default_controller.py File Reference

Controller module for the Vorarlberg The Game backend server. More...

Functions

	openapi_server.controllers.default_controller.get_db_connection ()
	Establish a connection to the MySQL database.
float	**openapi_server.controllers.default_controller.haversine_distance** (float lat1, float lon1, float lat2, float lon2)
	Calculate the great circle distance between two points on the earth.
List[Dict]	**openapi_server.controllers.default_controller.bus_routes_get** ()
	Get all available bus routes from the database.
List[Dict]	**openapi_server.controllers.default_controller.bus_stops_get** ()
	Get all bus stops from the database.
List[Dict]	**openapi_server.controllers.default_controller.bus_stops_in_range_get** ()
	Get all bus stops within 400 meters of the seeker's current position.
Tuple[None, int]	**openapi_server.controllers.default_controller.cards_card_id_discard_post** (int card_id)
	Discard a card from the player's hand.
Tuple[None, int]	**openapi_server.controllers.default_controller.cards_card_id_use_post** (int card_id)
	Use a card from the player's hand.
List[Dict]	**openapi_server.controllers.default_controller.cards_get** ()
	Get all cards currently in the player's hand.
Tuple[None, int]	**openapi_server.controllers.default_controller.curses_curse_id_complete_post** (int curse_id)
	Mark a curse as completed and remove it from active curses.
List[Dict]	**openapi_server.controllers.default_controller.curses_get** ()
	Get all currently active curses.
Tuple[None, int]	**openapi_server.controllers.default_controller.exists_get** ()
	Check if the server is running and accessible.
Tuple[None, int]	**openapi_server.controllers.default_controller.game_end_post** ()
	End the current game and reset all game state.
Dict	**openapi_server.controllers.default_controller.game_info_get** ()
	Get current game information.
Tuple[None, int]	**openapi_server.controllers.default_controller.game_start_post** ()
	Start a new game.
Tuple[None, int]	**openapi_server.controllers.default_controller.game_swap_roles_post** ()

	Swap the roles of hider and seeker players.
Tuple[None, int]	**openapi_server.controllers.default_controller.hider_bus_stop_bus_stop_id_** (int bus_stop_id)
	Set the bus stop where the hider is hiding.
Dict	**openapi_server.controllers.default_controller.hider_get_coordinates_get** (
	Get the current coordinates of the hider.
Dict	**openapi_server.controllers.default_controller.hider_info_get** ()
	Get detailed information about the hider's current location.
Tuple[None, int]	**openapi_server.controllers.default_controller.hider_post_coordinates_post** (Dict body)
	Update the hider's coordinates.
Tuple[None, int]	**openapi_server.controllers.default_controller.questions_answer_post** ()
	Mark the current question as answered.
Tuple[None, int]	**openapi_server.controllers.default_controller.questions_ask_question_id_p** (int question_id)
	Ask a new question.
Dict	**openapi_server.controllers.default_controller.questions_current_get** ()
	Get information about the currently asked question.
List[Dict]	**openapi_server.controllers.default_controller.questions_get** ()
	Get all available questions.
List[Dict]	**openapi_server.controllers.default_controller.questions_previous_get** ()
	Get all previously asked questions.
Tuple[None, int]	**openapi_server.controllers.default_controller.questions_veto_post** ()
	Veto (cancel) the current question.
Dict	**openapi_server.controllers.default_controller.seeker_get_coordinates_get**
	Get the current coordinates of the seeker.
Dict	**openapi_server.controllers.default_controller.seeker_info_get** ()
	Get detailed information about the seeker's current location.
Tuple[None, int]	**openapi_server.controllers.default_controller.seeker_post_coordinates_pos** (Dict body)
	Update the seeker's coordinates and check for game end condition.

Variables

dict	**openapi_server.controllers.default_controller.DB_CONFIG**
	!
dict	**openapi_server.controllers.default_controller.GAME_STATE**
	!
dict	**openapi_server.controllers.default_controller.HIDER_INFO**
	!

dict **openapi_server.controllers.default_controller.SEEKER_INFO**

!

Detailed Description

Controller module for the Vorarlberg The Game backend server.

This module implements all the controller functions for the game's API endpoints. It handles game state management, player coordinates, questions, cards, and more.

Function Documentation

◆ bus_routes_get()

```
List[Dict] openapi_server.controllers.default_controller.bus_routes_get ( )
```

Get all available bus routes from the database.

Returns

List of dictionaries containing route information Each dictionary contains:

- routeId: The unique identifier of the route
- routeName: The name of the route

([{}], 500) on database error

◆ bus_stops_get()

```
List[Dict] openapi_server.controllers.default_controller.bus_stops_get ( )
```

Get all bus stops from the database.

Returns

List of dictionaries containing bus stop information Each dictionary contains:

- busStopId: The unique identifier of the bus stop
- busStopName: The name of the bus stop
- busStopLat: The latitude coordinate of the bus stop
- busStopLon: The longitude coordinate of the bus stop

([{}], 500) on database error

◆ bus_stops_in_range_get()

List[Dict] openapi_server.controllers.default_controller.bus_stops_in_range_get ( )

Get all bus stops within 400 meters of the seeker's current position.

Returns

List of dictionaries containing nearby bus stop information Each dictionary contains:

- busStopId: The unique identifier of the bus stop
- busStopName: The name of the bus stop
- busStopLat: The latitude coordinate of the bus stop
- busStopLon: The longitude coordinate of the bus stop

> **Note**
> Uses SEEKER_INFO global variable for current seeker position
> Maximum range is 400 meters

Returns

([{}], 500) on database error

◆ cards_card_id_discard_post()

Tuple[None, int]
openapi_server.controllers.default_controller.cards_card_id_discard_post (int card_id)

Discard a card from the player's hand.

Parameters

card_id The unique identifier of the card to discard

Returns

(None, 200) if successful

(None, 404) if card not found or not in hand

(None, 500) on database error

◆ cards_card_id_use_post()

```
Tuple[None, int]
openapi_server.controllers.default_controller.cards_card_id_use_post                ( int card_id )
```

Use a card from the player's hand.

Parameters

card_id The unique identifier of the card to use

Returns

(None, 200) if successful

(None, 404) if card not found or not in hand

(None, 500) on database error

> **Note**
> For curse cards, they are added to the curses table with current timestamp
> Other cards are marked as 'In Effect'

◆ cards_get()

```
List[Dict] openapi_server.controllers.default_controller.cards_get ( )
```

Get all cards currently in the player's hand.

Returns

List of dictionaries containing card information Each dictionary contains:

- cardId: The unique identifier of the card
- name: The name of the card
- description: The description of the card's effect
- type: The type of the card (e.g., 'curse')

([{}], 500) on database error

◆ curses_curse_id_complete_post()

```
Tuple[None, int]
openapi_server.controllers.default_controller.curses_curse_id_complete_post     ( int curse_id )
```

Mark a curse as completed and remove it from active curses.

Parameters

curse_id The unique identifier of the curse to complete

Returns

(None, 200) if successful

(None, 404) if curse not found

(None, 500) on database error

◆ curses_get()

```
List[Dict] openapi_server.controllers.default_controller.curses_get ( )
```

Get all currently active curses.

Returns

List of dictionaries containing curse information Each dictionary contains:

- curseId: The unique identifier of the curse
- name: The name of the curse
- description: The description of the curse's effect

([{}], 500) on database error

◆ exists_get()

```
Tuple[None, int] openapi_server.controllers.default_controller.exists_get ( )
```

Check if the server is running and accessible.

Returns

(None, 200) if server is running

◆ game_end_post()

```
Tuple[None, int] openapi_server.controllers.default_controller.game_end_post ( )
```

End the current game and reset all game state.

Returns

(None, 200) if successful

(None, 500) on database error

> **Note**
> Resets all temporary tables (cards, curses, questions)
> Resets global game state variables

◆ game_info_get()

```
Dict openapi_server.controllers.default_controller.game_info_get ( )
```

Get current game information.

Returns

Dictionary containing game information:

- currentGameTimer: Time elapsed since game start in seconds
- hiderName: Name of the hider player
- seekerName: Name of the seeker player

◆ game_start_post()

```
Tuple[None, int] openapi_server.controllers.default_controller.game_start_post ( )
```

Start a new game.

Returns

(None, 200) if successful

(None, 500) on database error

> **Note**
> Initializes card deck with all cards set to 'In Deck'
> Deals initial hand of 3 cards
> Initializes questions table with all questions set to 'Not Asked'
> Clears any existing curses

◆ game_swap_roles_post()

```
Tuple[None, int] openapi_server.controllers.default_controller.game_swap_roles_post ( )
```

Swap the roles of hider and seeker players.

Returns

(None, 200) if successful

> **Note**
> Swaps player names in GAME_STATE
> Swaps player coordinates between HIDER_INFO and SEEKER_INFO
> Resets hider's bus_stop_id to None

◆ get_db_connection()

```
openapi_server.controllers.default_controller.get_db_connection ( )
```

Establish a connection to the MySQL database.

Returns

mysql.connector.connection.MySQLConnection object if successful, None if failed

◆ haversine_distance()

```
float openapi_server.controllers.default_controller.haversine_distance ( float lat1,
                                                                         float lon1,
                                                                         float lat2,
                                                                         float lon2 )
```

Calculate the great circle distance between two points on the earth.

Parameters

lat1 Latitude of the first point in decimal degrees

lon1 Longitude of the first point in decimal degrees

lat2 Latitude of the second point in decimal degrees

lon2 Longitude of the second point in decimal degrees

Returns

Distance in meters between the two points

◆ hider_bus_stop_bus_stop_id_post()

```
Tuple[None, int]
openapi_server.controllers.default_controller.hider_bus_stop_bus_stop_id_post ( int bus_stop_id )
```

Set the bus stop where the hider is hiding.

Parameters

bus_stop_id The unique identifier of the chosen bus stop

Returns

(None, 200) if successful

(None, 404) if bus stop not found

(None, 500) on database error

> **Note**
> Updates hider's coordinates to match the bus stop location

◆ hider_get_coordinates_get()

```
Dict openapi_server.controllers.default_controller.hider_get_coordinates_get ( )
```

Get the current coordinates of the hider.

Returns

Dictionary containing hider's coordinates:

- lat: Latitude coordinate
- lon: Longitude coordinate

◆ hider_info_get()

```
Dict openapi_server.controllers.default_controller.hider_info_get ( )
```

Get detailed information about the hider's current location.

Returns

Dictionary containing hider information:

- lat: Latitude coordinate
- lon: Longitude coordinate
- busStationName: Name of the bus station (if at one)
- busStationLat: Bus station latitude
- busStationLon: Bus station longitude
- busStationAltitude: Altitude of the bus station
- busStationBezirk: District where the bus station is located
- busStationGemeinde: Municipality where the bus station is located
- distanceToNearestTrainStation: Distance to nearest train station
- distanceToNearestMountain: Distance to nearest mountain
- distanceToNearestMinigolf: Distance to nearest minigolf
- distanceToNearestMuseum: Distance to nearest museum

(Dict, 500) on database error with basic location info only

◆ hider_post_coordinates_post()

```
Tuple[None, int]
openapi_server.controllers.default_controller.hider_post_coordinates_post          ( Dict body )
```

Update the hider's coordinates.

Parameters

body Dictionary containing new coordinates:

- lat: New latitude coordinate
- lon: New longitude coordinate

Returns

(None, 200) if successful

(None, 400) if request body is invalid

◆ questions_answer_post()

```
Tuple[None, int] openapi_server.controllers.default_controller.questions_answer_post ( )
```

Mark the current question as answered.

Returns

(None, 200) if successful

(None, 404) if no current question exists

(None, 500) on database error

> **Note**
> Updates question status to 'Previously Asked' with answer='Answered'

◆ questions_ask_question_id_post()

```
Tuple[None, int]
openapi_server.controllers.default_controller.questions_ask_question_id_post ( int question_id )
```

Ask a new question.

Parameters

question_id The unique identifier of the question to ask

Returns

(None, 200) if successful

(None, 404) if question not found or already asked

(None, 500) on database error

> **Note**
> Marks any current question as 'Previously Asked'
> Sets the new question as 'Currently Asked' with current timestamp

◆ questions_current_get()

```
Dict openapi_server.controllers.default_controller.questions_current_get ( )
```

Get information about the currently asked question.

Returns

Dictionary containing current question information:

- questionId: Unique identifier of the question
- name: Name of the question
- description: Description or text of the question
- timeLeft: Time remaining to answer in seconds

({}, 404) if no current question

({}, 500) on database error

◆ questions_get()

```
List[Dict] openapi_server.controllers.default_controller.questions_get ( )
```

Get all available questions.

Returns

List of dictionaries containing question information:

- id: Unique identifier of the question
- name: Name of the question
- description: Description or text of the question
- alreadyAsked: Boolean indicating if question was already asked

([{}], 500) on database error

◆ questions_previous_get()

```
List[Dict] openapi_server.controllers.default_controller.questions_previous_get ( )
```

Get all previously asked questions.

Returns

List of dictionaries containing previous question information:

- questionId: Unique identifier of the question
- name: Name of the question
- description: Description or text of the question
- answer: The answer given or "No answer" if unanswered

([{}], 500) on database error

◆ questions_veto_post()

```
Tuple[None, int] openapi_server.controllers.default_controller.questions_veto_post ( )
```

Veto (cancel) the current question.

Returns

(None, 200) if successful

(None, 404) if no current question exists

(None, 500) on database error

> **Note**
> Resets question status to 'Not Asked' and clears timestamp

◆ seeker_get_coordinates_get()

```
Dict openapi_server.controllers.default_controller.seeker_get_coordinates_get ( )
```

Get the current coordinates of the seeker.

Returns

Dictionary containing seeker's coordinates:

- lat: Latitude coordinate
- lon: Longitude coordinate

◆ seeker_info_get()

```
Dict openapi_server.controllers.default_controller.seeker_info_get ( )
```

Get detailed information about the seeker's current location.

Returns

Dictionary containing seeker information:

- lat: Latitude coordinate
- lon: Longitude coordinate
- altitude: Current altitude
- bezirk: Current district
- gemeinde: Current municipality
- distanceToNearestTrainStation: Distance to nearest train station
- distanceToNearestMountain: Distance to nearest mountain
- distanceToNearestMinigolf: Distance to nearest minigolf
- distanceToNearestMuseum: Distance to nearest museum

Basic info dict with default values on database error

◆ seeker_post_coordinates_post()

```
Tuple[None, int]
openapi_server.controllers.default_controller.seeker_post_coordinates_post ( Dict body )
```

Update the seeker's coordinates and check for game end condition.

Parameters

body Dictionary containing new coordinates:

- lat: New latitude coordinate
- lon: New longitude coordinate

Returns

(None, 200) if successful

(None, 400) if request body is invalid

Note

Checks if seeker is within 20 meters of hider for game end condition

Variable Documentation

◆ DB_CONFIG

dict openapi_server.controllers.default_controller.DB_CONFIG

Initial value:

```
= {
    'user': 'vtg_server',
    'password': 'vtg_server',
    'host': 'localhost',
    'database': 'vtg_data'
}
```

!

Database configuration settings

◆ GAME_STATE

dict openapi_server.controllers.default_controller.GAME_STATE

Initial value:

```
= {
    'is_started': False,
    'start_time': None,
    'hider_name': 'Hider',
    'seeker_name': 'Seeker'
}
```

!

Global game state variables

◆ HIDER_INFO

dict openapi_server.controllers.default_controller.HIDER_INFO

Initial value:

```
= {
    'lat': 47.4167,
    'bus_stop_id': None
}
```

!

Global hider information

◆ SEEKER_INFO

dict openapi_server.controllers.default_controller.SEEKER_INFO

Initial value:

```
= {
    'lat': 47.4167,
}
```

!

Global seeker information