PAINseau

Protocole de communication

Version 3.0

Historique des révisions

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Version** | **Description** | **Auteur** |
| 2021-02-07 | 1.0 | Rédaction initiale du protocole de communication | Équipe 107 |
| 2021-02-16 | 1.1 | Première révision en équipe du protocole de communication | Équipe 107 |
| 2021-02-18 | 1.2 | Version finale pour la réponse à l’appel d’offre | Équipe 107 |
| 2021-04-16 | 2.0 | Révision de l'architecture logicielle à la suite de la première correction | Équipe 107 |
| 2021-04-18 | 3.0 | Modifications finales pour la livraison du produit final | Équipe 107 |

Table des matières

[1. Introduction 1](#_Toc69755482)

[2. Communication client-serveur 1](#_Toc69755483)

[3. Description des paquets 2](#_Toc69755484)

[Communication REST API 2](#_Toc69755485)

[Communication des sockets 3](#_Toc69755486)

[Interfaces 4](#_Toc69755487)

[Énumérations 5](#_Toc69755488)

Protocole de communication

# 1. Introduction

Ce document a pour objectif de présenter les détails entourant la communication client-serveur afin de dresser une idée générale des flux de communication. En d’autres mots, ce présent document justifiera d’abord l’utilisation des technologies utilisées pour la communication client-serveur, tout en précisant les parties spécifiques de notre application qui y sont touchées. Une description des types de données transférées s’y retrouvera également. La section suivante contiendra ensuite une description du contenu des différents types de paquets utilisés au sein de notre protocole.

# 2. Communication client-serveur

Un premier type de communication entre le client et le serveur est les requêtes HTTP. Ces requêtes serviront principalement à la gestion des utilisateurs. Ainsi, elles permettront d’envoyer les informations nécessaires au serveur à partir du client pour enregistrer un nouvel utilisateur, pour se connecter, pour récupérer ou envoyer de l’information du profil d’un utilisateur. Cette communication est cependant unidirectionnelle et ne permet pas au serveur de communiquer au client. Le serveur sera hébergé sur la plateforme Heroku afin d'être accessible à travers Internet et donc on pourra réaliser des requêtes au serveur hébergé à travers n'importe quelle machine. Le code du côté serveur sera écrit en Typescript pour gérer les requêtes HTTP. Un port sera automatiquement assigné par Heroku pour gérer les requêtes. Une base de données MongoDB sera utilisée pour sauvegarder les informations. Cette dernière sera hébergée sur Atlas afin d'être accessible sur Internet. Les requêtes HTTP seront utilisées pour sauvegarder toutes les informations à propos des utilisateurs, pour créer et rejoindre un groupe et pour sauvegarder un dessin dans l’album de dessins d’un utilisateur.

Un deuxième type de communication utilisé pour notre application est celui des sockets. Ces derniers serviront principalement pour l’envoi, la réception et la gestion des messages en temps réel. Ils seront également utilisés dans les groupes afin d'informer les joueurs présents de l'arrivée ou encore du départ d'un joueur ainsi que pour joindre une équipe. Les sockets seront également utilisés dans les parties pour communiquer aux joueurs les informations sur la partie comme le début et la fin d'une partie ou d'une ronde, le droit de réplique et indiquer que le mot a été deviné. Les sockets sont initialisés dans les clients et le serveur. La librairie de socket.io sera utilisée pour implémenter cette partie de code du côté serveur ainsi que dans les deux clients. Heroku assignera des ports automatiquement pour gérer les émissions des sockets.

Lors de la connexion d’un utilisateur, il n’est pas nécessaire de fournir une adresse IP puisqu’une connexion automatique est effectuée.

# 3. Description des paquets

## Communication REST API

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Description | Méthode | Requête | Paramètres | Données retournées |
| Ajouter un usager | POST | /database/registerUserInfo | {user: IUserInfo} | {response: IResponse} |
| Recevoir la liste d’usagers | GET | /database/users | N/A | {users: IUserInfo[]} |
| Supprimer un utilisateur | DELETE | /database/delete-user | {user: IUserInfo} | {response: IResponse} |
| Recevoir informations à propos d’un usager | GET | /database/user | {username: string} | {user: IUserInfo} |
| Modifier un utilisateur | POST | /database/updateUserInfo | {user: IUserInfo} | {response: IResponse} |
| Connexion d’un usager | POST | /database/login | {user: IUserInfo, password: string} | {response: IResponse} |
| Déconnexion d’un usager | DELETE | /database/logout | {user: IUserInfo} | {response: IResponse} |
| Recevoir la liste des usagers connectés | GET | /database/connected | N/A | {users: IUserInfo[]} |
| Ajouter un dessin dans l’album de dessin | POST | /database/add-drawing | {drawing: IDrawingInfo, user: IUserInfo} | {response: IResponse} |
| Recevoir la liste de dessins de l’album de dessins d’un usager | GET | /database/drawings | {user: IUserInfo} | {drawings: IDrawingInfo[]} |
| Ajouter un dessin dans la banque de paire mot-image | POST | /database/add-wordImage | {pair: IWordImage} | {response: IResponse} |
| Recevoir la liste de dessins de la banque paire mot-image | GET | /database/wordImages | N/A | {drawings: IWordImage[]} |
| Recevoir l’historique d’un canal de discussion | GET | /database/room-history | {room: string} | {messages: IMessageContent[]} |
| Créer un groupe | POST | /database/insert-lobby | {lobby: ILobby} | {response: IResponse} |
| Recevoir la liste des groupes existant | GET | /database/all-lobbies | N/A | {lobbies: ILobby[]} |
| Vérifier le nombre de joueurs présents dans un groupe | POST | /database/lobby-users | {name: string} | {response: IResponse} |

## 

## Communication des sockets

|  |  |  |
| --- | --- | --- |
| Description | Événement envoyé | Données |
| Connexion d’un utilisateur à un canal de clavardage | user-joined | {username: string, room: string} |
| Déconnexion d’un utilisateur à un canal de clavardage | quit-room | {username: string, room: string} |
| Déconnexion d’un utilisateur | logout | {username: string} |
| Supprimer un canal de clavardage | delete-room | {room: string} |
| Échange de messages dans un canal de clavardage | send-message | {message: IMessageContent} |
| Rejoindre un groupe | join-lobby | {name: string, owner: string, difficulty: string, users: IPublic[], rounds: int} |
| Quitter un groupe | quit-lobby | {name: string, owner: string, difficulty: string, users: IPublic[], rounds: int} |
| Rejoindre une équipe dans un groupe | join-team | {playerName: string, playerAvatar: string, isTeam1: boolean, lobbyName: string} |
| Supprimer un groupe | delete-lobby | {lobbyName:string} |
| Envoyer les informations sur les équipes dans un groupe | teams-info | {teams: string[]} |
| Commencement d’une partie de jeu | start-game | {lobby: ILobby} |
| Arrêter une partie de jeu en cours | stop-game | {gameName: string} |
| Rejoindre une partie | join-game | {gameName: string} |
| Échange de traits de dessin | draw | {drawing: IDrawing} |
| Signalisation du début d’un tour | round-start | {gameName: string} |
| Signalisation de la fin d’un tour | round-end | {gameName: string} |
| Indiquer la fin d'une partie | game-end | {gameName: string} |
| Indiquer que le mot a été deviné | guess-word | {name: string, team: string, guesser: string} |
| Indiquer le début d'un droit de réplique | reply | {gameName: string} |
| Envoyer le résumé de la partie | game-info | {username: string, nbVictories: int, time: int, mode: string, score: int[], users: IPublic[], diff: string} |
| Demander un message d'un joueur virtuel | bot-msg | {lobbyName: string, state: int, sender: string} |
| Indique au serveur d’arrêter de dessiner l’image | stop-word | {gameName: string} |

## Interfaces

|  |  |
| --- | --- |
| IRoom | { name: string, history: IMessageContent[], admin: string } |
| IUserInfo | {public: {username: string, avatar: string, pointsXP: number, album: IDrawingInfo[], title: string, border: string}, private: {firstName: string, lastName: string, gameStats: IGameStats. connections: string[]}, connection: {username: string, password: string, socketId: string, isConnected: boolean, rooms: string[]} |
| IGameStats | {gamesPlayed: number, gamesWon: number, totalGameTime: number, allGames: IGameHistory[] |
| IGameHistory | {date: string, time: number, gameMode: string, scoreClassic: number[], usersPlayedWith: IuserInfo[“public”][], difficulty: string} |
| IDrawingInfo | {drawing: string, word: string} |
| ILobby | {name: string, owner: string, difficulty: string, users: IUserInfo[“public”], rounds: number} |
| IGame | {nbMatches: number, nbVictories: number, time: float,  date: string, users: any[], score: int[], mode: string} |
| IMessageContent | {message: string, sender: string, timestamp: string, room: string, avatar: string} |
| IDraw | {name: string, clientX: float, clientY: float, strokeWidth: int, isEraser: Boolean, type: string, color: string, isLight: Boolean, undo: Int, depth: Int, receiving: boolean } |
| IWordImage | {word: string, hints: string[]. difficulty: string[], mode: string, drawing: IPathDetails[]} |
| IPathDetails | {pathPoints: PathPoint[], color: string, size: number: depth: number} |
| PathPoint | {x: number, y: number} |
| IResponse | {status: Status, message: String} |

## Énumérations

|  |  |
| --- | --- |
| Status | {HTTP\_OK = 200,  HTTP\_CREATED = 201,  HTTP\_BAD\_REQUEST = 400,  HTTP\_NOT\_FOUND = 404,  HTTP\_INTERNAL\_SERVER\_ERROR = 500,  USER\_EXISTS = 0,  USER\_ALREADY\_CONNECTED = 1,  UPDATE\_OK = 2,  USER\_INEXISTENT = 3,  MAXIMUM\_USERS = 4,  LOBBY\_JOINED = 5} |