**GET /newgame**

Clear the table “GAMEBOARD” in database “gamestate.db”. The table holds the current “game” variable, which is the game state of the ongoing game.

Also redirect user to “gomoku.html”.

**POST /startgame**

Startgame request to be called by player 1. Frontend posts a JSON as described and the backend will initialize the game. The request will return a GameBoard JSON.

**Request body**

JSON object (same a Player object)

**type** char

The type of player 1. ‘X’ or ‘O’.

**id** string

The ID token of user returned by google.

**Response**

The game JSON.

**GameBoard** JSON object

**gameID** string

A string of UUID of the game.

**p1** Player object

Player p1 of the game.

**p2** Player object

Player p2 of the game.

**gameStarted** bool

Boolean showing whether the game has started.

**turn** int

Integer showing p1 / p2 ‘s turn. (turn == 1 means p1’s turn)

**boardState** char[][]

The actual game board of the current game. Each entry is either ‘\u0000’, ‘X’ or ‘O’.

**winner** int

Integer showing if p1 or p2 has won the game.

**isDraw** bool

Boolean showing if the game has ended with a draw.

**POST /joingame**

Join game request to be called by player 2. Frontend posts a JSON as described and the backend will add this player to the game. The request will return a GameBoard object JSON. This comes with assumption that 1: the second player type is handled at frontend; 2: when we add more feature like lobby, frontend will store all existing game ID and allow user to join any game they want. For now we only care about one game running in background. When we add “lobby”, we should also add a field gameID to the request body.

**Request body**

JSON object (same a Player object)

**type** char

The type of player 1. ‘X’ or ‘O’.

**id** string

The ID token of user returned by google.

**Response**

The game JSON.

**GameBoard** JSON object

**gameID** string

A string of UUID of the game.

**p1** Player object

Player p1 of the game.

**p2** Player object

Player p2 of the game.

**gameStarted** bool

Boolean showing whether the game has started.

**turn** int

Integer showing p1 / p2 ‘s turn. (turn == 1 means p1’s turn)

**boardState** char[][]

The actual game board of the current game. Each entry is either ‘\u0000’, ‘X’ or ‘O’.

**winner** int

Integer showing if p1 or p2 has won the game.

**isDraw** bool

Boolean showing if the game has ended with a draw.

**GET /getgame**

Retrieve the ongoing GameBoard from database and return.

**Response**

The game JSON.

**GameBoard** JSON object

**gameID** string

A string of UUID of the game.

**p1** Player object

Player p1 of the game.

**p2** Player object

Player p2 of the game.

**gameStarted** bool

Boolean showing whether the game has started.

**turn** int

Integer showing p1 / p2 ‘s turn. (turn == 1 means p1’s turn)

**boardState** char[][]

The actual game board of the current game. Each entry is either ‘\u0000’, ‘X’ or ‘O’.

**winner** int

Integer showing if p1 or p2 has won the game.

**isDraw** bool

Boolean showing if the game has ended with a draw.

**POST /move/{playerID}**

When frontend is posting a move request, add playerID as path parameter. Also, post request body as described.

**Request body**

**x** int

The x coordinate of playerID’s move.

**y** int

The y coordinate of playerID’s move.

**Response**

Return: object

**moveValidity** bool

Whether this move is valid.

**code** int

Specific error code.

**message** str

Error message to be shown. If move is valid, this field is empty