player's turn
- 404 Not Found - Game doesn't exist
- 422 Unprocessable Entity - Invalid tiles or not enough in bag

Socket.io Events Triggered:

- game:tiles:swap
- game:turn:change
- game:ended (if needed)

Endpoint Name: Leave game

HTTP Method & Route: /api/games/:game_id/leave

Purpose: To leave the game.

Authorization:
- User must be authenticated
- User must be in the specified game

Request Body:
{ }

Validation Checks:
- Game exists
- User is currently in the game
- User has valid session

State Updates:
- DELETE FROM game_participants (Remove player from game)
- UPDATE games SET current_players = current_players - 1
- If game is in lobby: UPDATE games status if needed
- If game is active and player was current turn: UPDATE games SET current_turn_user_id = next_player
- If only one player remains in active game: UPDATE games SET status = 'completed'
- Return player's tiles to tile bag if game is active

Success Response:

{
 "left": true,
 "game_status": "active",
 "remaining_players": 2
}

Error Cases:
- 401 Unauthorized - Not logged in
- 403 Forbidden - User not in this game
- 404 Not Found - Game doesn't exist

Socket.io Events Triggered:
- game:player:left
- lobby:game:updated (if in lobby)
- game:turn:change (if was current player's turn)
- game:ended

Endpoint Name: Send Chat Message

HTTP Method & Route: POST /api/chat

Purpose: Send a chat message to all logged-in players in the global chat

Authorization: User must be logged in

Request Body:
{
"message": "delete this game"
}
Validation Checks:
- Message is not empty
- User is logged in and has a valid session

State Updates:
- Save the message to a global message table with the player name, id, and timestamp for history

Success Response:
{
"player_id": 1,
"player_name": "Boop",
"message": "delete the game",
"created_at": "2025-11-16-16T22:00:00Z"
}
Error Cases:

- **401 Unauthorized - User not logged in**

Socket.io Events Triggered:
global:chat:message

Socket.io Events

Global Events

1.
Event: global:chat:message

Scope: All players on the website

Trigger: Anyone posts a message in the global chat

Data:
{

```
    player_name: string,
    player_id: number,
    Message: string
}
```

Public Lobby Events

1.
Event: game:player:joined

Scope: All players in the game room

Trigger: A player joins the game

Data:
```
{
    game_id: number,
    player_id: number,
    player_name: string,
    players: string[]
}
```

2.
Event: game:player:left

Scope: All players in the game room

Trigger: A player leaves the game

Data:
```
{
    game_id: number,
    player_id: number
}
```

3.
Event: game:board:update

Scope: All players in the game room

Trigger: A player submits a valid word

Data:
{
    game_id: number,
    placements: [
        {letter: char, x: number, y: number},
        {letter: char, x: number, y: number},
    ],
    board: [...]
}

4.
Event: game:score:update

Scope: All players in the game room

Trigger: After the server validates and scores the word submitted

Data:
{
    game_id: number,
    player_id: number,
    word: string,
    points: number,
    scores: number[]
}

5.
Event: game:turn:change

Scope: All players in the game room

Trigger: After a valid action that ends a turn

Data:
{
    game_id: number,
    next_player: number
}

6.
Event: game:turn:pass

Scope: All players in the game room

Trigger: A player uses “Skip Turn”

Data:
{
    game_id: number,
    player_id: number
    reason: string
}

7.
Event: global:chat:message

Scope: All players in the lobby

Trigger: A player sends chat message

Data:
{
    player_name: string
    player_id: number,
    Message: string
}

8.
Event: game:ended

Scope: All players in the game room

Trigger: The tile bag is empty and no more moves are possible. Or a player uses all tiles

Data:
{
    game_id: number,
    final_score: number[]
    winner: string
}

9.
Event: game:started

Scope: All players in the game room

Trigger: The lobby owner starts the game

Data:

{
    game_id: number,
}

10.
Event: lobby:game:created

Scope: All users in the lobby

Trigger: After a new game is created successfully

Data:
{
    game_id: number,
    created_by: number,
    creator_name: string,
    current_players: number,
    max_players: number
    status: string,
    created_at: string
}

11.
Event: lobby:game:updated

Scope: All users in the public lobby

Trigger: After a player joins a game successfully

Data:
{
    game_id: number
    current_players: number,
    max_players: number
    status: string,
}

Player (Private) Events

1.
Event: game:hand:update

Scope: Only the player receiving the tiles

Trigger: Player draws a tile or tiles refill after submitting a word or swapping tiles

Data:
{
    game_id: number,
    player_id: number,
    tiles: char[7]
}

2.
Event: game:tiles:swap

Scope: Only the player who swapped tiles

Trigger: Player swaps tiles in hand

Data:
{
    game_id: number,
    player_id: number,
    new_tiles: char[7]
}

3.
Event: game:action:invalid

Scope: Only the player attempting an invalid action

Trigger: Player makes an invalid action

Data:
{
    player_id: number,
    error_message: string,
}