

|  |
| --- |
| MorpiOnline  Documentation technique |
|  |
| 29 mars  Leo Montagna  Hodheil Melloul  Hugo Bao  Brian Grin  I.DA-P4B |



Table des matières

[Introduction 3](#_Toc99462549)

[Description de l’application 3](#_Toc99462550)

[Planification 3](#_Toc99462551)

[Diagramme de Gantt 3](#_Toc99462552)

[Analyse Organique 4](#_Toc99462553)

[Base de données 4](#_Toc99462554)

[Dictionnaire de données 4](#_Toc99462555)

[Structure du projet 5](#_Toc99462556)

[Technologies utilisées 5](#_Toc99462557)

[Routes de l’API 6](#_Toc99462558)

[Analyse fonctionnelle 7](#_Toc99462559)

[Plan de test 11](#_Toc99462560)

[Description des tests 11](#_Toc99462561)

[Scénarios de tests 13](#_Toc99462562)

[Conclusion 14](#_Toc99462563)

[Difficultés rencontrées 14](#_Toc99462564)

[Améliorations possibles 14](#_Toc99462565)

[Bilan Personnel 15](#_Toc99462566)

# Introduction

MorpiOnline est un jeu de morpion en ligne conçu pour être multiplateforme. Un serveur fait la liaison permettant à un joueur sur navigateur de jouer avec un joueur possédant le client CSharp.

|  |
| --- |
| Description de l’application L’application est divisée en plusieurs parties : un serveur express (javascript), un client web ainsi qu’un client CSharp. Le serveur fait la liaison entre les différentes plateformes permettant au client web de communiquer avec le client CSharp de manière instantanée. Pour réaliser ceci, nous avons utilisé un Framework nommé ExpressJS afin de créer une api permettant aux joueurs de se connecter sur les différents clients. Pour ce qui en est du jeu, nous avons utilisé le Framework Socket.IO. Ce Framework est indispensable pour notre projet car il permet de gérer des actions sur plusieurs clients en temps réel. |

# Planification

### Diagramme de Gantt

Sur le diagramme ci-dessous, vous pouvez apercevoir la répartition des tâches du projet sur le temps qui nous a été mis à disposition. Dans l’ensemble, toutes les tâches ont été réalisées sans grande difficulté. Les tâches ayant pris plus de temps que prévu sont la création du serveur ainsi que les clients. Les tâches ont été répartis équitablement entre chaque membre de l’équipe.

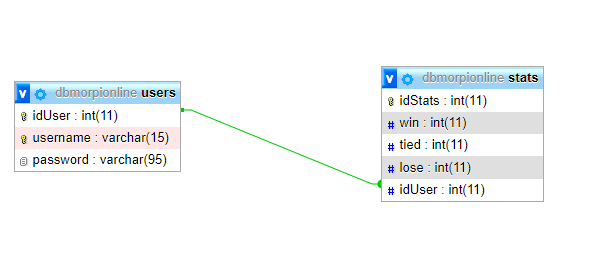
Une image contenant table

Description générée automatiquement

# Analyse Organique

### Base de données

MorpiOnline utilise une base de données MySQL afin de stocker les infos de connexion des utilisateurs ainsi que les statistiques de leurs parties. Ces informations sont ensuite appelées par l’API qui est elle-même appelé par les clients afin de récupérer les informations de connexion ce qui permettra de connecter les utilisateurs.



### Dictionnaire de données

##### users

|  |  |  |  |
| --- | --- | --- | --- |
| Colonne | Type | Null | Défaut |
| idUser | int(11) | Non |  |
| Username | varchar(15) | Non |  |
| Password | Varchar(95) | Non |  |

##### stats

|  |  |  |  |
| --- | --- | --- | --- |
| Colonne | Type | Null | Défaut |
| idStats | int(11) | Non |  |
| win | int(11) | Non | 0 |
| tied | int(11) | Non | 0 |
| lose | int(11) | Non | 0 |
| idUser | int(11) | Non |  |

### Structure du projet

Le projet contient plusieurs structures différentes que ce soit pour le serveur, le client web ainsi que le client CSharp.

##### Serveur node

* /db/ : contient le fichier de la base de données
* /src/ : contient tous les fichiers relatifs au fonctionnement du serveur

##### Client Web

* /css/ : contient toutes les feuilles de style de l’application
* /img/ : contient tous les éléments graphiques de l’application
* /js/ : contient tous les scripts nécessaires au bon fonctionnement de l’application

##### Client CSharp

* **/morpiONLINE-client/**: contient la solution Visual Studio du projet

### Technologies utilisées

Afin de réaliser ce projet dans les meilleures conditions, nous avons dû utiliser différentes technologies afin de regrouper toutes les parties du projet.

##### Serveur

* **expressJS**: utilisé pour la création de l’API et de ses routes
* **Socket.IO**: utilisé pour la connexion en direct avec les clients afin d’avoir le jeu en temps réel
* **Dotenv**: permet de gérer les informations importantes du serveur qui n’est pas publié sur le git. Cela permet de ne pas avoir des valeurs sensibles en dur dans le code.
* **Nodemon**: permet de ne pas avoir besoin de redémarrer le serveur manuellement à chaque modification. Nodemon le fait tout seul.

##### Client Web

* **Socket.IO**: utilisé afin de se connecter au serveur et de se lier à un autre client
* **Svelte** : utilisé afin d’avoir une application monopage.

##### Client CSharp

* **Socket.IO**: utilisé afin de se connecter au serveur et de se lier à un autre client

### Routes de l’API

/signup

Permet de créer un compte

/signin

Permet de se connecter

Une image contenant texte

Description générée automatiquement

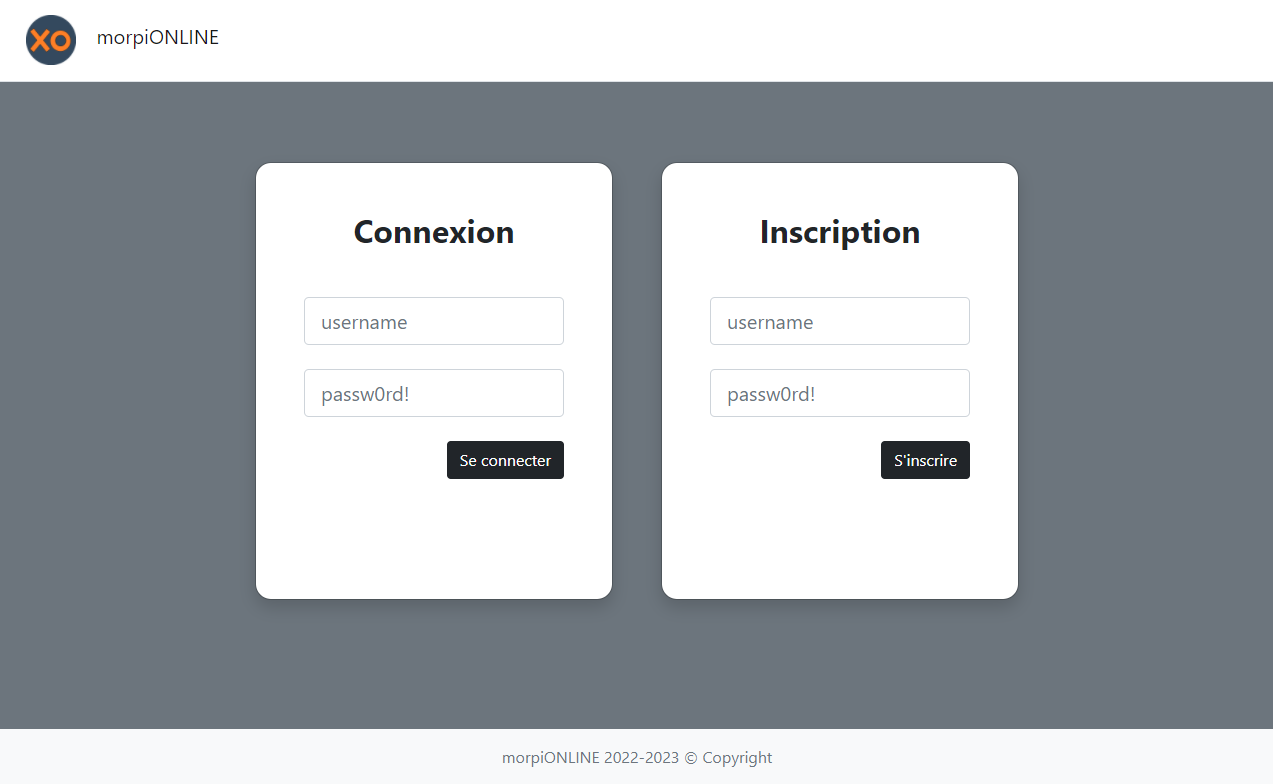
Vous trouverez ci-dessus les paramètres nécessaires à passer dans le body afin d’appeler l’API.

# Analyse fonctionnelle

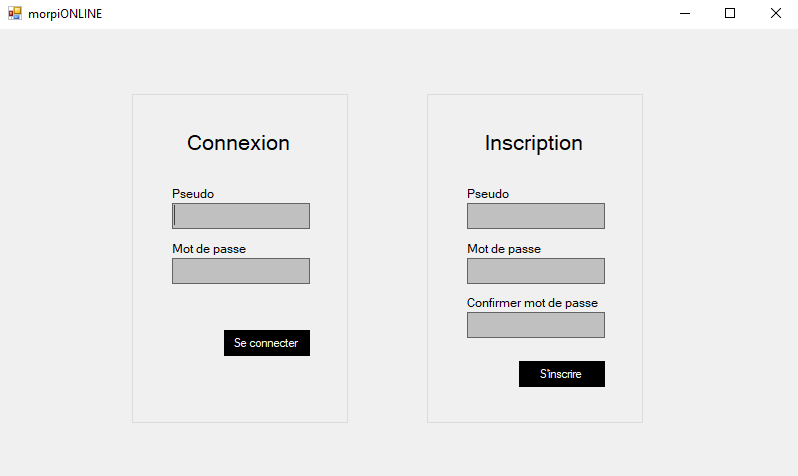
##### Page de connexion

La page de connexion permet à l’utilisateur de créer un compte ou de se connecter afin de pouvoir jouer au jeu. Il faut être connecté pour pouvoir jouer.

Web



CSharp



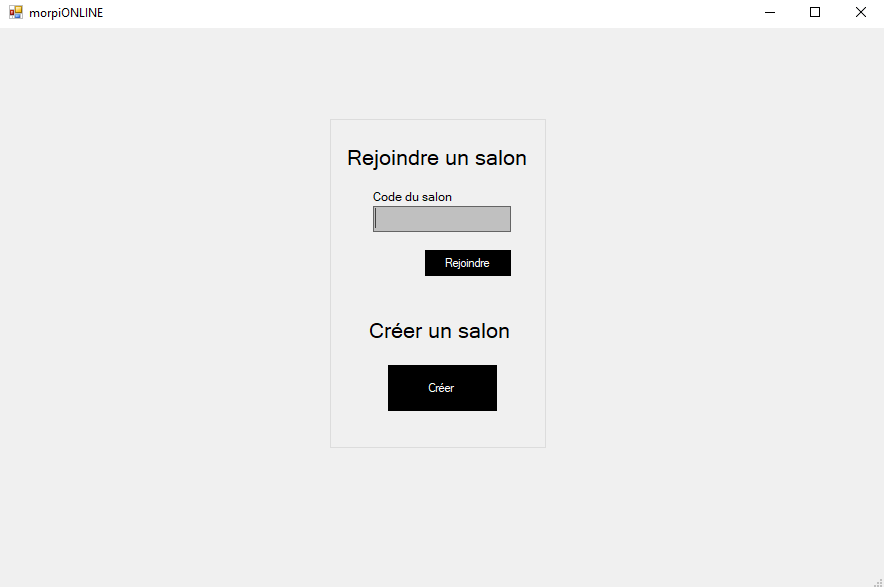
###### Page de création d’un salon

Cette page permet à l’utilisateur de créer un salon ou d’en rejoindre un existant

Web



CSharp



###### Page d’attente (dans le salon)

Cette page permet à l’utilisateur d’attendre le lancement de la partie et de laisser le temps aux joueurs de se connecter et de se préparer.

Web



CSharp

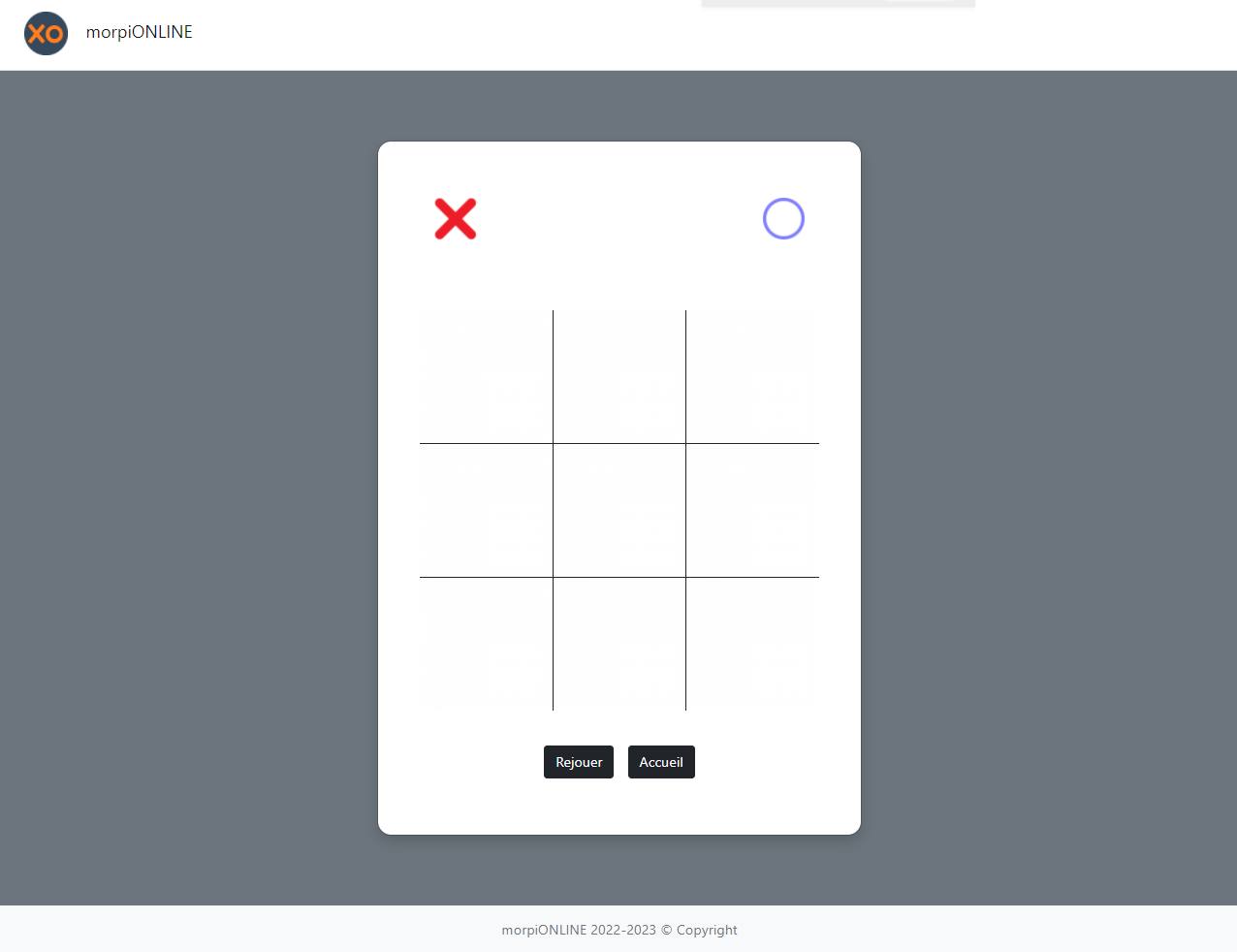
Une image contenant texte

Description générée automatiquement

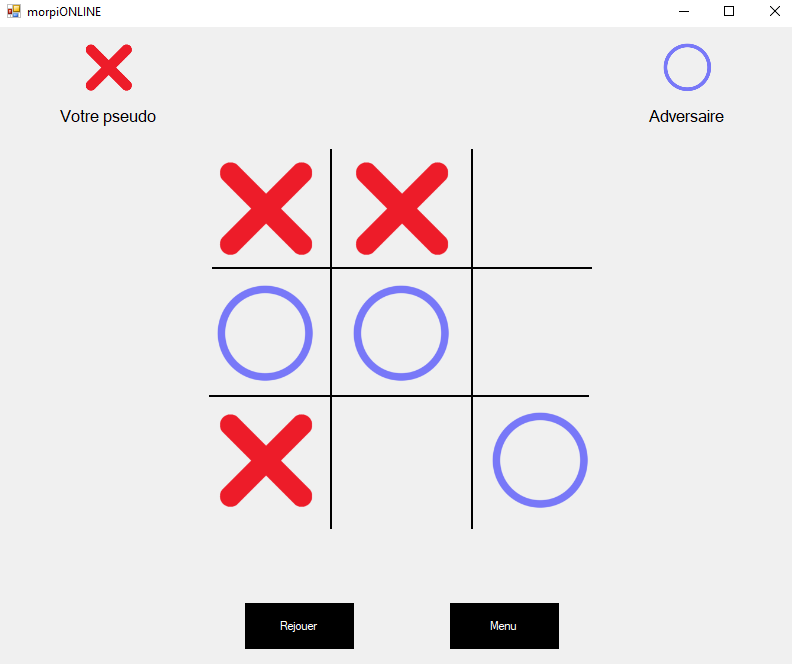
###### Page en jeu

Cette page est la page la plus importante de l’application. Elle permet d’effectuer le but premier de l’application : jouer. Une fois cette page lancée, tout est prêt. Il ne vous reste plus qu’à jouer.

Web



CSharp



# Plan de test

Dans le cadre du développement de mon application, il est exigé que je mette en place un plan de test.

### Description des tests

|  |  |
| --- | --- |
| Nom du test | 1.1 Inscription – Inscription réussie |
| Description | Créer un nouvel utilisateur. |
| Comment le tester | Effectuer une requête HTTP Post avec un JSON contenant un username et password sur l’endpoint /signup |

|  |  |
| --- | --- |
| Nom du test | 1.2 Connexion – Connexion réussie |
| Description | Se connecter à un utilisateur existant. |
| Comment le tester | Effectuer une requête HTTP Post avec un JSON contenant un username et password sur l’endpoint /signin |

|  |  |
| --- | --- |
| Nom du test | 1.3 Inscription – Nom d’utilisateur trop court |
| Description | Crée un nouvel utilisateur de moins de 8 caractères |
| Comment le tester | Effectuer une requête HTTP Post avec un JSON contenant un username de moins de 8 caractères et password sur l’endpoint /signup |

|  |  |
| --- | --- |
| Nom du test | 1.4 Inscription – Nom d’utilisateur existant |
| Description | Crée un nouvel utilisateur dont le nom existe déjà |
| Comment le tester | Effectuer une requête HTTP Post avec un JSON contenant un username et password sur l’endpoint /signup |

|  |  |
| --- | --- |
| Nom du test | 1.5 Inscription – Mot de passe trop court |
| Description | Crée un nouvel utilisateur avec un mot de passe de moins de 8 caractères. |
| Comment le tester | Effectuer une requête HTTP Post avec un JSON contenant un username et password de moins de 8 caractères sur l’endpoint /signup |

|  |  |
| --- | --- |
| Nom du test | 1.6 Connexion – Mot de passe incorrect |
| Description | Se connecter avec un mauvais mot de passe |
| Comment le tester | Effectuer une requête HTTP Post avec un JSON contenant un username et password sur l’endpoint /signin |

|  |  |
| --- | --- |
| Nom du test | 1.7 Connexion – L’utilisateur n’existe pas |
| Description | Se connecter avec un nom d’utilisateur inexistant. |
| Comment le tester | Effectuer une requête HTTP Post avec un JSON contenant un username et password sur l’endpoint /signin |

|  |  |
| --- | --- |
| Nom du test | 1.8 Connexion – Nom d’utilisateur trop court |
| Description | Se connecter avec un nom d’utilisateur de moins de 8 caractères |
| Comment le tester | Effectuer une requête HTTP Post avec un JSON contenant un username de moins de 8 caractères et password sur l’endpoint /signin |

|  |  |
| --- | --- |
| Nom du test | 1.9 Connexion – Mot de passe trop court |
| Description | Se connecter avec un mot de passse de moins de 8 caractères |
| Comment le tester | Effectuer une requête HTTP Post avec un JSON contenant un username et password de moins de 8 caractères sur l’endpoint /signin |

### Scénarios des tests

Ceci est un tableau récapitulatif de l’état des différents tests au fur et à mesure des journées.

X = signifie que le test est raté.

V = signifie que le test est passé.

# = signifie que le test est passé partiellement.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Test | J 1 | J 2 | J 3 | J 4 | J 5 | J 6 | J 7 | J 8 |
| 1.1 | X | X | V | V | V | V | V | V |
| 1.2 | X | X | X | X | # | # | # | # |
| 1.3 | X | X | X | X | # | # | # | # |
| 1.4 | X | X | X | X | # | # | # | # |
| 1.5 | X | X | X | X | # | # | # | # |
| 1.6 | X | X | X | X | # | # | # | # |
| 1.7 | X | X | X | X | # | # | # | # |
| 1.8 | X | X | X | X | # | # | # | # |
| 1.9 | X | X | X | X | # | # | # | # |

# Conclusion

### Difficultés rencontrées

Ce projet fut particulièrement compliqué étant donné que nous utilisons des technologies qui nous sont pour la plupart inconnue. Nous avons dû apprendre à les utiliser pour pouvoir réaliser à bien ce projet. Néanmoins, nous avons rencontré quelques difficultés. Pour le client CSharp, nous avons pris du temps à trouver le paquet NuGet de socket.io compatible avec la version du serveur. La difficulté de socket.io sur CSharp a notamment été sous-estimé. Pour le client web, nous avons rencontré un problème de connexion avec le socket.io du serveur et nous avons dû modifier certains paramètres du serveur afin de faire fonctionner la connexion du client web au serveur. Coté serveur, nous pensions que la réalisation de ce dernier prendrait moins de temps mais forcé de constater que ce n’est pas le cas, cela a ralenti tout le déroulement étant donné que le serveur est le cœur du projet et que les clients ne peuvent pas fonctionner sans ce dernier.

Pour finir, le covid a frappé notre équipe et un des membres n’a pas pu assister à la période de développement hebdomadaire retardant son travail d’une semaine.

### Améliorations possibles

Étant donné le court temps pour réaliser ce projet, nous avons dû faire des concessions. Avec plus de temps, certaines choses auraient pu être améliorées. En ce qui concerne le serveur Node, nous pourrions améliorer ses performances ainsi qu’optimiser son code. En ce qui concerne le client CSharp, il serait utile de remplacer « httpWebRequest » par « httpClient » car le httpWebRequest est obsolète. Pour le client web, un approfondissement du design pourrait être envisagé car le design actuel est trop simpliste. L’utilisation d’un Framework de vue pour la partie front-end est quelque chose qui peut être utile à ajouter. Pour finir, dans la partie gestion du projet, une meilleure organisation aurait sûrement facilité le bon déroulement du projet.

### Bilan Personnel

Leo

Je trouve que nous n’avions pas une très bonne organisation ce qui a beaucoup retardé le projet dans sa globalité. Nous n’avons donc pas pu faire tout ce que l’on souhaitait dans les délais. Néanmoins, ce projet m’a beaucoup appris sur le « cross platform » ainsi que les technologies que nous avons utilisées pour réaliser ce dernier. Il m’a aussi beaucoup appris côté documentation car j’ai en grande partie réaliser la documentation technique.

Hodheil

Avec ce projet, j'ai appris à faire communiquer un client CSharp avec un serveur via Socket.io (en asynchrone). Cela m'a aussi permis d'approfondir mes connaissances sur la construction d'une application CSharp, tant du côté maquette de l'application que de sa logique et son architecture. Pour ce qui du rendu final, je suis très fier de ce que nous avons fait avec mes camarades.

Hugo

Ce projet m’a permis de mettre en pratique toutes mes connaissances et par la même occasion d’en apprendre davantage sur les technologies utilisées. Je suis satisfait du travail que j’ai accompli et de celui de mes camarades.

Brian

Ce projet m’a beaucoup appris surtout en matière de socket.io. J’ai dû développer en grande partie le serveur et plus précisément la partie socket.io et je suis content de ce que j’ai pu réaliser lors de ce projet. Socket.io étant assez compliqué, j’ai su apprendre par moi-même afin de mettre le serveur en marche le plus vite possible pour le bon déroulement du projet.