Socket.io API Specification and Database Schema

May 17, 2019

Contents

1 Socket.io API

There are no unified rules to write Socket.IO APIs. Therefore, we came up with our own way to explain how the system communicates within the components.

In a Socket.IO API, there are four important parts. First, Socket.IO supports different types of communication, like broadcast, multicast, or unicast. Second, as the communication is bi-directional, we have to specify the direction of a message. Third, we have to specify the unique event names. Finally, the back-end and front-end should agree on the arguments for each message.

If the message has a complex structure, we use JSON and specify the key names and value types in the document.

- **Event name:** connection
  Sender: client
  Receiver: server
  Message type: unicast
  Arguments: The newly created socket object
  Note: This event would be automatically fired when a new socket is created.

- **Event name:** add_user
  Sender: client
  Receiver: server
  Message type: unicast
  Arguments: Player's name
  Note: This event would be fired when the player enters his nickname and press the button.

- **Event name:** ready
  Sender: client
  Receiver: server
  Message type: unicast
  Arguments: None
  Note: This event would be fired when the player is ready to start the game.

- **Event name:** move

  Sender: client

  Receiver: server

  Message type: unicast

  Arguments: The move direction as strings. Valid moves: left, right, up, down

  Note: This event would be fired when the player press an arrow key in his turn.

- **Event name:** disconnect

  Sender: client

  Receiver: server

  Message type: unicast

  Arguments: None

  Note: This event would be fired automatically when the player loses connection to the game.

- **Event name:** gaming

  Sender: server

  Receiver: players in the same game

  Message type: multicast

  Arguments: {

  result: boolean

  }

  Note: The result indicate if both of the players are ready to start the game.

- **Event name:** paired

  Sender: server

  Receiver: players in the same game

  Message type: multicast

  Arguments: {

  result: boolean,

  userName: string

  }

  Note: This event would be fired when a player is waiting for pairing. If the result is true. the userName would be the opponent's name. Otherwise it would be null.

- **Event name:** score

  Sender: server

  Receiver: players in the same game

  Message type: multicast

  Arguments: {

  score: integer

  }

  Note: This event would be fired when the score changes

- **Event name:** player_block_data

  Sender: server

  Receiver: players in the same game

  Message type: multicast

  Arguments: {

  nextBlockIndex: integer,

  currentPlayer: string,

  nextPlayer: string,

  }

  Note: This event would be fired when a step finishes.

- **Event name:** game_contents

  Sender: server

  Receiver: players in the same game

  Message type: multicast

  Arguments: {

  gameField: 2D array

  }

  Note: The 2D array represents the game panel.

- **Event name:** game_over

  Sender: server

  Receiver: players in the same game

  Message type: multicast

  Arguments: {

totalScore: integer,

players: 2D string array,

indivScores: 2D integer array,

totalScoreRanking: integer,

numberOfGamesInDB: integer }

Note: This event would be fired when the game is over.

- **Event name:** game_again

  Sender: client

  Receiver: server

  Message type: unicast

  Arguments: a boolean value standing for the willingness

  Note: This event would be fired when the player presses ”yes” when asking if he wants to play again.

- **Event name:** same_player

  Sender: client

  Receiver: server

  Message type: unicast

  Arguments: a boolean value standing for the willingness

  Note: This event would be fired when the player presses ”yes” when asking if he wants to play again with the same player.

- **Event name:** survey

  Sender: client

  Receiver: server

  Message type: unicast

  Arguments: {survey: object 2D array}

  Note: This event would be fired when the player presses ”yes” when asking if he wants to play again with the same player.

- **Event name:** rename

  Sender: server

  Receiver: client

  Message type: unicast

Arguments: a string value standing for the changed name

Note: This event would be fired when the two players have the same names.

- **Event name:** leaving

  Sender: server

  Receiver: players in the same game

  Message type: multicast

  Arguments: None

  Note: This event would be fired when one player quits the game.

2 Database Schema

In this project, we deployed MongoDB, a NoSQL database, and designed database schemes with Mongoose to store game logs for further research. The following is specifications of the database schemes.

2.1 Game Model Schema

- **Key:** _id

  Type: String

  Required: True

  Note: Room id generated for the socket room will be used as the game id.

- **Key:** player1

  Type: playerSchema

  Required: True

  Note: playerSchema is a sub-document schema for player information

- **Key:** player2

  Type: playerSchema

  Required: True

  Note: playerSchema is a sub-document schema for player information

- **Key:** totalScore

  Type: Number

Default: 0

Required: True

Note: Total score that two players achieve in one game.

- **Key:** totalTime

  Type: Number

  Default: 0

  Required: True

  Note: Total time that two players spend on the game.

- **Key:** totalSteps

  Type: Number

  Default: 0

  Required: True

  Note: Total steps that two players play on the game.

- **Key:** linesPerMin

  Type: An Array of Number

  Default: 0

  Required: True

  Note: Lines eliminated by two players per minute.

- **Key:** steps

  Type: An Array of stepSchema

  Note: stepSchema is a sub-document schema for each step information.

- **Key:** timestamps

  Note: Automatically generated time stamp.

2.2 Player Schema

- **Key:** playerId

  Type: String

  Required: True

  Note: Socket id will be used as the player id.

- **Key:** individualScore

  Type: Number

  Default: 0

  Required: True

  Note: Individual score that the player achieves in one game.

- **Key:** stepsPlayed

  Type: Number

  Default: 0

  Required: True

  Note: The number of steps played by the player

- **Key:** playAgain

  Type: Boolean

  Default: false

  Required: True

  Note: If the player choose to play again after a game.

- **Key:** playWithSameAgain

  Type: Boolean

  Default: false

  Required: True

  Note: If the player choose to play again with the same collaborator.

- **Key:** ifQuit

  Type: Boolean

  Default: false

  Required: True

  Note: If the player quits during a game.

- **Key:** survey

  Type: Array

  Default: []

  Note: Answers to survey questions will be stored to this array in order.

2.3 Step Schema

- **Key:** playerId

  Type: String

  Required: True

  Note: The player id of the player that played at this step.

- **Key:** score

  Type: Number

  Required: True

  Note: Score achieved at this step.

- **Key:** numOfRotations

  Type: Number

  Required: True

  Note: The number of rotations made by the player at this step.

- **Key:** timestamps

  Note: Automatically generated time stamp.