🌍

# Documentation - Country Guesser API

## ▼ Client ←→ API

### ▼ Authentification

#### ▼ Login

**URI :** `/login`

**METHOD :** `POST`

**FORMAT :** `JSON`

##### ▼ PARAMETERS

- `nickname_email : string`
- `password : string`

##### ▼ PARAMETERS EXAMPLE

```
{
  "nickname_email":"test",
  "password":"testtest"
}
```

##### ▼ RETURN : `JSON`

```
{
  "player_id":1,
  "nickname":"test",
  "email":"test@test.com"
  "credential":"$2y$12$mcekdxBGYvxEP8hLkRy7T.RyPMYagQmprv9FXwrNVpaV2hSjlgcQS"
}
```

##### ▼ CODE EXAMPLE

```
var data = {
    "nickname_email": "test", // or "test@test.com"
    "password": "testtest"
};


fetch("https://api.countryguesser.deletesystem32.fr/login", {
    method: "POST",
    body: JSON.stringify(data),
    headers: {
        "Content-Type": "application/json"
    }
}).then(function(res) {
    console.log(res.json());
});
```

#### ▼ Register

📌 Register request will add a new user and log him

**URI :** `/register`

**METHOD :** `POST`

**FORMAT :** `JSON`

▼ **PARAMETERS**

- `nickname : string`
- `email : string`
- `password : string`
- `password_confirmation : string`

▼ **PARAMETERS EXAMPLE**

```
{
  "nickname":"test",
  "email":"test@test.com",
  "password":"testtest",
  "password_confirmation":"testtest"
}
```

▼ **RETURN :** `JSON`

```
{
  "player_id":1,
  "nickname":"test",
  "email":"test@test.com"
  "credential":"$2y$12$mcekdxBGYvxEP8hLkRy7T.RyPMYagQmprv9FXwrNVpaV2hSjlgcQS"
}
```

▼ **CODE EXAMPLE**

```
var user_to_create = {
    "nickname": "testFromJS",
    "email": "testFromJS@test.com",
    "password": "testFromJSFile",
    "password_confirmation": "testFromJSFile"
};

fetch("https://api.countryguesser.deletesystem32.fr/register", {
    method: 'POST',
    body: JSON.stringify(user_to_create),
    headers: {
        "Content-Type": "application/json"
    }
}).then(function(response) {
    console.log(response.json());
});
```

## ▼ Leaderboard

### ▼ Get leaderboard

📌 The server response is ordered by descending, to get best players first

**URI :** `/player/getleaderboard`

**METHOD :** `GET`

**FORMAT :** `JSON`

**RETURN :** `JSON`

```
[
  {
    "id": 3,
    "player_id": 36,
    "nickname": "test",
    "games_won": 3,
    "games_played": 6,
    "created_at": "2022-12-26 12:12:08"
  },
  {
    "id": 2,
    "player_id": 28,
    "nickname": "test3",
    "games_won": 2,
    "games_played": 2,
    "created_at": "2022-12-26 12:03:00"
  },
  {
    "id": 4,
    "player_id": 35,
    "nickname": "testLead",
    "games_won": 1,
    "games_played": 5,
    "created_at": "2022-12-26 12:12:11"
  },
  {
    "id": 6,
    "player_id": 30,
    "nickname": "testLead2",
    "games_won": 1,
    "games_played": 1,
    "created_at": "2022-12-26 13:24:17"
  },
  {
    "id": 5,
    "player_id": 37,
    "nickname": "test2",
    "games_won": 0,
    "games_played": 1,
    "created_at": "2022-12-26 13:24:14"
  }
]
```

### ▼ Get player leaderboard stats

**URI :** `/player/getleaderboardstats`

**METHOD :** `POST`

**FORMAT :** `JSON`

**PARAMETERS :**

- `player_id : int`

**RETURN :** `JSON`

```
{
  "games_won": 3,
  "games_played": 6
}
```

## ▼ Discussion Client ← →WebSocket Server

## ▼ Connect the client to the WS Server

When the client connects to the WS Server, it must provide these data :

- player credential
- room size (number of players in a game)
- max rounds

> ⚠️ Max rounds % (modulo) room size must be equal to 1
>
> When the client connects to the WS Server, it must send `roundData` to the WS Server (cf. DATA TYPE : roundData) and at each new round the client must send roundData to the server

**CODE EXEMPLE :**

```
ws = new WebSocket('ws://ws.countryguesser.deletesystem32.fr?playerCredential=' + playerDataLoaded.credential + '&roomSize=' + parse
// Send a message a random country to guess for the game to the server
ws.onopen = () => {
    ws.send(JSON.stringify({
        type: "roundData",
        name: "France",
        code: "Fr",
        flag: "[FLAG_URL_API]",
        latLng: "[LAT_LGN_COUNTRY_API]",
    }));
}
```

## ▼ Client → WS Server

### ▼ DATA TYPE : `roundData`

**FORMAT :** `JSON`

**DETAILS :** The roundData sent to the WebSocket Server by the client will be the country to guess for the current round

#### ▼ PARAMETERS :

- `type : string`
- `countryToGuess : string`

#### ▼ PARAMETERS EXAMPLE :

```
{
  "type": "roundData",
  "name": "France",
  "code": "Fr",
  "flag": "[FLAG_URL_API]",
  "latLng": "[LAT_LGN_COUNTRY_API]"
}
```

### ▼ DATA TYPE : `playerResponse`

**FORMAT :** `JSON`

**DETAILS** : The playerResponse sent to the WebSocket Server is the player response for the current round

#### ▼ PARAMETERS :

- `type : string`
- `playerResponse : string`

#### ▼ PARAMETERS EXAMPLE :

```
{
  "type": "playerResponse",
  "playerResponse": "Au"
}
```

▼ **DATA TYPE :** `cancelMultiplayerGame`

**FORMAT :** `JSON`

**DETAILS** : The cancelMultiplayerGame data sent to the WebSocket Server will inform the server that the client want to quit the current game or queue (if the player is in the queue, looking for a game)

▼ **PARAMETERS :**

- `type : string`

▼ **PARAMETERS EXAMPLE :**

```
{
  "type": "cancelMultiplayerGame"
}
```

## ▼ WS Server → Client

▼ **DATA TYPE :** `information`

**FORMAT :** `JSON`

**DETAILS :** The information data sent by the WS server to the client will be all the information about what happened that will be useful for the client to play a game correctly

▼ **PARAMETERS :**

- `type : string`
- `informationType : string`
- `message : string`

▼ **INFORMATION TYPE :** `inQueue`

**DETAILS :** Sent when the WS server put the client in the queue

**PARAMETERS EXEMPLE :**

```
{
  "type": "information",
  "informationType": "inQueue",
  "message": "Waiting for a room to join"
}
```

▼ **INFORMATION TYPE :** `roomCreated`

**DETAILS :** Sent to the client when the server has found at least one other player  looking for a room with the same size, so the WS server create a room and put them into

**PARAMETERS EXEMPLE :**

```
{
  "type": "information",
  "informationType": "roomCreated",
  "message": "Room created"
}
```

▼ **INFORMATION TYPE :** `roomFound`

**DETAILS :** Sent to the client when the WS server has found a room for him

**PARAMETERS EXEMPLE :**

```
{
  "type": "information",
  "informationType": "roomFound",
  "message": "Room found"
}
```

▼ **INFORMATION TYPE :** `waitingPlayers`

**DETAILS :** Sent to the client when he is in a room but the room isn't full

**PARAMETERS EXEMPLE :**

```
{
  "type": "information",
  "informationType": "waitingPlayers",
  "message": "Waiting for other players"
}
```

▼ **INFORMATION TYPE :** `roomFull`

**DETAILS :** Sent to the client when the room is full

**PARAMETERS EXEMPLE :**

```
{
  "type": "information",
  "informationType": "roomFull",
  "message": "Room full"
}
```

▼ **INFORMATION TYPE :** `gameCreated`

**DETAILS :** Sent to the client when the room is full, the WS server created the game

**PARAMETERS EXEMPLE :**

```
{
  "type": "information",
  "informationType": "gameCreated",
  "message": "Game created"
}
```

▼ **INFORMATION TYPE :** `roundCreated`

**DETAILS :** Sent to the client when a round is created

**PARAMETERS EXEMPLE :**

```
{
  "type": "information",
  "informationType": "roundCreated",
  "message": "Round created"
}
```

▼ **INFORMATION TYPE :** `roundOver`

**DETAILS :** Sent to the client when a round is over

**PARAMETERS EXEMPLE :**

```
{
  "type": "information",
  "informationType": "roundOver",
  "message": "Round over"
}
```

▼ **INFORMATION TYPE :** `gameOver`

**DETAILS :** Sent to the client when the game is over

**PARAMETERS EXEMPLE :**

```
{
  "type": "information",
  "informationType": "gameOver",
  "message": "Game over"
}
```

▼ **INFORMATION TYPE :** `removedFromQueue`

> 📌 When this information type is sent to a client who leaves a room, an `error` data is sent to the rest of players in the room, players are disconnected and the room is deleted

**DETAILS :** Sent to the player when he has been removed from the queue, when he quit the queue

**PARAMETERS EXEMPLE :**

```
{
  "type": "information",
  "informationType": "removedFromQueue",
  "message": "You have been removed from the queue"
}
```

▼ **INFORMATION TYPE :** `removedFromRoom`

> 📌 When this information type is sent to a client who leaves a room, an `error` data is sent to the rest of players in the room, players are disconnected and the room is deleted

**DETAILS :** Sent to the player when he has been removed from the room, when he quit the game

**PARAMETERS EXEMPLE :**

```
{
  "type": "information",
  "informationType": "removedFromQueue",
  "message": "You have been removed from the room"
}
```

▼ **INFORMATION TYPE :** `wrongAnswer`

**DETAILS :** Sent to the client when he send a wrong answer for the country to guess for the current round

**PARAMETERS EXEMPLE :**

```
{
  "type": "information",
  "informationType": "wrongAnswer",
  "message": "Wrong answer"
}
```

▼ **DATA TYPE :** `error`

**FORMAT :** `JSON`

**DETAILS :** Error data will be sent to the client when something that the WS server didn't expect happened

▼ **PARAMETERS :**

- `type : string`
- `errorType : string`
- `message : string`

▼ **ERROR TYPE :** `aPlayerLeft`

**DETAILS :** Sent to the players in a room when a player has left the room, players are disconnected and the room is deleted

**PARAMETERS EXEMPLE :**

```
{
  "type": "error",
  "errorType": "aPlayerLeft",
  "message": "A player has left the room"
}
```

▼ **DATA TYPE :** `data`

**FORMAT :** `JSON`

**DETAILS :** This data type is sent to give the client data about the game

▼ **PARAMETERS :**

- `type : string`
- `roomId : string`
- `gameId : int`
- `roomSize : int`

▼ **PARAMETERS EXEMPLE :**

```
{
  "type": "data",
  "roomId": "63ab134e2a008"
  "gameId": 132
  "roomSize": 3
}
```

---

## ▼ WebSocket Server ←→ API

### ▼ Create a game

**URI :** `/game/create`

**METHOD :** `GET`

**FORMAT :** `JSON`

**PARAMETERS :**

- `game_id : int`

**RETURN :** `JSON`

```
{
  "game_id": 43
}
```

## ▼ Create a round

**URI :** `/game/round/create`

**METHOD :** `POST`

**FORMAT :** `JSON`

**PARAMETERS :**

- `game_id : int`
- `round_id : int`
- `response : string`

## ▼ Insert round data

**URI :** `/game/round/playeranswer`

**METHOD :** `POST`

**FORMAT :** `JSON`

**PARAMETERS :**

- `game_id : int`
- `round_id : int`
- `player_id : int`
- `player_answer : string`

## ▼ Update a game

**URI :** `/game/update`

**METHOD :** `POST`

**FORMAT :** `JSON`

**PARAMETERS :**

- `game_id : int`

## ▼ Get player data with credential key

**URI :** `/player/playerdata`

**METHOD :** `POST`

**FORMAT :** `JSON`

**PARAMETERS :**

- `credential_key : string`

**RETURN :** `JSON`

```
{
  "player_id": 36,
  "nickname": "testLead2",
  "email": "testLead2@gmail.com",
  "credential": "$2y$12$n7uuLBaKWqRY4/jfTGyIdOJcxsOqg.0nfLTwwoOdvUs5liQ5PL8eC"
}
```

## ▼ Get game data

**URI :** `/game/getgamedata`

**METHOD :** `POST`

**FORMAT :** `JSON`

**PARAMETERS :**

- `game_id : int`

**RETURN :** `JSON`

```
{
  "created_at": "2022-12-28 06:48:24"
  "game_id": 129
  "id": 209
  "nb_rounds": 3
  "player_id": 36
  "player_response": "Ir"
  "response": "Ir"
  "round_id": 3
  "winner_id": 35
}
```

## ▼ Check round

**URI :** `/game/round/check`

**METHOD :** `POST`

**FORMAT :** `JSON`

**PARAMETERS :**

- `game_id : int`
- `round_id : int`

**RETURN :** `JSON` (bool)

```
{
  true
}
```

## ▼ Update leaderboard

**URI :** `/player/updateleaderboard`

**METHOD :** `POST`

**FORMAT :** `JSON`

**PARAMETERS :**

- `player_id : int`

## ▼ Delete game

**URI :** `/game/delete`

**METHOD :** `POST`

**FORMAT :** `JSON`

**PARAMETERS :**

- `game_id : int`

## ▼ Add game participant

**URI :** `/game/participants`

**METHOD :** `POST`

**FORMAT :** `JSON`

**PARAMETERS :**

- `game_id : int`
- `player_id : int`