



# Projet de Développement Web - Rapport

## Équipe

Jordan Baumard

Charles Hurst

Pierre Leocadie

**Groupe : 209**



Lien du jeu hébergé : <https://playcountryguesser.netlify.app/>

## Sommaire

1. Explication de notre projet
2. Cahier des charges
3. Technologies utilisées
4. Partie Back-end
  - a. La base de données
  - b. L'API de Country Guesser
  - c. Le serveur websocket
  - d. Faire fonctionner l'API de Country Guesser et le serveur websocket en local
  - e. Déploiement sur un serveur dans le cloud avec le service cloud Linode

5. Partie Front-End
6. Installer la partie front-end en local
7. Les points d'améliorations à prévoir

## En quoi consiste notre projet ?

Notre projet est un jeu que nous avons décidé d'appeler **Country Guesser** et directement inspiré d'un autre jeu déjà existant se dénommant **GeoGuessr**.

Country Guesser se joue de la manière suivante :

On donne au joueur le drapeau d'un pays sélectionné de manière aléatoire par le jeu et le joueur doit retrouver le pays sur la carte du monde le plus rapidement possible.

Si le joueur ne parvient pas à trouver, il a droit à 3 indices qui lui indiqueront sur quelle partie de du globe terrestre se trouve le pays.

Country Guesser se joue seul ou à plusieurs. Le principe reste le même pour une partie multi-joueur, le but est de trouver le pays correspondant au drapeau donné le plus rapidement possible avant le joueur adverse.

Une partie multi-joueur se compose de plusieurs manches. Pour remporter une manche, le joueur doit retrouver en premier le pays correspondant au drapeau choisi aléatoirement.

Le joueur ayant remporté le plus de manches gagne la partie.

## Cahier des charges

Le joueur doit pouvoir :

- S'inscrire
  - Pseudo
  - Email
  - Mot de passe
- Se connecter
  - Pseudo ou email

- Mot de passe
- Se déconnecter
- Consulter les statistiques de ses parties multi-joueurs
  - Nombre de parties jouées
  - Nombre de parties gagnées
- Consulter le classement des meilleurs joueurs du jeu
- Jouer une partie en solo
  - Pour jouer une partie en solo, le joueur doit avoir un compte et être connecté
- Jouer une partie en multi-joueurs
  - Pour jouer une partie en multi-joueurs, le joueur doit avoir un compte et être connecté
- Créer une partie multi-joueur personnalisée

## Technologies utilisées

- Git pour le versioning du projet
- GitHub pour collaborer sur le code en équipe
- Discord pour faire des points sur l'avancée du projet
- Notion pour s'organiser et organiser le projet

## Back-end

- PHP
  - Le gestionnaire de dépendance Composer
  - Package PHP Workerman (pour le multi-joueur → Serveur Websocket)
- Mysql pour la base de données
- PHPMyAdmin pour gérer la base de données
- Le service cloud Linode pour l'hébergement
- Docker pour déployer sur le serveur dans le cloud :

- Mysql,
- PHPMyAdmin,
- L'API de Country Guesser
- Le serveur Websocket de Country Guesser pour le multi-joueur
- Portainer.io pour la gestion et le monitoring des containers Docker
- Reverse proxy Ngnix pour la redirection de chaque sous-domaine vers le service correspondant

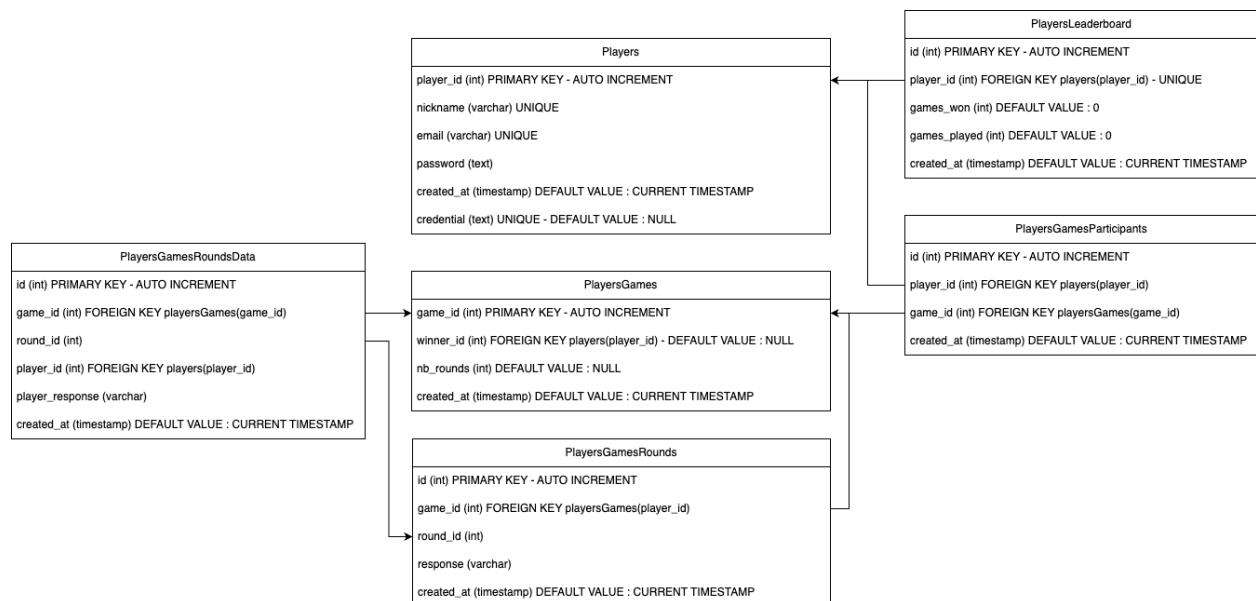
## Front-end

- ReactJS
- Material UI
- Mapbox pour l'affichage du globe terrestre

## Back-end

### La base de données

Voici comment nous avons décidé de structurer notre base de données.



Lorsqu'un joueur s'inscrit, nous stockons les données du joueur dans **la table** `players` . Le pseudonyme (**le champs** `nickname`) du joueur doit être unique ainsi que son `email` et peuvent être d'une **longueur maximale de 255 caractères**. Le mot de passe `password` de l'utilisateur est haché avec **la fonction de hachage** `bcrypt` en PHP qui offre l'avantage d'avoir plusieurs hash possible pour un même mot et donc rend les attaques plus difficiles.

Par exemple :

Le mot "testtest" **plusieurs hash possibles** avec `bcrypt` :

- \$2y\$12\$IsC9/0Kjgp183HJHDhFQXO1kAzMpKFqUeIR5GC7emLCMIRUadWNW6
- \$2y\$12\$mGlrZuj36t7mL2EtLfuAIOFWtAf8k/MiKW4nEna/SDfgta.j0P2ie
- \$2y\$12\$nRipYb6VU5WNvIRlj4jwuO6qJq.7vYeJyxYFwyT/HjsNMTqDp3nCq
- ....

Le mot "testtest" **un unique hash possible** avec `SHA-256` :

- 37268335dd6931045bdcdf92623ff819a64244b53d0e746d438797349d4da578

Lors de l'inscription nous enregistrons aussi le moment où l'inscription a été faite (date et heure) (**le champs** `created_at`). Une "credential key" (**le champs** `credential`) est également générée pour le joueur, le `credential` consiste dans le **hash de la concaténation** suivante avec la **fonction de hachage** `bcrypt` :

`player_id + nickname + email + password + created_at`

Cela permet d'éviter qu'un joueur usurpe l'identité d'un autre joueur, en changeant tout simplement son `player_id` et/ou son `nickname` et/ou son `email` par des informations qui ne sont pas les siennes. Le `credential` vient en complément de l'identifiant par défaut du joueur qui est le `player_id` qui correspond également à la clé primaire de la table.

Ainsi, lors de la connexion, seules les informations suivantes sont renvoyées au joueur :

- `player_id`
- `nickname`
- `email`
- `credential`

Lorsqu'une partie est créée, nous enregistrons les joueurs qui participent à la partie dans **la table** `playersGamesParticipants` avec l'identifiant du joueur (`player_id`) et l'identifiant de la partie (`game_id`) à laquelle il a participé.

Une partie multi-joueur se compose de plusieurs manches. Lors d'une nouvelle manche, nous la créons dans **la table** `playersGamesRounds`. Nous stockons l'identifiant de la partie (`game_id`), l'identifiant de la manche (`round_id`) ainsi que la bonne réponse de la manche qui correspond au code du pays à deviner (`response`). Nous enregistrons également le moment où la manche a été créée (`created_at`).

Lorsque le joueur envoie sa réponse, nous la gardons dans **la table** `playersGamesRoundsData` avec l'identifiant de la partie (`game_id`) dans laquelle le joueur se trouve, l'identifiant de la manche (`round_id`) pour laquelle il a envoyé sa réponse, son identifiant (`player_id`), sa réponse (`player_response`) ainsi que le moment où il a envoyé sa réponse (`created_at`).

**La table** `playersLeaderboard` contient les statistiques de chaque joueurs en partie multi-joueur, l'identifiant du joueur (`player_id`), son nombre de parties jouées (`games_played`) et son nombre de partie gagnées (`games_won`). Cette table est mise à jour à la fin de chaque partie multi-joueur.

## L'API de Country Guesser

Puisque nous avons décidé de faire la partie front-end de Country Guesser avec React et de faire un jeu multi-joueur, afin de simplifier les interactions entre la base de données et le front-end ainsi que les interactions entre le serveur websocket et la base de données. Nous avons décidé de faire une API (cf.  [Documentation - Country Guesser API](#)).

Ainsi, l'API évite à la partie front-end (client) et le serveur websocket d'intéragir directement avec la base de données. De plus, l'API nous donne plus de liberté du côté client si nous souhaitons par exemple implémenter un autre type d'interface pour Country Guesser.

Nous nous sommes appuyés sur le design pattern MVC (la seule exception ici, c'est que nous n'avons pas de réelles vues puisqu'il s'agit d'une API) avec **de la POO** afin de construire l'API et la rendre le plus maintenable possible.

```
src
└── Controllers
    ├── Auth
    │   └── LoginController.php
    │   └── RegisterController.php
    ├── Game
    │   └── CheckRoundController.php
    │   └── CreateGameController.php
    │   └── CreateRoundController.php
    │   └── DeleteGameController.php
    │   └── GameParticipantsController.php
    │   └── GetGameDataController.php
    │   └── InsertRoundDataController.php
    │   └── UpdateGameController.php
    ├── Player
    │   └── GetLeaderboard.php
    │   └── GetLeaderboardStats.php
    │   └── PlayerDataController.php
    │   └── UpdateLeaderboard.php
    └── Lib
        └── CheckIfDataExist.php
        └── CheckIfDataValid.php
        └── RequestDataCompliance.php
        └── SecureData.php
└── Models
    > Auth
    └── Database
        └── DatabaseConnection.php
    └── Game
        └── CheckRound.php
        └── CreateGame.php
        └── CreateRound.php
        └── DeleteGame.php
        └── GameParticipants.php
        └── GetGameData.php
        └── InsertRoundData.php
        └── UpdateGame.php
    └── Player
        └── Leaderboard.php
        └── PlayerData.php
    └── Router.php
└── index.php
```

Structure de l'API de Country Guesser avec le design pattern MVC

Le dossier `Lib` est l'équivalent d'un dossier `Utils`

Lors d'une requête à l'API, la page `index.php` qui est utilisé comme routeur, va se charger d'aller chercher le contrôleur correspondant à la requête faite et le contrôleur va aller chercher le modèle qui lui est associé. La réponse générée par le modèle est remonté à l'utilisateur sous forme de JSON. Dans le cas où la requête ne correspond à aucune des routes qui ont été définies ou que la requête ne spécifie aucune route, une redirection est faite vers la documentation de l'API  [Documentation - Country Guesser API](#)

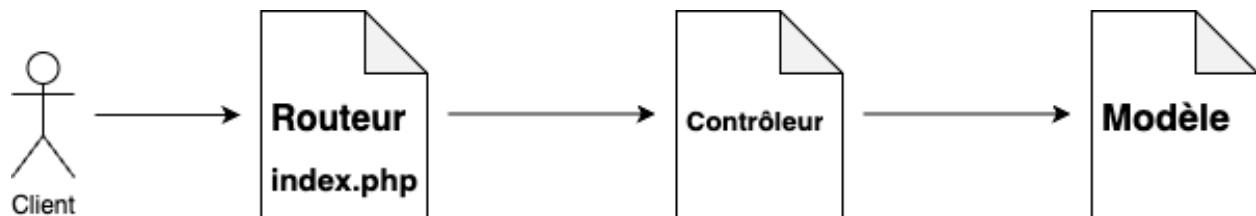
Par exemple :

On fait la requête suivante à l'API afin d'obtenir le classement des joueurs :

Requête : <https://api.countryguesser.deletesystem32.fr/player/getleaderboard>

Requête → Route : `/player/getleaderboard` → `index.php` → Contrôleur :

`Player/GetLeaderboard.php` → Modèle : `Player/Leaderboard.php`



```
index.php X
src > index.php
1  <?php
2
3  require_once __DIR__ . "/../vendor/autoload.php";
4
5  use CountryGuesser\Models\Router;
6
7  // To avoid CORS errors
8  header("Access-Control-Allow-Origin: *");
9  header("Access-Control-Allow-Methods: *");
10 header("Access-Control-Allow-Headers: *");
11
12 $uri = $_SERVER["REQUEST_URI"];
13
14 $router = new Router();
15 $router->defineRoute("/login", "Auth/LoginController.php");
16 $router->defineRoute("/register", "Auth/RegisterController.php");
17 $router->defineRoute("/game/create", "Game/CreateGameController.php");
18 $router->defineRoute("/game/update", "Game/UpdateGameController.php");
19 $router->defineRoute("/game/delete", "Game/DeleteGameController.php");
20 $router->defineRoute("/game/getgamedata", "Game/GetGameDataController.php");
21 $router->defineRoute("/game/participants", "Game/GameParticipantsController.php");
22 $router->defineRoute("/game/round/create", "Game/CreateRoundController.php");
23 $router->defineRoute("/game/round/playeranswer", "Game/InsertRoundDataController.php");
24 $router->defineRoute("/game/round/check", "Game/CheckRoundController.php");
25 $router->defineRoute("/player/playerdata", "Player/PlayerDataController.php");
26 $router->defineRoute("/player/getleaderboard", "Player/GetLeaderboard.php");
27 $router->defineRoute("/player/updateleaderboard", "Player/UpdateLeaderboard.php");
28 $router->defineRoute("/player/getleaderboardstats", "Player/GetLeaderboardStats.php");
29
30 $router->redirect($uri);
31
```

Le fichier `index.php` de l'API qui est utilisé comme routeur - Liste des routes qui sont définies

```

Router.php ×
src > Models > Router.php
1  <?php
2
3  namespace CountryGuesser\Models;
4
5  require_once __DIR__ . "/../../vendor/autoload.php";
6
7 /**
8  * Router class
9  * This class provides a method to define routes and redirect to the correct controller.
10 */
11 class Router
12 {
13     private array $routes = [];
14
15     /**
16      * defineRoute method
17      * This method defines a route.
18
19      * @param string $route The route to define,
20      * @param string $controllerPath The path to the controller to redirect to.
21      * @return void
22  */
23     public function defineRoute(string $route, string $controllerPath): void
24     {
25         $this->routes[$route] = [
26             "controller" => $controllerPath,
27         ];
28     }
29
30     /**
31      * redirect method
32      * This method redirects to the correct controller.
33
34      * @param string $route The route to redirect to.
35      * @return void
36  */
37     public function redirect(string $route): void
38     {
39         if (array_key_exists($route, $this->routes)) {
40             $controller = $this->routes[$route]["controller"];
41             require_once __DIR__ . "/../Controllers/" . $controller;
42         } else {
43             header("location: https://pierreleocadie.notion.site/Documentation-Country-Guesser-API-51457d0229eb40198094ad8bbfde51fc");
44         }
45     }
46 }

```

Notre simple classe Router dans le dossier [src/Models/Router.php](#)

Autre exemple :

Si on ne spécifie pas de route ou que la route demandée n'existe pas.

Requête : <https://api.countryguesser.deletesystem32.fr> ou  
<https://api.countryguesser.deletesystem32.fr/sqdsqs>

Redirection vers : [Documentation - Country Guesser API](#)

Voici l'ensemble des requêtes qui sont prises en charge par l'API de Country Guesser :

<https://api.countryguesser.deletesystem32.fr/login>  
<https://api.countryguesser.deletesystem32.fr/register>  
<https://api.countryguesser.deletesystem32.fr/game/create>  
<https://api.countryguesser.deletesystem32.fr/game/update>  
<https://api.countryguesser.deletesystem32.fr/game/delete>

<https://api.countryguesser.deleteSystem32.fr/game/getgamedata>

<https://api.countryguesser.deleteSystem32.fr/game/participants>

<https://api.countryguesser.deleteSystem32.fr/game/round/create>

<https://api.countryguesser.deleteSystem32.fr/game/round/playeranswer>

<https://api.countryguesser.deleteSystem32.fr/game/round/check>

<https://api.countryguesser.deleteSystem32.fr/player/playerdata>

<https://api.countryguesser.deleteSystem32.fr/player/getleaderboard>

<https://api.countryguesser.deleteSystem32.fr/player/updateleaderboard>

<https://api.countryguesser.deleteSystem32.fr/player/getleaderboardstats>

Chaque requête est détaillée dans la  [Documentation - Country Guesser API](#)

## Le serveur websocket

Afin de rendre le jeu multi-joueur nous avions eu besoin de créer un serveur websocket. Pour créer le serveur websocket, nous avons utiliser le package PHP Workerman.

Lorsqu'un joueur lance une partie multi-joueur, le client se connecte au serveur websocket et place le joueur en file d'attente si aucun autre joueur n'est en recherche de partie. Si un autre joueur en recherche de partie est trouvé, une partie (une `room`) se crée.

À chaque fois qu'un client se connecte, il envoie un pays aléatoire à deviner pour la première manche de la partie, au serveur. C'est donc le pays aléatoire envoyé par le dernier joueur ayant rejoint une partie qui sera le pays à deviner lors de la première manche.

Lorsqu'une manche se termine, les clients de la partie sont informés par le serveur que la manche est terminée et lorsqu'une nouvelle manche est créée les clients de la partie sont également informés par le serveur.

Lorsqu'une manche se termine, chaque client de la partie envoie un pays aléatoire à deviner pour la prochaine manche au serveur.

Si un joueur quitte une partie, les autres joueurs sont exclus de la partie, la partie est supprimée sur le serveur websocket et la partie est également supprimée de la base de données ainsi que toutes les données liées à cette partie.

Lorsque la partie est terminée, les clients sont informés par le serveur et le serveur envoie aux clients toutes les données de la partie (les réponses de chaque joueur pour

chaque manche, le vainqueur de chaque manche...)

Le serveur websocket permet d'envoyer en même temps à chaque client les bonnes informations à chaque étape, pour le bon déroulement d'une partie. Le serveur websocket agit un peu comme un routeur de l'information entre les joueurs et garantie l'état de la partie.

Nous avons définis pour les interactions client ↔ serveur websocket, les types de messages, les messages, le formats des messages qui peuvent être envoyés du client → serveur websocket et du serveur websocket → client. (Pour plus de détails voir  [Documentation - Country Guesser API](#) dans la section **Discussion client ↔ Websocket server**)

Exemple de communication serveur websocket → client

```
// Send the room id to the players
$rooms->sendToRoom($roomId, json_encode(array("type" => "information", "informationType" => "roomCreated", "message" => "Room created")));
$rooms->sendToRoom($roomId, json_encode(array("type" => "information", "informationType" => "roomFull", "message" => "Room full")));
$rooms->sendToRoom($roomId, json_encode(array("type" => "information", "informationType" => "gameCreated", "message" => "Game created")));
$rooms->sendToRoom($roomId, json_encode(array("type" => "data", "roomId" => $roomId, "gameId" => $rooms->getGameId($roomId), "roomSize" => $roomSize)));
```

Pour les interactions serveur websocket → API Country Guesser, nous faisons les appels API avec cURL (fichier [server/API.php](#)). Exemple ci-dessous :

The screenshot shows a code editor window with the file 'API.php' open. The file contains PHP code for a class 'API' that interacts with an external API using cURL. The code includes a static root URL, a method to create a game, and logic to handle the response from the API.

```
1 <?php
2
3 class API
4 {
5     private static string $rootUrl = "https://api.countryguesser.deleteSystem32.fr";
6
7     public function createGame()
8     {
9         $curl = curl_init(self::$rootUrl . "/game/create");
10        curl_setopt($curl, CURLOPT_RETURNTRANSFER, true);
11        curl_setopt($curl, CURLOPT_TIMEOUT, 10);
12        curl_setopt($curl, CURLOPT_CUSTOMREQUEST, "GET");
13        $data = curl_exec($curl);
14        if($data === false)
15        {
16            return '\nCurl error: ' . curl_error($curl) . '\n';
17        }
18        else
19        {
20            $data = json_decode($data, true);
21            return $data["game_id"];
22        }
23    }
24 }
```

Les interactions :

client ↔ serveur websocket,

serveur websocket ↔ API Country Guesser

sont détaillés dans la [Documentation - Country Guesser API](#) dans les sections

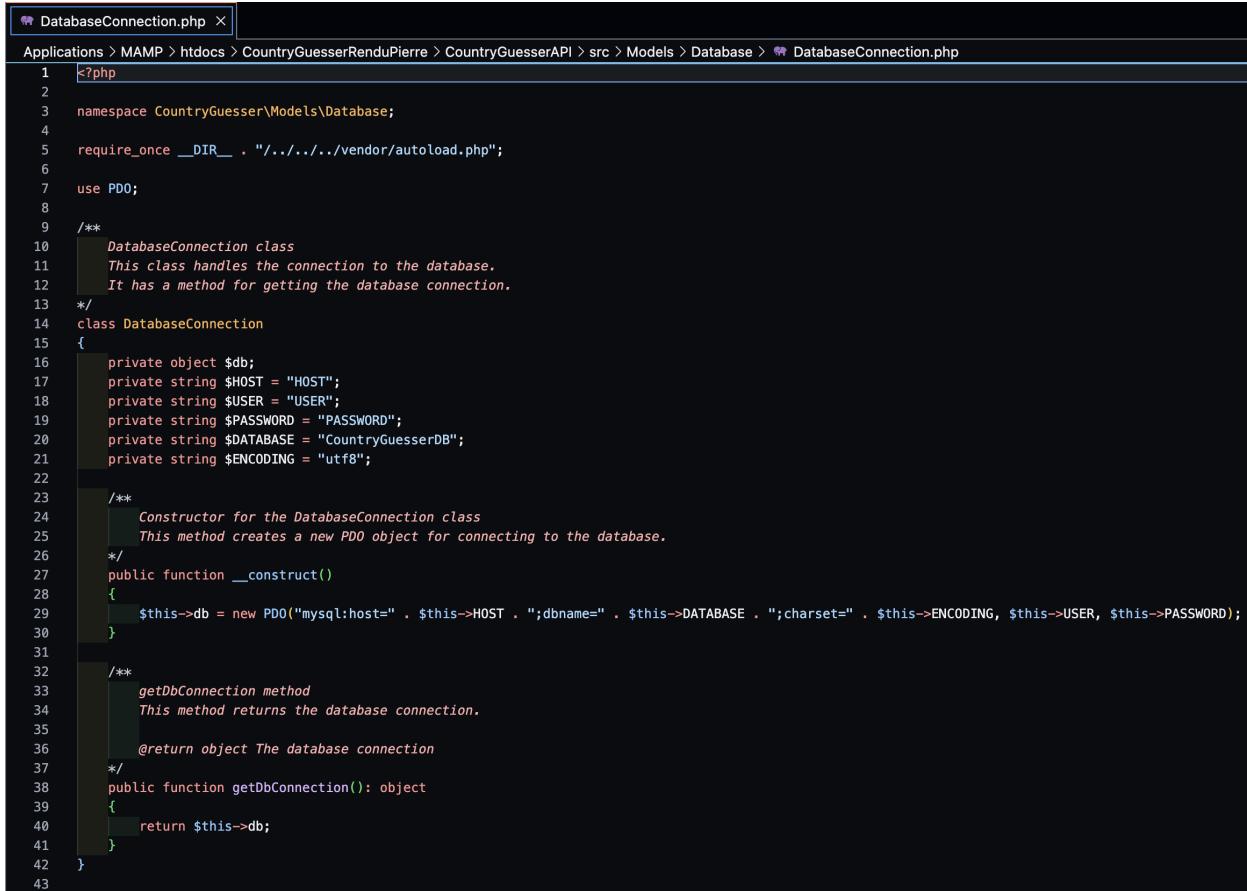
***Discussion client ↔ → Websocket server et Websocket server → API***

## Faire fonctionner l'API de Country Guesser et le serveur websocket en local

Afin de faire fonctionner l'API de Country Guesser en local, il est important de modifier le fichier `DatabaseConnection.php` qui se trouve dans le dossier

`CountryGuesserAPI/src/Models/Database` afin de mettre les bons identifiants de connexion à la base de données. Puis ouvrez votre terminal et rendez-vous dans le dossier

`CountryGuesserAPI/src/`, lancez la commande `php -S localhost:8000`



```

DatabaseConnection.php X
Applications > MAMP > htdocs > CountryGuesserRenduPierre > CountryGuesserAPI > src > Models > Database > DatabaseConnection.php
1  <?php
2
3  namespace CountryGuesser\Models\Database;
4
5  require_once __DIR__ . "/../../../../vendor/autoload.php";
6
7  use PDO;
8
9  /**
10   * DatabaseConnection class
11   * This class handles the connection to the database.
12   * It has a method for getting the database connection.
13  */
14  class DatabaseConnection
15  {
16      private object $db;
17      private string $HOST = "HOST";
18      private string $USER = "USER";
19      private string $PASSWORD = "PASSWORD";
20      private string $DATABASE = "CountryGuesserDB";
21      private string $ENCODING = "utf8";
22
23      /**
24       * Constructor for the DatabaseConnection class
25       * This method creates a new PDO object for connecting to the database.
26      */
27      public function __construct()
28      {
29          $this->db = new PDO("mysql:host=" . $this->HOST . ";dbname=" . $this->DATABASE . ";charset=" . $this->ENCODING, $this->USER, $this->PASSWORD);
30      }
31
32      /**
33       * getDbConnection method
34       * This method returns the database connection.
35      */
36      @return object The database connection
37
38      public function getDbConnection(): object
39      {
40          return $this->db;
41      }
42  }
43

```

Pour faire fonctionner le serveur websocket avec l'API CountryGuesser en local, il faut modifier le fichier `API.php` qui se trouve dans le dossier

`CountryGuesserWebSocketServer/server/` afin de modifier la variable `$rootUrl` par celui de l'API hébergée en local : `http://localhost:8000`. Puis ouvrez votre terminal et rendez-vous dans le dossier `CountryGuesserWebSocketServer`, lancez la commande `php server/server.php start`

API.php X

Applications > MAMP > htdocs > CountryGuesserWebSocketServer > server > API.php

```
1 <?php
2
3 class API
4 {
5     private static string $rootUrl = "https://api.countryguesser.deletesystem32.fr";
6
7     public function createGame()
8     {
9         $curl = curl_init(self::$rootUrl . "/game/create");
10        curl_setopt($curl, CURLOPT_RETURNTRANSFER, true);
11        curl_setopt($curl, CURLOPT_TIMEOUT, 10);
12        curl_setopt($curl, CURLOPT_CUSTOMREQUEST, "GET");
13        $data = curl_exec($curl);
14        if($data === false)
15        {
16            return '\nCurl error: ' . curl_error($curl) . '\n';
17        }
18        else
19        {
20            $data = json_decode($data, true);
21            return $data["game_id"];
22        }
23        curl_close($curl);
24    }
25
26    public function updateGame(int $gameId)
27    {
28        $dataToSend = array(
29            "game_id" => $gameId
30        );
31        $dataToSend = json_encode($dataToSend);
32        $curl = curl_init(self::$rootUrl . "/game/update");
33        curl_setopt($curl, CURLOPT_RETURNTRANSFER, true);
34        curl_setopt($curl, CURLOPT_TIMEOUT, 10);
35        curl_setopt($curl, CURLOPT_CUSTOMREQUEST, "POST");
36        curl_setopt($curl, CURLOPT_POSTFIELDS, $dataToSend);
37        $data = curl_exec($curl);
38        if($data === false)
39        {
40            echo '\nCurl error: ' . curl_error($curl) . '\n';
41        }
42        curl_close($curl);
43    }
44
45    public function createRound(int $gameId, int $roundId, string $response)
46    {
```

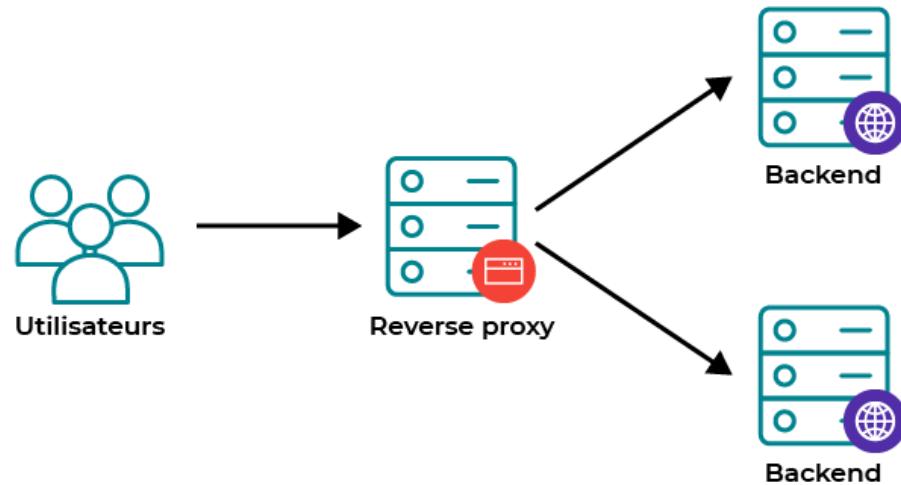
## Déploiement sur un serveur dans le cloud avec le service cloud Linode

Dans le but de pouvoir travailler plus facilement ensemble, nous avons déployé les services suivants :

- notre base de données Mysql,
  - `deletesystem32.fr:2818`
- PHPMyAdmin pour gérer plus simplement la base de données,
  - <https://phpmyadmin.countryguesser.deleteSystem32.fr/>
- l'API de Country Guesser
  - <https://api.countryguesser.deleteSystem32.fr/>
- le serveur websocket de Country Guesser
  - <ws://ws.countryguesser.deleteSystem32.fr/>
  - <https://ws.countryguesser.deleteSystem32.fr/>
- un reverse proxy nginx pour faire des redirections
  - <https://proxymanager.deleteSystem32.fr/>
- Portainer pour la gestion et le monitoring des containers Docker
  - <https://portainer.deleteSystem32.fr/>

Nous avons utilisé Docker afin de pouvoir déployer, gérer et monitoner plus simplement chaque service.

Puisque nous avons qu'un seul serveur, nous avons utilisé un reverse proxy nginx associé à une base de données MariaDB afin de pouvoir faire des redirections vers les ports de chaque service qui sont lancés dans des containers Docker en utilisant des sous-domaines. De plus le reverse proxy nginx nous permet de bénéficier de certificats SSL pour nos différents services et donc d'avoir de l'HTTPS.



Fonctionnement d'un reverse proxy

The screenshot shows the Linode DNS Manager interface. The top navigation bar includes tabs for 'DNS Records', 'Cloudflare', and 'Cloudflare DNS'. The main content area is titled 'cloud.linode.com/domains/1924486'. On the left, there's a sidebar with various icons for managing domains, subdomains, and records. The main panel is divided into sections:

- MX Record**: Contains a table with columns 'Mail Server', 'Preference', 'Subdomain', and 'TTL'. A button 'Add a MX Record' is visible.
- A/AAAA Record**: Contains a table with columns 'Hostname', 'IP Address', and 'TTL'. It lists several entries with IP addresses 178.79.134.83. A button 'Add an A/AAAA Record' is visible.
- CNAME Record**: Contains a table with columns 'Hostname', 'Aliases to', and 'TTL'. A button 'Add a CNAME Record' is visible.

Gestion du nom de domaine utilisé pour le projet et des sous-domaines associés avec le service cloud Linode

The screenshot shows the Nginx Proxy Manager interface. At the top, there's a header with the title "Nginx Proxy Manager" and a sub-header "proxymanager.deleteyestem32.fr/nginx/proxy". Below the header is a navigation bar with links: Dashboard, Hosts, Access Lists, SSL Certificates, Users, Audit Log, and Settings. On the right side of the header, there's a user profile for "Admin-Pierre-Leocadie" and a link to "Administrator". The main content area is titled "Proxy Hosts" and contains a table with the following data:

SOURCE	DESTINATION	SSL	ACCESS	STATUS
<a href="#">api.countryguesser.deleteyestem32.fr</a> Created: 4th December 2022	http://deleteyestem32.fr:2819	Let's Encrypt	Public	● Online
<a href="#">edit.deleteyestem32.fr</a> Created: 28th November 2022	http://deleteyestem32.fr:1408	Let's Encrypt	Public	● Online
<a href="#">phpmyadmin.countryguesser.deleteyestem32.fr</a> Created: 28th November 2022	http://deleteyestem32.fr:2817	Let's Encrypt	Public	● Online
<a href="#">portainer.deleteyestem32.fr</a> Created: 25th December 2022	http://deleteyestem32.fr:9000	Let's Encrypt	Public	● Online
<a href="#">proxymanager.deleteyestem32.fr</a> Created: 28th November 2022	http://deleteyestem32.fr:81	Let's Encrypt	Public	● Online
<a href="#">ws.countryguesser.deleteyestem32.fr</a> Created: 16th December 2022	http://deleteyestem32.fr:2820	Let's Encrypt	Public	● Online

v2.9.19 © 2022 jc21.com. Theme by Tabler

Fork me on Github

Les redirections depuis les sous-domaines vers des ports spécifiques liés aux différents services que nous avons déployés

The screenshot shows the "Edit Proxy Host" dialog box over a blurred background of the Nginx Proxy Manager dashboard. The dialog has a tabbed interface with "Details" selected. The "Domain Names" field contains "api.countryguesser.deleteyestem32.fr". The "Scheme" dropdown is set to "http", the "Forward Hostname / IP" dropdown is set to "deleteyestem32.fr", and the "Forward Port" dropdown is set to "2819". There are two toggle buttons: "Cache Assets" (disabled) and "Block Common Exploits" (enabled). The "Access List" field contains "Publicly Accessible". At the bottom right of the dialog are "Cancel" and "Save" buttons.

Paramétrage d'une redirection

### Paramétrage d'une redirection - Certificat SSL

Ce qui nous permet par exemple d'accéder à PHPMyAdmin avec le lien suivant :

<https://phpmyadmin.countryguesser.deleteSystem32.fr/> au lieu de

<http://deleteSystem32.fr:2817>

ou encore de ce connecter au serveur websocket avec le lien suivant :

<ws://ws.countryguesser.deleteSystem32.fr/>

Cela a pour but de nous simplifier la tâche pour accéder à nos différents services mais également afin que ce soit plus lisible dans le code, sans que nous ayons à nous demander sur quel port avons nous héberger tel ou tel service.

Nous avons également créé un script python pour synchroniser les différents repos GitHub et la production sur le serveur dans le cloud Linode. On crée un cronjob sur serveur cloud qui execute le script python tous les x temps. Le script python va aller vérifier si un nouveau commit a été fait sur le repo souhaité et comparer avec la version existante en production, si les deux versions diffèrent, il va supprimer le dossier du repo existant sur le serveur, arrêter et supprimer le container docker lié, clone le repo, build et run le container docker.

```
import os, json

PERSONAL_ACCESS_TOKEN = "YOUR_PERSONAL_GITHUB_ACCESS_TOKEN"
```

```

OWNER = "GITHUB_REPO_OWNER"
REPO = "GITHUB_REPO_NAME"
CONTAINER = "DOCKER_CONTAINER_NAME"
PATH = "PATH_TO_REPO"
PORT_MAPPING = "PORT_MAPPING_FOR_DOCKER_CONTAINER"

CURL_REQUEST = f'curl --request GET \
    --url "https://api.github.com/repos/{OWNER}/{REPO}/commits" \
    --header "Accept: application/vnd.github+json" \
    --header "Authorization: Bearer {PERSONAL_ACCESS_TOKEN}" > {PATH}/lastCommit-1.json'

BUILD_AND_RUN_CONTAINER = f"cd {PATH}/; \
    git clone https://github.com/{OWNER}/{REPO}.git; \
        docker stop {CONTAINER}; \
        docker rm {CONTAINER}; \
        cd {PATH}/{REPO}; \
        docker build -t {CONTAINER} .; \
        docker run -d --name {CONTAINER} -p {PORT_MAPPING} {CONTAINER}"

def checkRemoveBuildRun():
    if(os.path.exists(f"{PATH}/{REPO}")):
        os.system(f"rm -rf {PATH}/{REPO}")
    os.system(BUILD_AND_RUN_CONTAINER)

    if(os.path.exists(f"{PATH}/lastCommit.json")):
        os.system(CURL_REQUEST)
        with open(f"{PATH}/lastCommit.json", "r") as f:
            if(f.read() != open(f"{PATH}/lastCommit-1.json", "r").read()):
                checkRemoveBuildRun()
            os.system(f"rm {PATH}/lastCommit.json; \
                mv {PATH}/lastCommit-1.json {PATH}/lastCommit.json")
    else:
        os.system(CURL_REQUEST)
        checkRemoveBuildRun()
        os.system(f"mv {PATH}/lastCommit-1.json {PATH}/lastCommit.json")

```

```

*/1 * * * * python3 /root/countryguesser_api_service/checkGithubRepoNewCommit.py 2>> /root/countryguesser_api_service/cronjoberr.txt
*/1 * * * * date >> /root/countryguesser_api_service/cronjoblog.txt
*/1 * * * * python3 /root/countryguesser_websocket_server/Sync.py 2>> /root/countryguesser_websocket_server/cronjoberr.txt
*/1 * * * * date >> /root/countryguesser_websocket_server/cronjoblog.txt

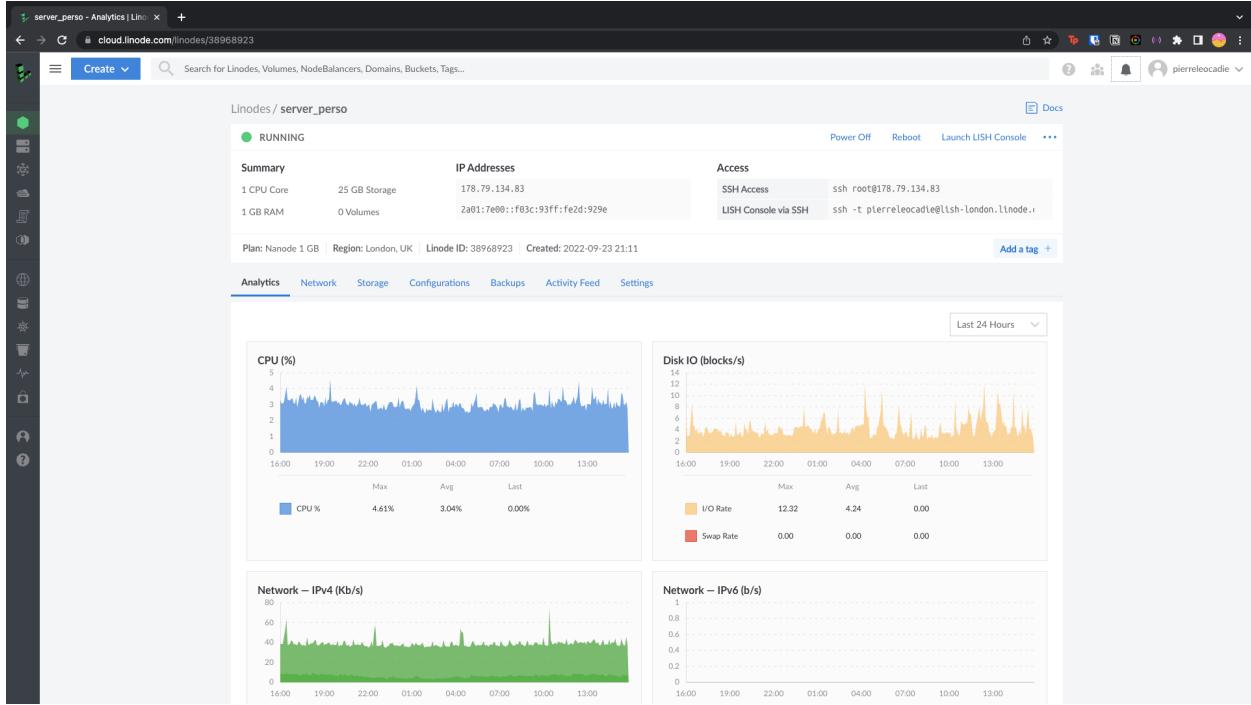
```

Cronjob pour synchroniser

Vous pouvez si vous le souhaitez consulter directement notre base données avec PHPMyAdmin, ou nos différents containers Docker des différents services avec Portainer à l'aide des identifiants que nous vous avons créé pour l'occasion et que nous vous avons transmis par mail.

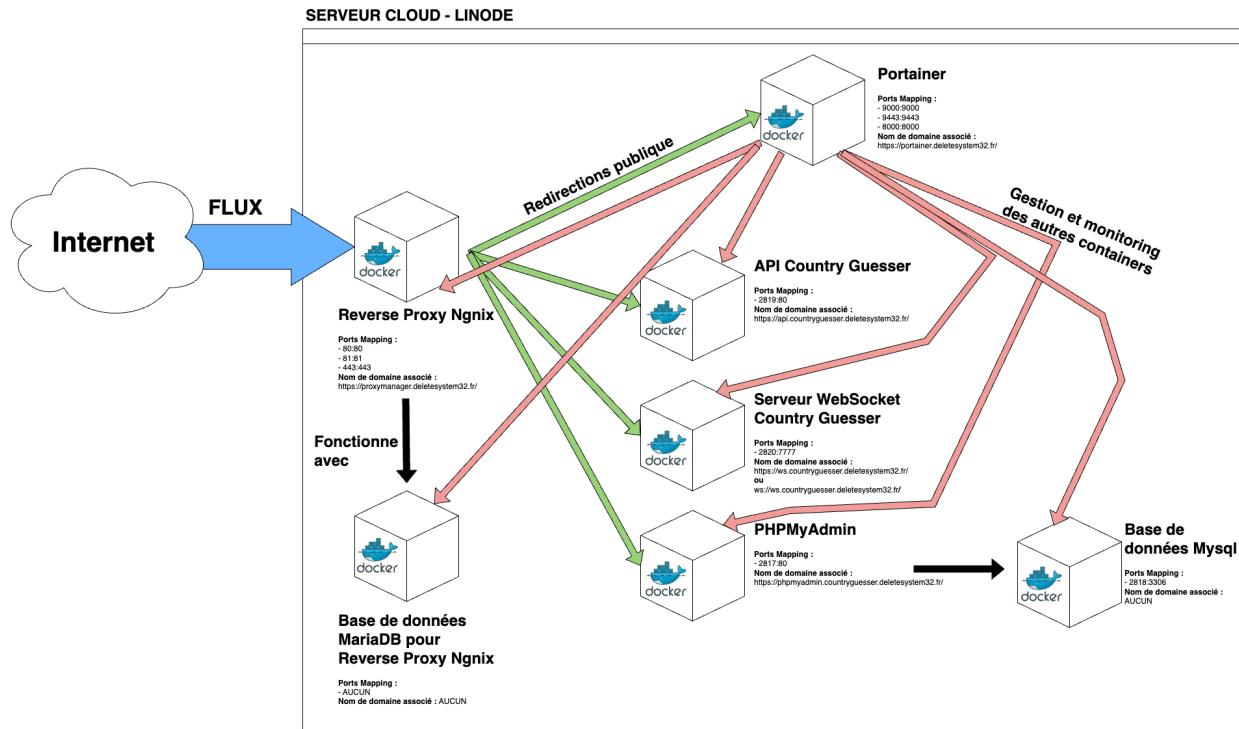
PHPMyAdmin : <https://phpmyadmin.countryguesser.deleteSystem32.fr/>  
 Portainer : <https://portainer.deleteSystem32.fr/>

## Dashboard Linode pour gérer et montrer le serveur cloud



Dashboard linode qui permet de gérer et montrer le serveur

## Schéma récapitulatif de l'infrastructure back-end de Country Guesser



L'infrastructure de notre serveur cloud

## Front-end

### Accueil



Pour la page d'accueil, nous avons opté pour une interface futuriste et simple à utiliser pour tout le monde.

Elle comporte une barre de navigation avec quelques onglets comme les statistiques des parties des joueurs, une page à propos du projet et un onglet pour pouvoir se connecter.

The screenshot shows the 'Statistiques' (Statistics) page of the 'Country Guesser' application. At the top, there is a navigation bar with links for 'ACCUEIL', 'STATISTIQUES', and 'À PROPOS'. A user profile icon is also present. Below the navigation bar, the title 'Vos statistiques' is displayed. Underneath, three metrics are shown: '0 parties gagnées' (with a trophy icon), '0 parties perdues' (with a red cross icon), and '0 parties jouées' (with a play button icon). The next section, 'Classement - Multijoueurs', displays a table of player statistics. The table has columns for 'Nom du joueur' (Player Name), 'Parties jouées' (Parties Played), and 'G/P/Ratio'. The data is as follows:

Nom du joueur	Parties jouées	G/P/Ratio
test2	72	11/61/0.18
test	83	4/79/0.05
pierre.leocadie	2	2/0/2.00
jordan.baumard	0	0/0/0.00
Mathis	0	0/0/0.00
test3	0	0/0/0.00
sexygaldem	2	0/2/0.00

La page statistiques contient : (uniquement les statistiques du mode multijoueur !)

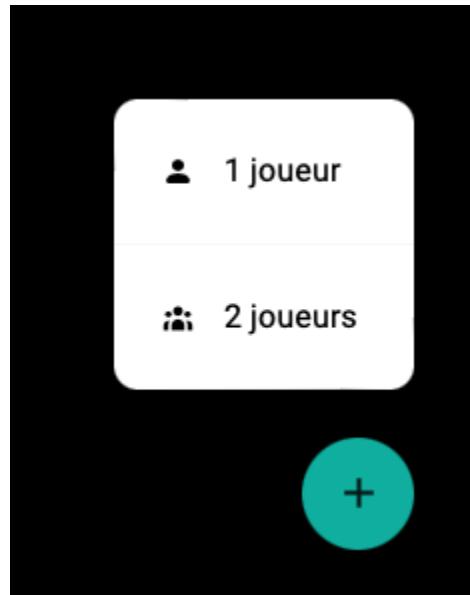
- le nombre de parties gagnées du joueur
- le nombre de parties perdues
- le nombre de parties jouées
- le classement des meilleurs joueurs

Par la suite, nous souhaiterions ajouter une barre de recherche afin de trouver directement sa position dans le classement.

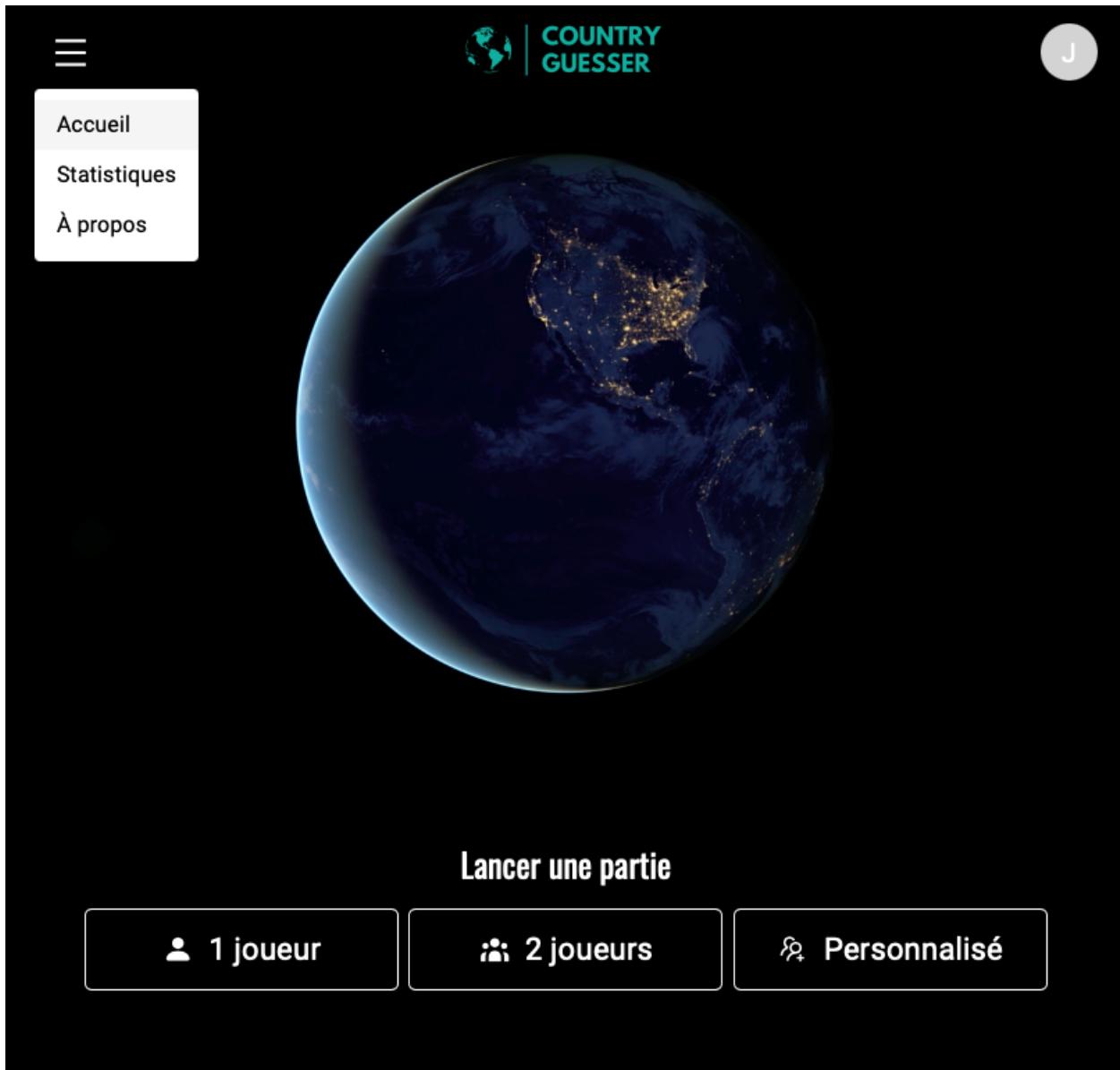
The screenshot shows the 'À propos / Crédits' (About / Credits) page of the CountryGuesser website. At the top left is the logo 'COUNTRY GUESSER' with a small globe icon. At the top right is a user profile icon with the letter 'J'. The main title 'À propos / Crédits' is centered at the top. Below it is the heading 'Développeurs' followed by a list of names: Jordan BAUMARD, Pierre LÉOCADIE, and Charles HURST. A note below states 'Étudiants à l'IUT de Paris - Rives de Seine (France)'. The text 'Date de réalisation : nov/déc 2022' follows, along with a thank you message 'Merci de jouer à CountryGuesser !'. At the bottom right is a teal circular button with a white plus sign, and at the bottom left is a teal circular button with a white right-pointing arrow.

La page “À propos” contient la signature du site, c'est-à-dire nos noms et le nom de l'IUT. Une animation de défilement est présente type “Star Wars” qu'on peut mettre en pause en cliquant sur l'écran.

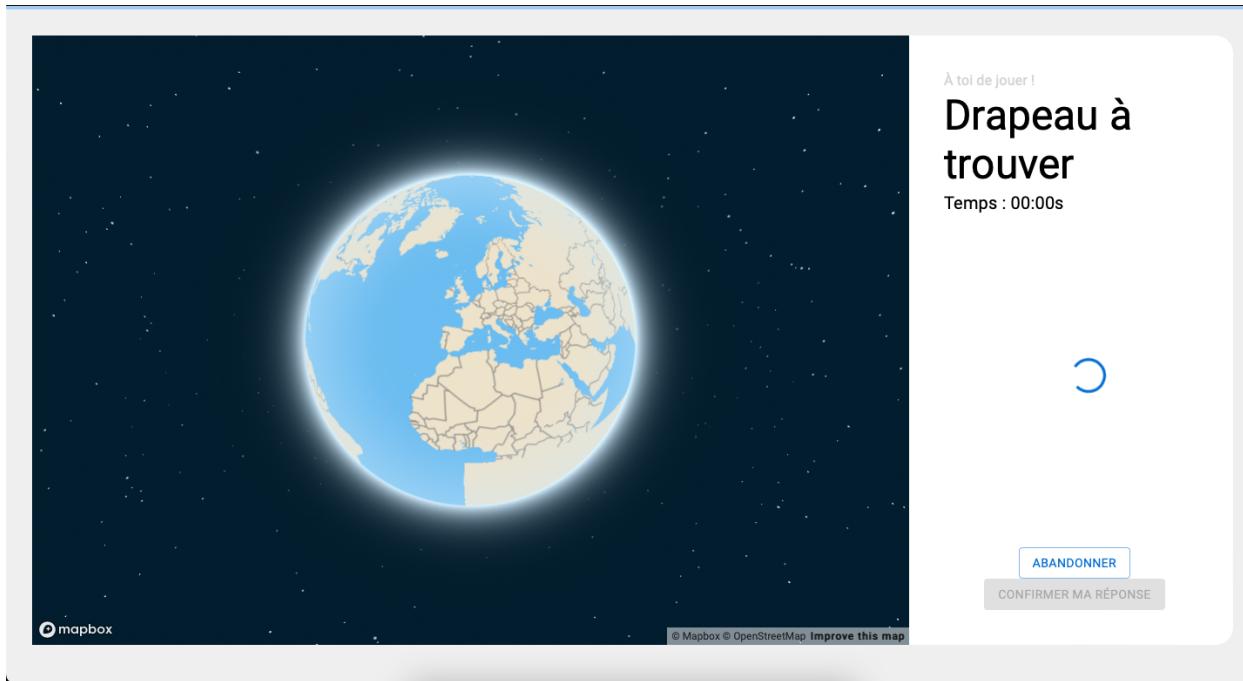
## Fonctionnalités pratiques



Possibilité de lancer une partie depuis n'importe quelle page grâce au bouton flottant situé en bas à droite.

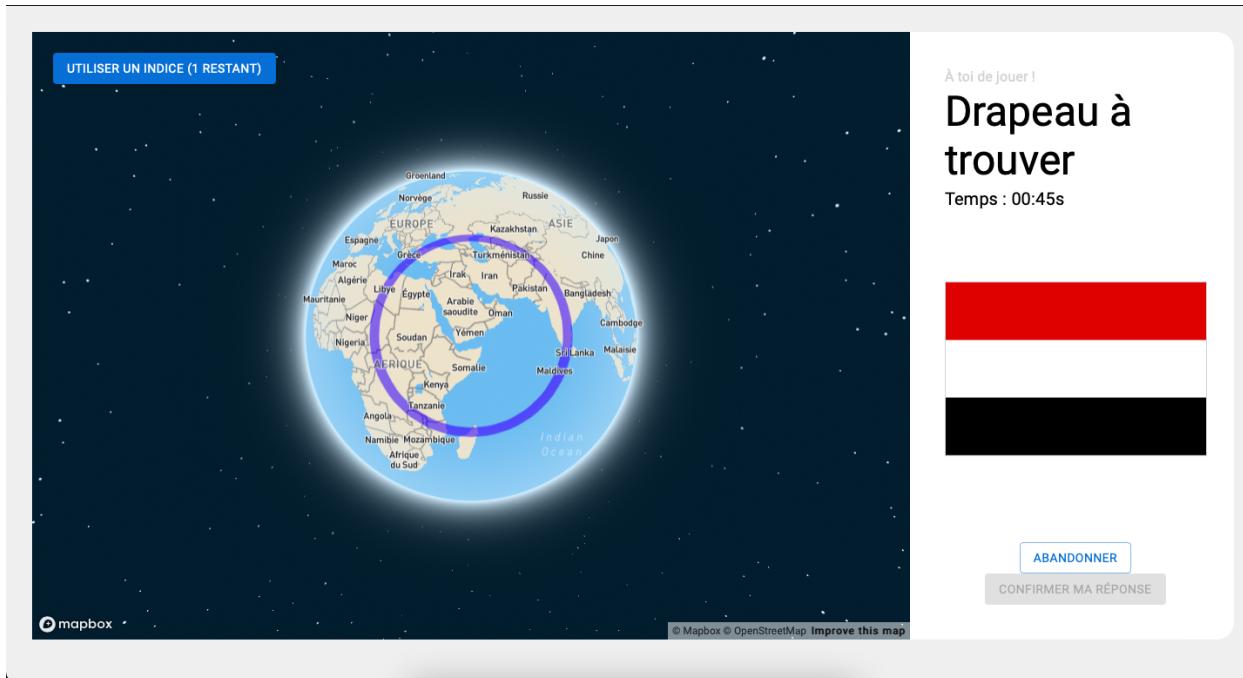


Le site est entièrement responsive, ce qui permet à n'importe qui de jouer sur mobile, tablette et PC depuis n'importe où.



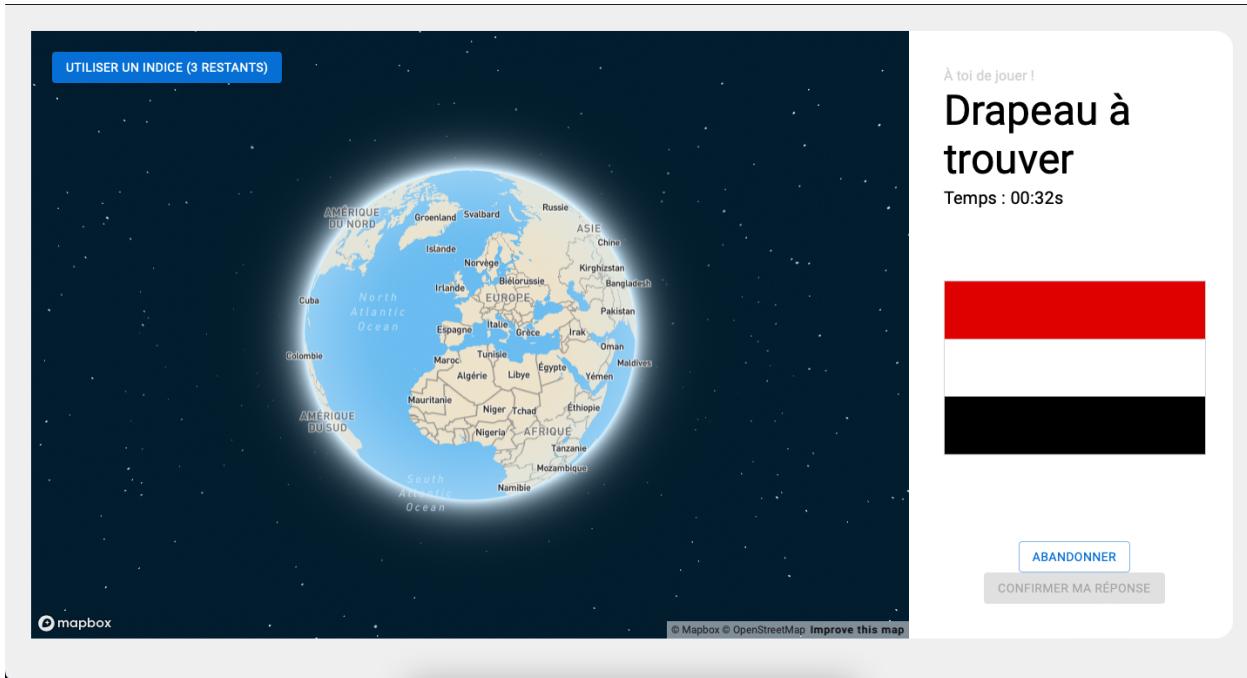
Lors du chargement des informations nécessaires au jeu, une barre de chargement en haut défile et un rond de chargement est situé à la place du drapeau.

De plus, les boutons sont désactivés pour indiquer à l'utilisateur qu'il doit patienter.



L'interface du jeu propose des indices sous forme de cercles. Le pays à deviner est présent dans le cercle. Il est possible d'utiliser jusqu'à 3 indices :

- Cercle très large (difficile)
- Cercle large (assez difficile)
- Cercle proche (facile)



Quelques animations sont présentes sur le jeu, les voici :

- Quand on abandonne la partie, la caméra vole jusqu'au pays qui était à deviner
- Quand on valide un mauvais pays, l'écran tremble pour montrer que ce n'est pas la bonne réponse

**UTILISER UN INDICE (1 RESTANT)**

Votre choix :  
Yemen

Pays sélectionné : Yemen

© Mapbox © OpenStreetMap Improve this map

À toi de jouer !

## Drapeau à trouver

Temps : 01:02s

Vous êtes sur le point de valider votre réponse : Yemen

ABANDONNER
**CONFIRMER MA RÉPONSE**

Lorsqu'on sélectionne un pays, le bouton "Confirmer ma réponse" se débloque, une popup apparaît à l'endroit cliqué pour indiquer le pays sélectionné. Il est également indiqué en bas à droite de la carte du globe et juste au dessus du bouton "Confirmer".

Recherche d'un joueur en cours...

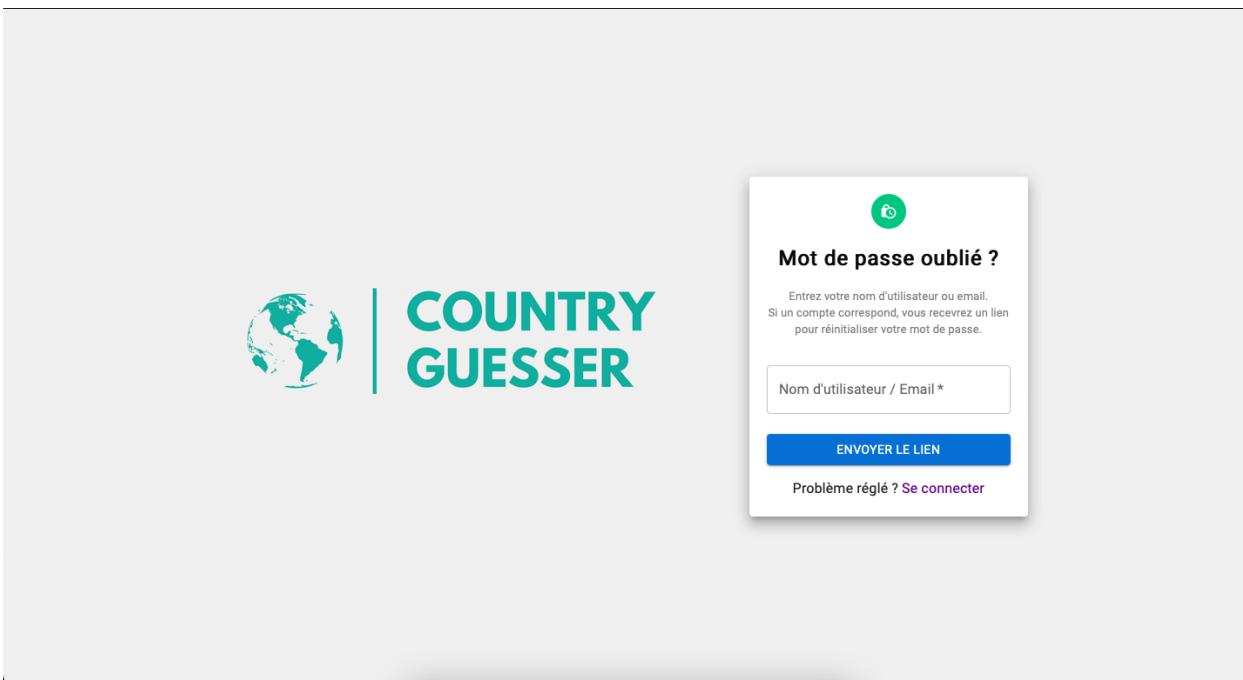
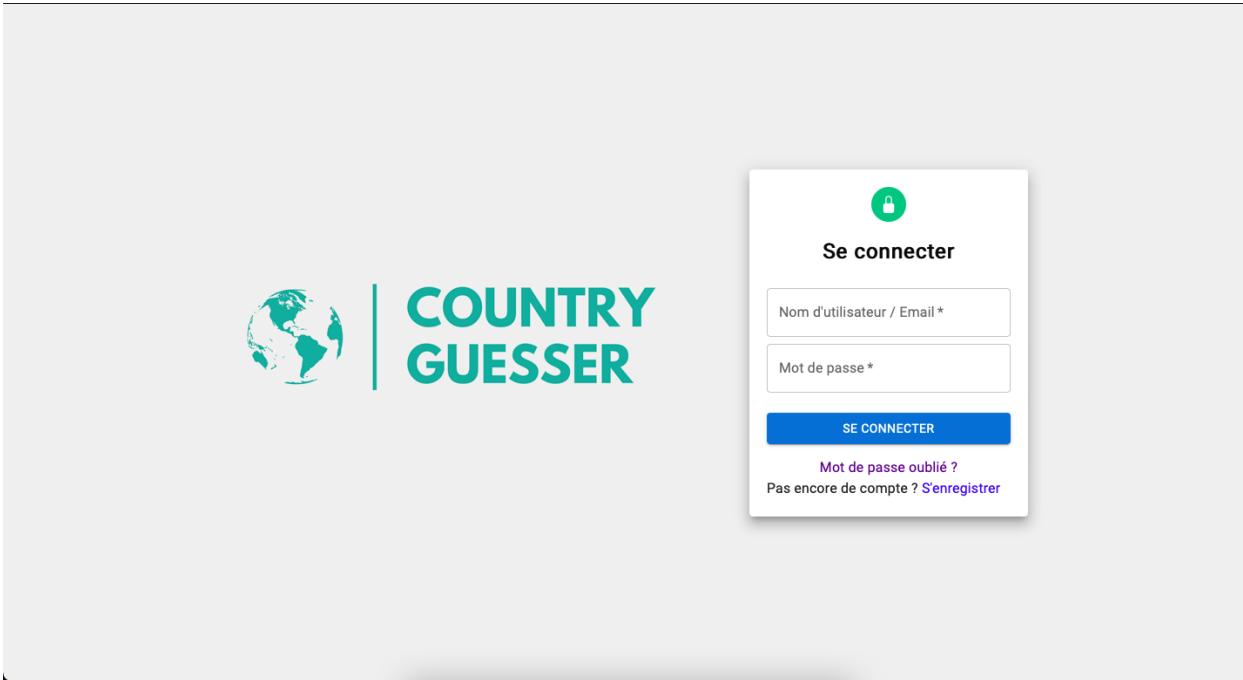
"Life is not measured by the breaths we take, but by the moments that take our breath."  
- Inconnu

**Question :** Entrainement

Dans quel pays peut-on visiter le Taj Mahal ?

- 1** Mexique
- 2** Inde
- 3** Maroc
- 4** Chine

Lorsqu'un joueur cherche une partie multijoueurs disponible, il y a une animation de terre qui tourne, une citation de motivation et un questionnaire pour l'occuper !



Un joueur peut se connecter/créer son compte. Il y a une page mot de passe oublié mais celle-ci ne fonctionne pas. C'est une amélioration que nous prévoyons de faire par

la suite.

## Installer la partie front-end en local

### Étape 1 -

- Rendez-vous dans le dossier CountryGuesserUI

ou

- Ouvrez un terminal et clonez le dépôt Git suivant :

<https://github.com/jordanbmrd/CountryGuesser> à l'aide de la commande :

```
git clone https://github.com/jordanbmrd/CountryGuesser.git && cd CountryGuesser
```

**Étape 2** - Installez les dépendances (il faut avoir installé NodeJS au préalable pour utiliser la commande npm, si vous ne l'avez pas installé, installez-le et relancez votre terminal) :

```
npm install
```

**Étape 3** - Si vous avez installé la partie backend en local, le lien du websocket et le lien de l'api seront différents, modifiez ceux qui sont dans le fichier .env avec les URL que vous souhaitez utiliser, sinon, laissez-ceux à distance.

**Étape 4** - Lancez le serveur local :

```
npm start
```

## Les points d'améliorations à prévoir

- Optimisation de la base de données
- Optimisation des interactions API ↔ base de données, utilisation de procédures et fonctions stockées SQL
- Refactoring et optimisation du serveur websocket
- Améliorer notre architecture MVC

- Donner à l'utilisateur la possibilité de modifier ses informations (pseudonymes, email, mot de passe)
- Intégrer une messagerie lorsque le joueur se trouve dans une partie multijoueurs
- Gérer le cas du mot de passe oublié