Server Node

Table of contents

Controllers

Module: controllers/AuthController

Functions

- generateJWT

Functions

generateJWT

- **generateJWT**(`requestData`): `Promise<string>`

Generate the user access token that identifies each connected user, and create their user within the database.

Parameters

Name	Type	Description
`requestData`	`Object`	The username, game code, and type of user in which is generating their aut
`requestData.color?`	`Color`	If the user is a client node user, they will provide the color for their player
`requestData.gameCode`	`string`	The game code for the game that the user is going to be related to.
`requestData.userType`	`"Game"` | `"Client"`	Whether the user being created is a game or client node user.
`requestData.username`	`string`	The username of the user sending the request to generate a JWT token.

Returns `Promise<string>`

Resolution code with JWT embedded.

Module: controllers/ClientController

Functions

- joinGame

Functions

joinGame

- **joinGame**(`req`, `res`): `Promise<never>`

Allow user to join a game assuming they provide their username and the game code.

Parameters

Name	Type	Description
`req`	`FastifyRequest<{ Body: { game_code: string ; username: string } }, RawServerDefault, IncomingMessage, FastifySchema, FastifyTypeProviderDefault, unknown, FastifyBaseLogger, ResolveFastifyRequestType<FastifyTypeProviderDefault, FastifySchema, { Body: { game_code: string ; username: string } }>>`	The user request containing their username and the game id.
`res`	`FastifyReply<RawServerDefault, IncomingMessage, ServerResponse<IncomingMessage>, RouteGenericInterface, unknown, FastifySchema, FastifyTypeProviderDefault, unknown>`	The response to indicate to the user whether that their request succeeded.

Returns `Promise<never>`

A resolution, or rejection, to indicate if the request was successful.

Module: controllers/GameController

Functions

- checkWinner
- createGame
- generateQuestion
- getThemePacksAPI
- movePlayer
- nextPlayer
- playerTurnOrder
- questionAnswer
- questionEnd
- startGame
- turn

Functions

checkWinner

- **checkWinner**(`context`): `Promise<boolean>`

Check if any players are in the winner state.

Parameters

Name	Type
`context`	`Context`

Returns `Promise<boolean>`

Whether there is a winning player in the game.

createGame

- **createGame**(`req`, `res`): `Promise<never>`

Creates a game object from an incoming request.

Parameters

Name	Type	Description
`req`	`FastifyRequest<{ Body: { theme_pack: string } }, RawServerDefault, IncomingMessage, FastifySchema, FastifyTypeProviderDefault, unknown, FastifyBaseLogger, ResolveFastifyRequestType<FastifyTypeProviderDefault, FastifySchema, { Body: { theme_pack: string } }>>`	Incoming request object from the game node.
`res`	`FastifyReply<RawServerDefault, IncomingMessage, ServerResponse<IncomingMessage>, RouteGenericInterface, unknown, FastifySchema, FastifyTypeProviderDefault, unknown>`	Outgoing response handler.

Returns `Promise<never>`

Returns a response wrapped in a promise to be handled by the Fastify router.

generateQuestion

- **generateQuestion**(`context`, `movement_die`, `turn_modifier`, `challenge_die`): `Promise<[WebsocketType, QuestionData | ConsequenceData]>`

Fetches a question, or consequence, and formats it appropriately.

Parameters

Name	Type	Description
`context`	`Context`	The context of the request sender.
`movement_die`	`number`	The original value of the movement die.
`turn_modifier`	`TurnModifier`	The turn modifier, if any exists.
`challenge_die`	`number`	The value of the challenge dice.

Returns `Promise<[WebsocketType, QuestionData | ConsequenceData]>`

The formatted question and request type.

getThemePacksAPI

- **getThemePacksAPI**(`req`, `res`): `Promise<never>`

Parameters

Name	Type
req	FastifyRequest<RouteGenericInterface, RawServerDefault, IncomingMessage, FastifySchema, FastifyTypeProviderDefault, unknown, FastifyBaseLogger, ResolveFastifyRequestType<FastifyTypeProviderDefault, FastifySchema, RouteGenericInterface»>
res	FastifyReply<RawServerDefault, IncomingMessage, ServerResponse<IncomingMessage>, RouteGenericInterface, unknown, FastifySchema, FastifyTypeProviderDefault, unknown>

Returns `Promise<never>`

movePlayer

- **movePlayer**(`gameID`, `movement_die`): `Promise<Document<unknown, any, UserType & { _id: ObjectId }>`

Takes the player whose current turn it is, and moves them forward.

Parameters

Name	Type	Description
`gameID`	`string`	The game ID of the player who must move.
`movement_die`	`number`	How far the player is moving forward.

Returns `Promise<Document<unknown, any, UserType & { _id: ObjectId }>`

The updated user data to ensure they moved.

nextPlayer

- **nextPlayer**(`context`): `Promise<Player[]>`

Given a game id, shift the player list left and return the next player in the turn order.

Parameters

Name	Type	Description
`context`	`Context`	The context of the user who sent the message, and the game it is connected to.

Returns `Promise<Player[]>`

The the player order, with the first in the list being the player who has first turn.

playerTurnOrder

- **playerTurnOrder**(`context`, `method`): `Promise<Player[]>`

Returns the player list in the turn order.

Name	Type	Description

Parameters

Name	Type	Description
`context`	`Context`	The context of the user and game the message are connected to.
`method`	`0 | 1 | 2`	Indicates function operation method. 0 for game start, 1 for next player, 2 for game rankings.

Returns `Promise<Player[]>`

A player list, sorted according to the method.

questionAnswer

- **questionAnswer**(`connections`, `data`, `context`): `Promise<boolean>`

Handle a user sending an answer request to the server

Parameters

Name	Type	Description
`connections`	`Connection`	The websocket information of all players connected to the specific game.
`data`	`WebsocketRequest`	Information related to the request, such as request id and the question answer.
`context`	`Context`	The context of the request sender.

Returns `Promise<boolean>`

Whether the answer submitted is, or is not, correct.

questionEnd

- **questionEnd**(`connections`, `data`, `context`, `early`, `question`): `Promise<boolean>`

The question has ended, either by timeout or by answer. Handle accordingly.

Parameters

Name	Type	Description
`connections`	`Connection`	The websocket information of all players connected to the specific game.
`data`	`WebsocketRequest`	Information related to the request, such as request id.
`context`	`Context`	The populated game instance to fetch information about the current game state.
`early`	`boolean`	Is this request ending the game before the timeout?
`question`	`boolean`	Are we ending a question or consequence?

Returns `Promise<boolean>`

This is a mutation function in which modifies the next game state and sends it to the players.

startGame

- **startGame**(`context`): `Promise<Player[]>`

Given a game id, prepare to start the game. To do so: 1. Randomize the player array to determine turn order. 2. Change the boolean in the game model to be True. 3. Return the username of the first player in the turn order.

Parameters

Name	Type	Description
`context`	`Context`	The context of the user who sent the message, and the game it is connected to.

Returns `Promise<Player[]>`

The the player order, with the first in the list being the player who has first turn.

turn

- **turn**(`connections`, `data`, `context`): `Promise<{ all_play: boolean ; answered: string[] ; movement_die: number ; timeout?: Timeout ; turn_modifier: TurnModifier ; turn_start: number }>`

Handle the turn logic for a single round of the game, triggered by the game node sending a message.

Parameters

Name	Type	Description
`connections`	`Connection`	List of all WebSockets relevant to the game that this turn is for.
`data`	`WebsocketRequest`	Any relevant data that the game node sends across the websocket stream.
`context`	`Context`	The context of the request sender.

Returns `Promise<{ all_play: boolean ; answered: string[] ; movement_die: number ; timeout?: Timeout ; turn_modifier: TurnModifier ; turn_start: number }>`

Module: controllers/QuizController

Functions

- formatConsequence
- formatQuestion
- getThemePacks
- validateAnswer

Functions

formatConsequence

- **formatConsequence**(`theme_pack_name`, `used_consequences`): `Promise<Consequence>`

Generate a consequence for the game.

Parameters

Name	Type	Description
`theme_pack_name`	`string`	The name of the theme pack file.
`used_consequences`	`number[]`	List of already used consequence ids.

Returns `Promise<Consequence>`

The consequence fetched for the game.

formatQuestion

- **formatQuestion**(theme_pack_name, category, question_type, used_questions): Promise<{ id: number ; media_type: null | "image" | "video" ; media_url: null | string ; options: string[] ; question: string }>

Fetches a random question from the given theme pack, formatted for display.

Parameters

Name	Type	Description
theme_pack_name	string	Name of the Theme Pack file in which a question is being g
category	string	The name of the category that the question must belong to.
question_type	"Multiple Choice" | "Text Question"	Denotes whether the question is multiple choice or text.
used_questions	number[]	A list of question ids in which have already been used by th

Returns Promise<{ id: number ; media_type: null | "image" | "video" ; media_url: null | string ; options: string[] ; question: string }>

Formatted question data, loaded from file.

getThemePacks

- **getThemePacks**(): Promise<string[]>

Gets a list of all available theme pack options for the game.

Returns Promise<string[]>

A list of all available theme packs.

validateAnswer

- **validateAnswer**(themePackName, questionID, questionCategory, userAnswer, questionType?): Promise<boolean>

Returns whether or not a user's answer to a question is correct.

Parameters

Name	Type	Description
themePackName	string	Name of the Theme Pack file in which a question needs to be validated against.
questionID	number	The specific question id within that question file.
questionCategory	string	The category in which the question can be found in.
userAnswer	string	The user answer to the question, in which needs to be validated.
questionType?	string	The specific type of question asked, if known.

Returns Promise<boolean>

Module: controllers/WebSocketController

Table of contents

Type Aliases

- ClientConn
- Connection
- Connections

Type Aliases

ClientConn ⊤ **ClientConn**: `Object`

Type declaration

Name	Type
conn	SocketStream
username	string

Connection ⊤ **Connection**: `Object`

Type declaration

Name	Type
clients	ClientConn[]
host	ClientConn
mutex	MutexInterface
turn?	{ all_play: boolean ; answered: string[] ; movement_die: number ; timeout?: NodeJS.Timeout
turn.all_play	boolean
turn.answered	string[]
turn.movement_die	number
turn.timeout?	NodeJS.Timeout
turn.turn_modifier	TurnModifier
turn.turn_start	number

Connections ⊤ **Connections**: `Record<string, Connection>`

Models

Module: models/Game

Table of contents

Type Aliases

- GameType

Variables

- default

Type Aliases

GameType ⊤ **GameType**: `Object`

The definition of what a game looks like within the database.

Type declaration

Name	Type
game_code	string
hostId	mongoose.Types.ObjectId
players	mongoose.Types.ObjectId[]
started	boolean
theme_pack	string
turn	number
used_consequences	number[]
used_questions	number[]

Variables

default

- Const **default**: Model<GameType, {}, {}, {}, any>

Module: models/User

Table of contents

Type Aliases

- UserType

Variables

- default

Type Aliases

UserType T **UserType**: Object

The definition of what a user looks like within the database.

Type declaration

Name	Type
color	string
game	mongoose.Types.ObjectId
position	number
token	string
userType	string
username	string

Variables

default

- Const **default**: Model<UserType, {}, {}, {}, any>

Routes

Module: routes/basic.router

Table of contents

- default

Functions

default

- **default**(`instance`, `opts`, `done`): `void`

A universal router meant for handling requests that are non-node specific.

Parameters

Name	Type	Description
`instance`	`FastifyInstance<RawServerDefault, IncomingMessage, ServerResponse<IncomingMessage>, FastifyBaseLogger, FastifyTypeProviderDefault>`	-
`opts`	`Record<never, never>`	Configuration options relevant to only this specific sub-router.
`done`	(`err?`: `Error`) => `void`	Function that indicates the end of definitions.

Returns `void`

Module: routes/client.router

Table of contents

Functions

- default

Functions

default

- **default**(`instance`, `opts`, `done`): `void`

The handling function for the client node router. It receives a request and various parameters, and handles it appropriately.

Parameters

Name	Type	Description
`instance`	`FastifyInstance<RawServerDefault, IncomingMessage, ServerResponse<IncomingMessage>, FastifyBaseLogger, FastifyTypeProviderDefault>`	-

Name	Type	Description
opts	Record<never, never>	Configuration options relevant to only this specific sub-router.
done	(err?: Error) => void	Function that indicates the end of definitions.

Returns void

Module: routes/game.router

Table of contents

Functions

- default

Functions

default

- **default**(instance, opts, done): void

The handling function for the game node router. It receives a request and various parameters, and handles it appropriately.

Parameters

Name	Type	Description
instance	FastifyInstance<RawServerDefault, IncomingMessage, ServerResponse<IncomingMessage>, FastifyBaseLogger, FastifyTypeProviderDefault>	-
opts	Record<never, never>	Configuration options relevant to only this specific sub-router.
done	(err?: Error) => void	Function that indicates the end of definitions.

Returns void

Module: routes/ws.router

Table of contents

Functions

- default
- checkUserAuthorization
- gameConsequenceRequest
- gameJoinRequest
- gameNextPlayerRequest
- gameQuestionAnswerRequest
- gameQuestionRequest

- gameSetupRequest
- gameStartRequest
- handleDisconnect
- handleMessage
- tryGetGame
- tryTurnAction
- sendError

Functions

default

- **default**(`instance`, `opts`, `done`): `void`

The handling function for the websocket router. It receives a request and various parameters, and handles it appropriately.

Parameters

Name	Type	Description
`instance`	`FastifyInstance<RawServerDefault, IncomingMessage, ServerResponse<IncomingMessage>, FastifyBaseLogger, FastifyTypeProviderDefault>`	-
`opts`	`Record<never, never>`	Configuration options relevant to only this specific sub-router.
`done`	`(err?: Error) => void`	Function that indicates the end of definitions.

Returns `void`

checkUserAuthorization

- **checkUserAuthorization**(`conn`, `data`, `context`, `goalUserType`): `void`

Given a user and a desired user type, checks if the user is authorized to perform the action.

Parameters

Name	Type	Description
`conn`	`SocketStream`	The websocket connection
`data`	`any`	The data sent from the client about the game
`context`	`Context`	The user context information
`goalUserType`	`string`	The desired user type to check against

Returns `boolean`

Whether or not the user is authorized

gameConsequenceRequest

- **gameConsequenceRequest**(`conn`, `data`, `context`): `void`

Handles when a client sends a Consequence Request.

Parameters

Name	Type	Description
`conn`	`SocketStream`	The websocket connection
`data`	`any`	The data sent from the client about the game
`context`	`Context`	The user context information

Returns `void`

gameJoinRequest

- **gameJoinRequest**(`conn`, `data`, `context`): `void`

Handles when a client sends a Game Join Request.

Parameters

Name	Type	Description
`conn`	`SocketStream`	The websocket connection
`data`	`any`	The data sent from the client about the game
`context`	`Context`	The user context information

Returns `void`

gameNextPlayerRequest

- **gameNextPlayerRequest**(`conn`, `data`, `context`): `void`

Handles when a client sends a NextPlayer Request.

Parameters

Name	Type	Description
`conn`	`SocketStream`	The websocket connection
`data`	`any`	The data sent from the client about the game
`context`	`Context`	The user context information

Returns `void`

gameQuestionAnswerRequest

- **gameQuestionAnswerRequest**(`conn`, `data`, `context`): `void`

Handles when a client sends a QuestionAnswer Request.

Parameters

Name	Type	Description
`conn`	`SocketStream`	The websocket connection
`data`	`any`	The data sent from the client about the game
`context`	`Context`	The user context information

Returns `void`

gameQuestionRequest

- **gameQuestionRequest**(`conn`, `data`, `context`): `void`

Handles when a client sends a Question Request.

Parameters

Name	Type	Description
`conn`	`SocketStream`	The websocket connection
`data`	`any`	The data sent from the client about the game
`context`	`Context`	The user context information

Returns `void`

gameSetupRequest

- **gameSetupRequest**(`conn`, `data`, `context`): `void`

Handles when a client sends a GameSetup Request.

Parameters

Name	Type	Description
`conn`	`SocketStream`	The websocket connection
`data`	`any`	The data sent from the client about the game
`context`	`Context`	The user context information

Returns `void`

handleDisconnect

- **handleDisconnect**(`conn`, `gameID`): `void`

Handles when a client disconnects from the server.

Parameters

Name	Type	Description
`conn`	`SocketStream`	The websocket connection
`gameID`	`string`	The game that the client belonged to

Returns `void`

handleMessage

- **handleMessage**(`conn`, `data`, `context`): `void`

Grabs the type of the incoming request and calls the appropriate function.

Parameters

Name	Type	Description
`conn`	`SocketStream`	The websocket connection
`data`	`any`	The data sent from the client about the game
`context`	`Context`	The user context information

Returns `void`

tryGetGame

- **tryGetGame**(`conn`, `data`, `context`): `Promise<Any>`

Attempts to get the game from the database.

Parameters

Name	Type	Description
`conn`	`SocketStream`	The websocket connection
`data`	`any`	The data sent from the client about the game
`context`	`Context`	The user context information

Returns `Promise<Any>`

The game if it exists, otherwise throws an error.

tryTurnAction

- **tryTurnAction**(`conn`, `data`, `context`, `actionName`, `action`): `boolean | void`

Attempts to execute a "Turn" action within the game and handles any errors that occur.

Parameters

Name	Type	Description
`conn`	`SocketStream`	The websocket connection
`data`	`any`	The data sent from the client about the game
`context`	`Context`	The user context information
`actionName`	`string`	Descriptive name of the action
`context`	() => `Promise<boolean | void>`	Function to execute the action

Returns `boolean | void` The result of the action was successful or not

sendError

- **sendError**(`conn`, `data`, `context`, `err`, `message`, `fatal`): `void`

When ever an error occurs, this function is called to send the error to the client.

Parameters

Name	Type	Description
conn	SocketStream	The websocket connection
data	any	The data sent from the client about the game
context	Context	The user context information
err	any	The error that occurred
message	string | null	A message to send to the client about the error
fatal	boolean	Whether or not the error is fatal to the backend running

Returns void

Shared Folder

Table of contents

Module: shared/apis

Type Aliases

ConnectionEstablished T **ConnectionEstablished**: Object

This data structure is to act as a confirmation that the user has connected to the websocket with the correct information.

Type declaration

Name	Type
JWT	string
gameCode	string
message	string
userType	string
username	string

ConsequenceData T **ConsequenceData**: { consequence_type: ConsequenceType ; id: number ; movement_die: number ; story: string } & TimedData

Detailing the consequence data that the server sends to the game and client nodes.

ErrorData T **ErrorData**: Object

If an error occurs, send back data of this format to ensure it can be handled.

Type declaration

Name	Type
error	string | Error
fatal	boolean

Name	Type
message?	string
token	string

GameEndAckData T **GameEndAckData**: Object

When a game has ended, the final rankings of the players is sent.

Type declaration

Name	Type
ranking	Player

GameJoinAckData T **GameJoinAckData**: Object

When a player connects to the websocket, send a player list to everyone to notify them.

Type declaration

Name	Type
players	Player[]

NextPlayerData T **NextPlayerData**: Object

When the game progresses to the next turn, the game node must know whose turn it is next.

Type declaration

Name	Type
player	Player

QuestionAnswerData T **QuestionAnswerData**: Object

When a client sends their answer to the server via a request, this is how that answer is formatted.

Type declaration

Name	Type
answer	string
category	string
id	number
question_type?	string

QuestionData T **QuestionData**: { all_play?: boolean ; category: string ; challenge_die: QuestionCategory ; id: number ; media_type?: "image" | "video" | null ; media_url?: string | null ; movement_die: number ; options: string[] ; question: string ; question_type: "Multiple Choice" } & TimedData

Detailing the question data that the server sends to the game and client nodes.

QuestionEndedData T **QuestionEndedData**: `Object`

Upon each question ending, send an array detailing where each player is located.

TimedData T **TimedData**: `Object`

Any response from the server that involves the server starting a timer will include the following information.

Type declaration

Name	Type
`timer_length`	`number`
`timer_start?`	`Date` | `number`

Module: shared/enums

Color

- `RED` - `"#FF0000"`
- `ORANGE` - `"#FFA500"`
- `YELLOW` - `"#FFFF00"`
- `GREEN` - `"#008000"`
- `BLUE` - `"#0000FF"`
- `PURPLE` - `"#800080"`
- `PINK` - `"#FFC0CB"`
- `BROWN` - `"#A52A2A"`

ConsequenceType

- `LoseATurn` - 2
- `MoveBackward` - 1
- `MoveForward` - 0
- `SkipATurn` - 3

QuestionCategory

- `TakeThreeAllPlay` - 0
- `MiscellaneousAllPlay` - 1
- `MusicalAllPlay` - 2
- `Consequence` - 3
- `TakeThreeMyPlay` - 4
- `MusicalMyPlay` - 5
- `MiscellaneousMyPlay` - 6
- `ConsequenceB` - 7

TurnModifier

- `Normal` - 0
- `DoubleFeature` - 1
- `FinalCut1` - 5
- `FinalCut2` - 4
- `FinalCut3` - 3
- `AllPlayToWin` - 2
- `Winner` - 6

WebsocketTypes

- `Consequence`
- `ConsequenceAck`
- `ConsequenceEnded`
- `ConsequenceEndedAck`
- `Error`
- `GameEnded`
- `GameEndedAck`
- `GameJoin`
- `GameJoinAck`
- `GameSetup`
- `GameSetupAck`
- `GameStart`
- `GameStartAck`
- `NextPlayer`
- `NextPlayerAck`
- `Ping`
- `PlayerDisconnectAck`
- `Pong`
- `QuestionAck`
- `QuestionAnswer`
- `QuestionEnded`
- `QuestionEndedAck`
- `QuestionRequest`
- `QuestionTimeOut`
- `QuestionTimerTick`
- `QuestionTimerTickAck`

Module: shared/types

Consequence T **Consequence**: `Object`

The format of a consequence card as data in the system.

Type declaration

Name	Type
`consequenceType`	`ConsequenceType`
`id`	`number`
`story`	`string`
`timerLength?`	`number`

WebsocketMessage T **WebsocketMessage**: `Object`

The generic interface of all messages sent across the websocket.

Type declaration

Name	Type
`data`	`any`
`requestId?`	`string`
`type`	`WebsocketType`

WebsocketRequest T **WebsocketRequest**: `WebsocketMessage` & { `token`: `string` ; `type`: `GameSetup` | `GameJoin` | `GameStart` | `GameEnded` | `QuestionRequest` | `QuestionTimerTick` | `QuestionEnded` | `QuestionAnswer` | `Consequence` | `ConsequenceEnded` | `NextPlayer` | `Ping` }

Extending WebsocketMessage, a websocket request limits the legal types and adds a token parameter.

WebsocketResponse T **WebsocketResponse**: `WebsocketMessage` & { `type`: `Error` | `GameSetupAck` | `GameJoinAck` | `GameStartAck` | `GameEndedAck` | `QuestionAck` | `QuestionTimerTickAck`| `QuestionEndedAck` | `ConsequenceAck` | `ConsequenceEndedAck` | `PlayerDisconnectAck` | `NextPlayerAck` | `Pong` }

Extending WebsocketMessage, a websocket response limits the legal types.

Player T **Player**: `Object`

The format of a player as data in the system. This is what the game and client would see, but may not be all relevant information about them.

Type declaration

Name	Type
color	string
position	number
username	string

Question T **Question**: `Object`

The format of a question as data in the system.

Type declaration

Name	Type
answer	string
clue_list	string[]
fake_answers	[]
id	number
media_type	"image" | "video" | null
media_url	string | null
question	string
question_type	"Multiple Choice" | "Text Question"

Module: shared/util

Class: MathUtil

Constructors

constructor

- **new default**()

Methods

choice

- `Static` **choice**<`T`>(`choices`, `amount?`): `T` | `T[]`

Return random entity from an array of choices.

Type parameters

Name
`T`

Parameters

Name	Type	Default value	Description
`choices`	`T[]`	`undefined`	Options to choose from.
`amount`	`number`	`1`	-

Returns `T` | `T[]`

The randomly selected option from the array.

randInt

- `Static` **randInt**(`a`, `b`): `number`

Generate a random integer between 2 integral bounds, inclusive.

Parameters

Name	Type	Description
`a`	`number`	Lower bound, rounded down if not integral.
`b`	`number`	Upper bound, rounded down if not integral.

Returns `number`

A random integer between a and b, inclusive.

shuffle

- `Static` **shuffle**<`T`>(`arr`): `T[]`

Type parameters

Name
`T`

Parameters

Name	Type
`arr`	`T[]`

Returns `T[]`