Projet “Fais-moi-un dessin”

Protocole de communication

Version 1.0

Historique des révisions

*[Dans l’historique de révision, il est nécessaire d’écrire le nom du ou des auteurs ayant travaillé sur chaque version.]*

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Version** | **Description** | **Auteur** |
| 2020-09-21 | 1.0 | Rédaction d’une partie de la communication client-serveur | Aubierge Gloria Sohou |
| 2020-09-27 | 1.0 | Mise à jour de la description paquets et des interfaces | Aubierge Gloria Sohou |
| 2020-09-29 | 1.0 | Révision sur des détails pour les subscription et mutation | Kevin Nguyen |
| 2020-09-30 | 1.0 | Révision et correction du document | Sophie Dorval |
| 20-10-02 | 1.0 |  | Aubierge Gloria Sohou |

Table des matières

[**1. Introduction**](#_heading=h.30j0zll) **4**

[**2. Communication client-serveur**](#_heading=h.1fob9te) **4**

[**3. Description des classes principales**](#_heading=h.3rdcrjn) **5**

[**4. Description des paquets**](#_heading=h.ithfgdmz6jd9) **9**

Protocole de communication

# 1. Introduction

Ce document décrit la manière dont les différentes entités de notre application “Fais-moi un dessin” communiquent entre elles. Tout d’abord, la section 2 décrit en détail notre choix de communication entre les clients android et PC avec le serveur. Ensuite, la section 4 présente les principales classes utilisées pour la communication entre les clients et le serveur. Enfin, la section 3 décrit les différents paquets utilisés le long de notre protocole de communication.

# 2. Communication client-serveur

Le bon fonctionnement de l’application “Fais-moi un dessin” nécessite trois grandes catégories de requêtes entre les clients et le serveur.

Tout d’abord, certaines fonctionnalités nécessitent une communication en temps réel entre le client et le serveur. À cet effet, on utilisera les sockets. Vu que le système communiquera grâce à un serveur NodeJS utilisant le langage GraphQL, on utilisera les **souscriptions de GraphQL** fonctionnant avec des **websockets**.

Les souscriptions sont une fonctionnalité de GraphQL permettant au serveur d'envoyer des données à ses clients lorsqu'un **événement spécifique** se produit. À cet effet, le serveur maintient une connexion permanente avec chaque **client souscripteur.** Chaque client souscripteur ouvre initialement une connexion de longue durée au serveur en envoyant une requête de souscription qui spécifie l'événement qui l'intéresse. Chaque fois que cet événement particulier se produit, le serveur utilise la connexion pour transmettre les données de l'événement à chaque client souscripteur.

Voici des **exemples d'événements** qui impliquent l'utilisation des websockets avec les souscriptions GraphQL:

* quand un joueur reçoit un message
* quand un joueur rejoint une partie
* quand un joueur quitte un partie
* quand un joueur rejoint un canal de discussion
* quand un joueur quitte un canal de discussion
* quand un nouveau canal de discussion est créé
* quand un canal de discussion est supprimé
* lorsqu’un joueur se connecte
* lorsqu’un joueur se déconnecte
* quand un joueur reçoit une invitation à un défi
* quand une partie commence
* quand une partie est terminée
* quand un joueur dessine lors d’une partie, il faudra une communication en temps réel avec le serveur afin que le dessin apparaisse sans interruption sur l’écran de tous les autres joueurs

Ensuite, on utilisera **les mutations de GraphQL** pour les opérations des clients qui nécessitent des modifications au niveau de la base de données. Il s’agit par exemple des opérations d’insertion, de mise à jour ou la suppression de données au niveau de la base de données. Voici une liste des fonctionnalités nécessitant l’utilisations des mutations de GraphQL:

* créer un compte
* connecter avec un compte existant
* envoyer un message
* rejoindre un channel
* quitter un channel
* modifier les statistiques d’un joueur
* créer une paire mot-image
* valider une tentative de réponse de jeu
* commencer un jeu
* envoyer les informations d’un dessin

Enfin, on utilisera **les queries (requêtes) de GraphQL** pour les opérations des clients ne nécessitant pas une communication en temps réel avec le serveur mais qui consistent à chercher des données au niveau de la base de données. Voici une liste de fonctionnalités nécessitant l’utilisations des ***queries*** de GraphQL:

* obtenir les informations publiques d’un joueur
* obtenir la liste des joueurs en ligne
* obtenir la liste des canaux de discussion
* obtenir les informations sur un canal de discussion
* obtenir les informations sur un jeu
* obtenir les informations du leaderboard
* obtenir les statistiques d’un joueur

Par ailleurs, les données échangées entre le clients et le serveur seront de format JSON.

# 3. Description des classes principales

Cette section présente la structure des différents objets qui seront échangés lors des différentes communications entre le serveur et les clients.

* **User:** classe pour modéliser un utilisateur.

|  |  |
| --- | --- |
| **Attribut** | **Type** |
| id | number |
| name | string |
| avatar | string |
| password | string |
| statistics | UserStatistic |
| loginHistory | LoginInfo[] |

*Tableau 4.1 : User*

* **UserStatistic***:* classe pour modéliser les différentes statistiques d’un utilisateur.

|  |  |
| --- | --- |
| **Attribut** | **Type** |
| nGames | number |
| winPercent | double |
| totalGameTime | number |
| playTimeAverage | double |
| bestSprintSoloScore | number |

*Tableau 4.2: UserStatistic*

* **LoginInfo**: classe pour modéliser les informations d’ **une** connexion d’un utilisateur à l’application.

|  |  |
| --- | --- |
| **Attribut** | **Type** |
| loginDate | datetime |
| logoutDate | datetime |
| playedGameHistoric | GameInfo[] |

*Tableau 4.3: Login*

* **GameInfo:** classe pour modéliser les informations d'une **partie jouée** par un utilisateur.

|  |  |
| --- | --- |
| **Attribut** | **Type** |
| gameMode | string |
| players | string[] |
| winners | string[] |
| bestScore | number |

*Tableau 4.4: GameInfo*

* **Message**: classe pour modéliser un message.

|  |  |
| --- | --- |
| **Attribut** | **Type** |
| id | number |
| senderId | number |
| content | string |
| createdAt | datetime |

*Tableau 4.5: Message*

* **Channel**: classe pour modéliser un canal de discussion.

|  |  |
| --- | --- |
| **Attribut** | **Type** |
| id | number |
| name | string |
| createdAt | datetime |
| messages | Message[] |
| users | User[] |

*Tableau 4.6 : Channel*

* **GameSession**: classe pour modéliser une session de jeu ou encore une partie de jeu.

|  |  |
| --- | --- |
| **Attribut** | **Type** |
| id | number |
| players | User[] |
| mode | string |
| difficultyLevel | number |
| createdAt | datetime |
| gameHall | Channel |
| pairsWordDrawing | pairWordDrawing[] |
| minPlayers | number |
| getTimeLeft | function: number |
| getScore | function: [{userId: score}] |
| status | * enum * Peut être STARTED, ENDED ou PENDIND |

*Tableau 4.7 : GameSession*

* **PairWordDrawing**: classe pour modéliser une paire mot-image

|  |  |
| --- | --- |
| **Attribut** | **Type** |
| id | number |
| difficultyLevel | number |
| word | string |
| drawing | ManualDrawing |
| hints | string[] |

*Tableau 4.8 : PairWordDrawing*

* **ManualDrawing**: classe pour modéliser un dessin fait manuellement dans la zone de dessin par un utilisateur non virtuel.

|  |  |
| --- | --- |
| **Attribut** | **Type** |
| id | number |
| mode | string |
| paths | Path[] |
| screenWidth | number |
| ScreenHeight | number |
| commands | Command[] |

*Tableau 4.9: ManualDrawing*

* **Command**: classe pour modéliser une action lorsqu’un joueur dessine. Une commande est envoyée au serveur lorsque le joueur qui dessine “lève le crayon”.

|  |  |
| --- | --- |
| **Attribut** | **Type** |
| commanId | number |
| drawingId | number |
| pathId | number |
| undo | (fonction anonyme) |
| redo | (fonction anonyme) |

*Tableau 4.10: Command*

* **Path**: classe pour modéliser un chemin dans un dessin SVG. Un objet path est régulièrement envoyé au serveur lorsque le joueur dessine.

|  |  |
| --- | --- |
| **Attribut** | **Type** |
| pathId | number |
| drawingId | number |
| data | string (coordonnées) |
| strokeColor | string |
| strokeWidth | number |
| pathStatus | enum peut être BEGIN, END, ONGOING |

*Tableau 4.11: Path*

* **SprintGameSession:** classe dérivée de GameSession pour modéliser une session de jeu avec le mode sprint solo ou coopératif.

|  |  |
| --- | --- |
| **Attribut** | **Type** |
| leftTime | number |
| nLeftTries | number |
| bonus Time | SoloBonusTimeEnum |

*Tableau 4.12 : SprintGameSession*

* **SoloBonusTimeEnum:** Structure pour modéliser les différents socores associés à chaque niveau de difficulté.

|  |  |
| --- | --- |
| **Attribut** | **Type** |
| EASY | number |
| MEDIUM | number |
| HARD | number |

*Tableau 4.13 : SoloBonusTimeEnum*

* **FreeForAllGameSession:** classe dérivée de GameSession pour modéliser une session de jeu avec le mode free-For-All.

|  |  |
| --- | --- |
| **Attribut** | **Type** |
| nWordsMax | number |
| timePerWord | number |

*Tableau 4.14 : FreeForAllGameSession*

* **VirtualDrawing**: classe pour modéliser un dessin fait par un joueur virtuel

|  |  |
| --- | --- |
| **Attribut** | **Type** |
| id | number |
| mode | string |
| data | BitMap |

*Tableau 4.15: VirtualDrawing*

* **Invitation**: classe pour modéliser une invitation **à un challenge** ou **d’amitié**

|  |  |
| --- | --- |
| Attribut | Type |
| id | number |
| senderId | number |
| receiverId | number |
| isAccepted | number |

*Tableau 4.16: Invitation*

* **GameChallengeInvitation**: classe dérivée de la classe Invitation pour modéliser une invitation à un challenge

|  |  |
| --- | --- |
| Attribut | Type |
| invitation | Invitation |
| gameSessionId | number |

*Tableau 4.17: GameChallengeInvitation*

# 

# 4. Description des paquets

Cette section détaille le contenu des différents types de paquets utilisés au sein du protocole de communication.

**Paquets utilisés lors des communications avec les souscriptions:**

Chaque client peut souscrire à chacune des opérations citées dans le tableau ci-dessous. Dès qu’un événement survient, le serveur envoie à tous les clients souscripteurs, les données présentées dans la dernière colonne du tableau. Un client peut aussi annuler sa souscription à une opération. Ainsi, il ne recevra plus informé dès que l'événement survient.

|  |  |  |  |
| --- | --- | --- | --- |
| Événement | Nom de la souscription | Paramètre(s) de de souscription | Donnée(s) reçue(s) par un client souscripteur lorsque l'événement survient |
| un joueur rejoint une partie | onPlayerJoinGame | gameSessionId | GameSession |
| un joueur quitte une partie | onPlayerLeftGame | gameSessionId | GameSession |
| un nouveau canal de discussion est créé | onNewChannel | Aucune | Channel |
| un canal de discussion est supprimé | onChannelDelete | channelId | Channel |
| un joueur rejoint un canal de discussion | onChannelChange | channelId | Channel |
| un joueur quitte un canal de discussion | onChannelChange | channelId | Channel |
| un joueur reçoit un message | onNewMessage | channelId | Message, ChannelId |
| un joueur reçoit une invitation à un défi à un joueur | onNewChallenge | userId | GameChallengeInvitation |
| un joueur reçoit une invitation d'amitié | onNewFriend | userId | Invitation |
| une nouvelle partie a commencé | onGameStarted | gameSessionId | GameSession |
| Assigner le rôle de dessinateur à un joueur | onNewDrawer | gameSessionId | drawerId |
| un joueur dessine | onNewPath | gameSessionId | Path |
| un joueur qui dessine annule sa dernière action | undo | Aucune | Command(Action annulée) |
| un joueur qui dessine refait sa dernière action | redo | Aucune | Command(Action refaite) |

**Paquets utilisés lors des requêtes mutations et queries de GraphQL**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Demande | Opération | Type | Données envoyées | Données de retour |
| Créer un compte d’utilisateur | register | mutation | username,  password | User ou Error |
| Se connecter | login | mutation | username,  password | User ou Error |
| Mettre à jour les statistiques d’un joueur | setStatistic | mutation | UserStatistic | User |
| Statistiques d’un joueur | userStatistic | query | userId | UserStatistic |
| Liste des canaux | channels | query | Aucune | Channel[] |
| Un canal | channel | query | channelId | Channel |
| Ajouter un canal | createChannel | mutation | channelName | Channel |
| Ajouter un utilisateur à un canal | enterChannel | mutation | ChannelId, userId | Channel |
| Quitter un channel | exitChannel | mutation | ChannelId, userId | Channel |
| Envoyer un message | addMessage | mutation | Message, channelId | Message |
| Créer une paire mot- image | createPairWordDrawing | mutation | PairWordDrawing | PairWordDrawing |
| Valider une tentative de réponse de jeu | validateWord | mutation | word | Boolean |
| Commencer un jeu | startGame | mutation | userId,  difficulty,  gameMode | GameSession |