Master 1 GIL - Document technique - partie Plateforme : Description de la bibliothèque de code

Groupe 1 Agora V3-1 : Partie plateforme Web

19 Mai 2024

Table des matières

1 Introduction

Ce document a pour objectif de présenter les fonctionnalités non implémentées de la plateforme web AGORA, ainsi que des pistes d'implémentation pour les prochaines itérations du projet. Ces fonctionnalités ont soit été implémentées partiellement, soit seulement envisagées et abandonnées faute de temps. Il est donc possible dans le code source du projet des traces de ces tentatives d'implémentations, auquel cas ce document tentera au maximum d'en faire référence pour guider au mieux les prochains groupes.

Pour les fonctionnalités qui n'en sont restées qu'à la phase de conception, ce document proposera des propositions de mise en œuvre pour chacune d'entre elles. Ces pistes d'implémentation ont pour but de guider les prochains groupes de travail dans leur réflexion et leur développement de l'application.

Nous espérons que ce document contribuera à l'amélioration continue d'AGORA et à son succès auprès de ses utilisateurs.

2 Expiration de la notification d'une invitation

Lorsque un joueur invite un autre joueur à une table lors de sa création, une notification d'invitation est envoyée au joueur concernée. Comme une invitation à une table doit durer 1 semaine avant d'expirer, il était souhaité par le client que la notification de cette invitation soit supprimer (ou bien remplacée par une autre notification indiquant cette expiration), afin d'éviter que celui-ci utilise l'invitation alors qu'elle n'est plus valide. Actuellement, la notification d'invitation reste affichée même après l'expiration, mais la route associée au lien de l'invitation, gérée par le BoardControlleur, vérifie toutefois la validité de l'invitation, et renvoie vers la page d'accueil de la dashboard avec un message flash d'erreur.

La solution qui était envisagé était l'utilisation du composant Messenger de Symfony (installable avec composer avec la commande suivant : "composer require symfony/messenger"). L'idée était d'envoyer un objet Message (via une classe crée pour l'occasion) à la création de la table contenant l'ID de la table, et se servir d'une classe MessageHandler qui va réceptionner ce type de message, récupérer la table via son ID, afin de faire appel à une fonction vérifiant s'il reste encore des joueurs invités qui n'ont pas rejoint celle-ci, afin de les notifier de l'expiration de l'invitation. Afin que cette vérification se fasse 1 semaine après, le composant Messenger propose une classe Stamp (enveloppe) où il est possible d'applique un délai d'envoi du message, il est donc possible d'envoyer le message une semaine précisémment après l'invitation. Cette solution avait été validé par le référent technique de la première itération du projet M. Stéphane BURESI.

Les informations sur le composant sont disponible à ce lien : https ://symfony.com/doc/current/messenger.html.

Une autre solution serait l'utilisation de cron jobs, pour demander au serveur de vérifier de manière quotidienne (par exemple tout les matins) l'intégrité des invitations de chaque Board, mais cette solution peut s'avérer être une opération trop lourde si le nombre de Board est élevé, alors que l'on souhaiterait seulement que les Boards se vérifient eux-mêmes.

3 Exclure un joueur d'une partie

Il doit être evidemment possible d'exclure un joueur d'une partie en cours dans une table, mais cela dans deux contextes différents : si un modérateur ou l'administrateur décide de l'exclusion pour un quelconque motif, ou si le joueur à exclure dépasse le temps d'inactivité de la table, au quel cas l'exclusion peut être décidé par l'un des joueurs de la table (un bouton "Exclusion" sera proposé sur la vue du jeu).

Pour le premier cas, il suffit simplement de définir une nouvelle route dans l'AdminControlleur qui appelle un service (le BoardManagerService peut être une bonne cible d'appel) qui va procéder à l'exclusion du joueur. Il est nécessaire, dans la fonction d'exclusion, de faire appel aussi à la fonction d'exclusion *excludePlayer()* du GameManagerService (présent dans le package Service/Game) afin de réaliser l'exclusion côté jeu et de le

mettre en pause en attendant un nouveau joueur. Il faudra ensuite définir la table à l'état "PAUSED", afin de permettre l'affichage de cette table dans la liste des tables disponibles, mais comme une table en pause. De la, le bouton "Rejoindre" habituelle associé à chaque table disponible appellera lui une route différente (il faudra donc créer cette route dans le BoardController), qui va appeler la fonction *replacePlayer()* du GameManagerService afin d'effectuer le remplacement du joueur, ajouter le joueur à la table puis redéfinir la table comme "IN_GAME". Il faudra aussi envisager un attribut "blacklist" pour l'entité Board, qui sera une collection d'entités User correspondant aux joueurs exclus de la table, pour éviter qu'ils la rejoigne à nouveau.

Pour le deuxième cas d'exclusion, le processus d'exclusion peut s'effectuer de la même manière que décrit précédemment. Toutefois pour gérer l'affichage de ce bouton (qui doit seulement se faire après avoir dépasser le temps d'inactivité), un attribut "inactivityTimer" et un attribut "inactivityHours" sont déjà présents dans l'entité Board. Une solution possible serait donc, à chaque coup réalisé, de rafraichir l'attribut inactivityTimer en instanciant un nouvel objet DateTime, auquel on ajoute le nombres d'heures contenus dans inactivityHours, afin d'obtenir la prochaine date (et heure) limite pour le coup du prochain joueur. Donc lors de chaque chargement de la vue du jeu, il serait possible de récupérer la date et l'heure actuelle, et de vérifier que ceux-ci ne dépasse pas l'inactivityTimer. Si c'est le cas, on affiche le bouton d'exclusion pour les joueurs.

4 Suppression d'une table

Il doit être possible pour un modérateur ou un administrateur de pouvoir supprimer une table de jeu (peut importe la raison invoquée). Une solution serait donc de prévoir une fonction côté jeu pour supprimer la partie en cours (il n'y a pas d'intérêt à conserver les données d'une partie qui ne s'est pas fini, on peut donc la supprimer de la base de données), et d'en faire appel dans une méthode de service (par exemple le BoardManagerService) qui va supprimer la partie, mais aussi supprimer la table et notifier chaque joueur de la suppression de celle-ci (via le NotificationService).

5 Modifier les droits d'un modérateur

Afin de pouvoir, pour l'administrateur, de modifier les droits d'un modérateur, une solution serait de créer une nouvelle entité ModeratorRight, avec un attribut booléen pour chaque droit possible. Il suffira ensuite d'associer cette entité avec l'entité User du modérateur, et de créer un formulaire qui va enregistrer les nouveaux droits ou ceux supprimer, et donc modifier après soummission du formulaire l'entité ModeratorRights associée au modérateur.

Il faudra ensuite bien veiller, du côté front de l'application, que lorsque l'on affiche l'une des options de modérations pour un modératuer (par exemple la supression d'une table), de vérifier d'abord que le modérateur ait bien accès à ce droit, avant de procéder à l'affichage.

6 Bannir un joueur

Il doit être possible pour un modérateur ou un administrateur de pouvoir bannir un joueur, selon une raison spécifiée dans le formulaire de bannissement. Lors du bannissement, il est nécessaire d'abord d'exclure le joueur de chacune des tables dans laquelle il était présent, en utilisant la solution expliquée à la partie `3 - Exclure un joueur d'une partie` de ce document, puis de l'ajouter à une table "banned_user" en y insérant son id, la raison du ban et la date limite avant son débannissement.

Lors d'une connexion d'un utilisateur, il suffira simplement de vérifier qu'il n'est pas présent dans la table "banned_user" pour lui donner accès à son compte. Si il est présent dans cette table et que la date de bannissement n'a pas encore expiré, on pourra le rediriger vers une page d'erreur, mais si la date est expiré, on peut dans ce cas lui donnès accès au site et en même temps supprimer (ou archiver) son bannissement de la table banned_user.

Pour éviter qu'un joueur qui se fait bannir alors qu'il était connecté puisse toujours avoir accès au site tant que sa session n'a pas expiré, une solution possible serait de créer un UserProvider personnalisé pour les entités User. En spécifiant ce provider dans le fichier config/services/security.yaml, le composant Security de l'application passera par ce nouveau provider lors de la récupération des données de notre user dans la session. Il suffira ensuite d'implémenter la fonction refreshUser dans notre UserProvider et de vérifier si l'user est banni, auquel cas on pourra le rediriger vers une page indiquant son bannissement, et le déconnecter du site.

Pour plus d'informations sur cette solution, l'URL de la documentation Symfony sur les UserProvider est le suivant : https ://symfony.com/doc/current/security/user_providers.html.