HEIG VD HAUTE ÉCOLE D'INGÉNIERIE ET DE GESTION DU CANTON DE VAUD

Département : TIC

Filière : Informatique et systèmes de communication

Orientation : Informatique logicielle

Travail de Bachelor

Documentation Technique

Non confidentiel

Étudiant : Quentin Forestier

Travail proposé par : Pier Donini

HEIG-VD

Route de Cheseaux 1

1400 Yverdon-les-Bains

Enseignant responsable : Pier Donini

Année académique : 2021-2022

Yverdon-les-Bains, le 27 juillet 2022

Table des matières

1 Login / Register

Le login et le register se font de manière synchrone. Le formulaire est envoyé, et le serveur renvoie une nouvelle page. Twirl et Play ayant un système de gestion d'erreurs pour les formulaires, c'était la façon la plus simple d'arriver à rendre un feedback précis à l'utilisateur.

Les conditions sur chaque input sont gérées grâce à Twirl et Play, tandis que l'unicité d'une adresse email est gérée à la main, en questionnant la base de données.

1.1 Sécurités et validations mises en place

Le mot de passe de l'utilisateur est hashé et salé grâce à l'utilitaire BCrypt. [1]

Validation des entrées utilisateurs :

- Email : L'email est validé à l'inscription en utilisant le Constraint.Email de Play
- Mot de passe : Le mot de passe est validé à l'inscription avec un regex (8 caractères, un chiffre, une majuscule, une minuscule et un caractère spécial)
- Nom : Le nom est valide s'il a plus de 2 caractères
- Confirmation du mot de passe : Vérification que le mot de passe est bien égal à la confirmation

Toutes ces entrées utilisateurs sont requises, autant pour l'authentification que pour l'enregistrement d'un utilisateur.

[1] https://www.mindrot.org/projects/jBCrypt/

1.2 Diagramme de séquences

1.2.1 Login

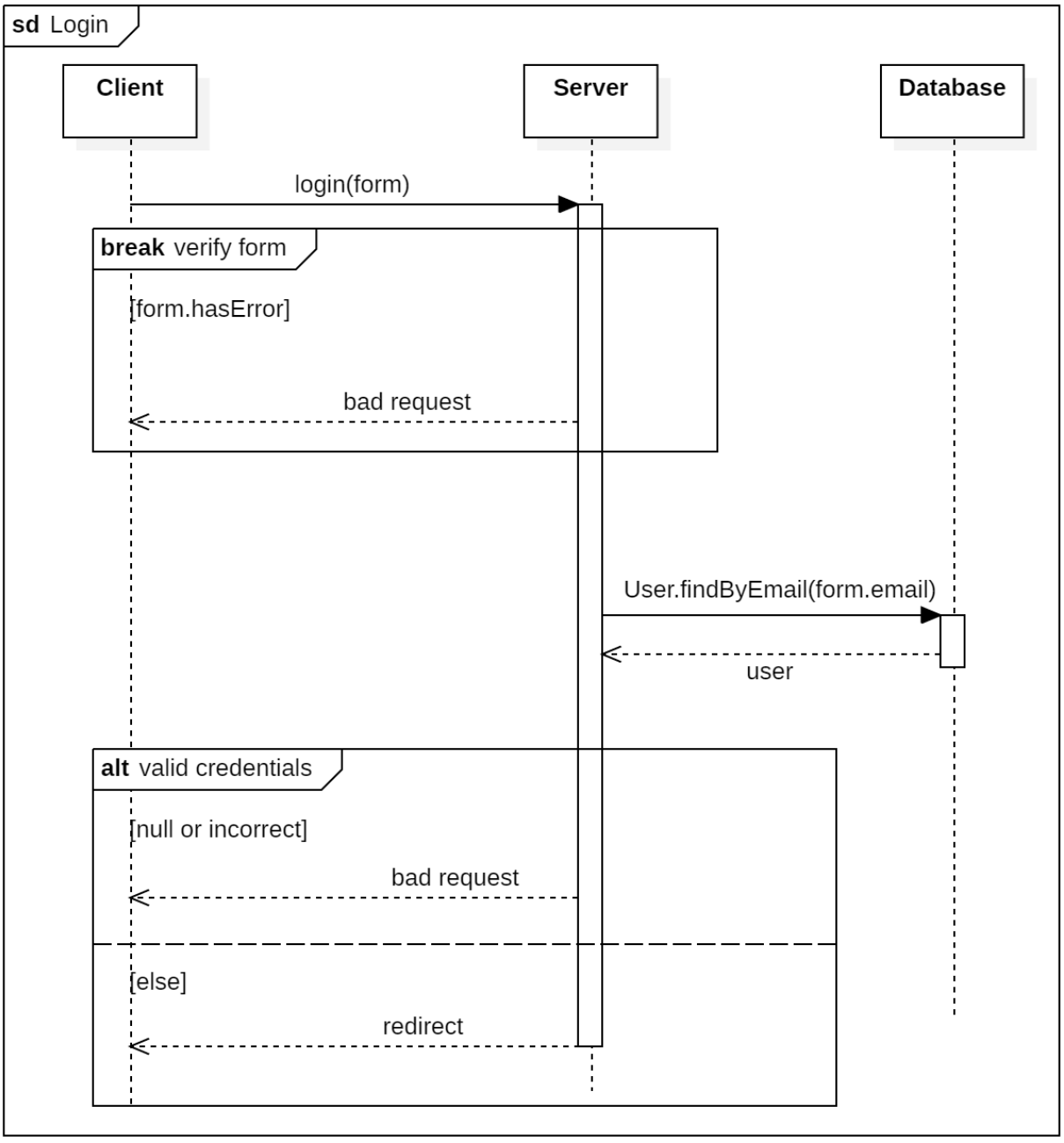

Figure 1 Diagramme de séquence de l'authentification

Le paramètre « form » représente le formulaire envoyé, avec tous les champs nécessaires à l'authentification.

1.2.2 Register

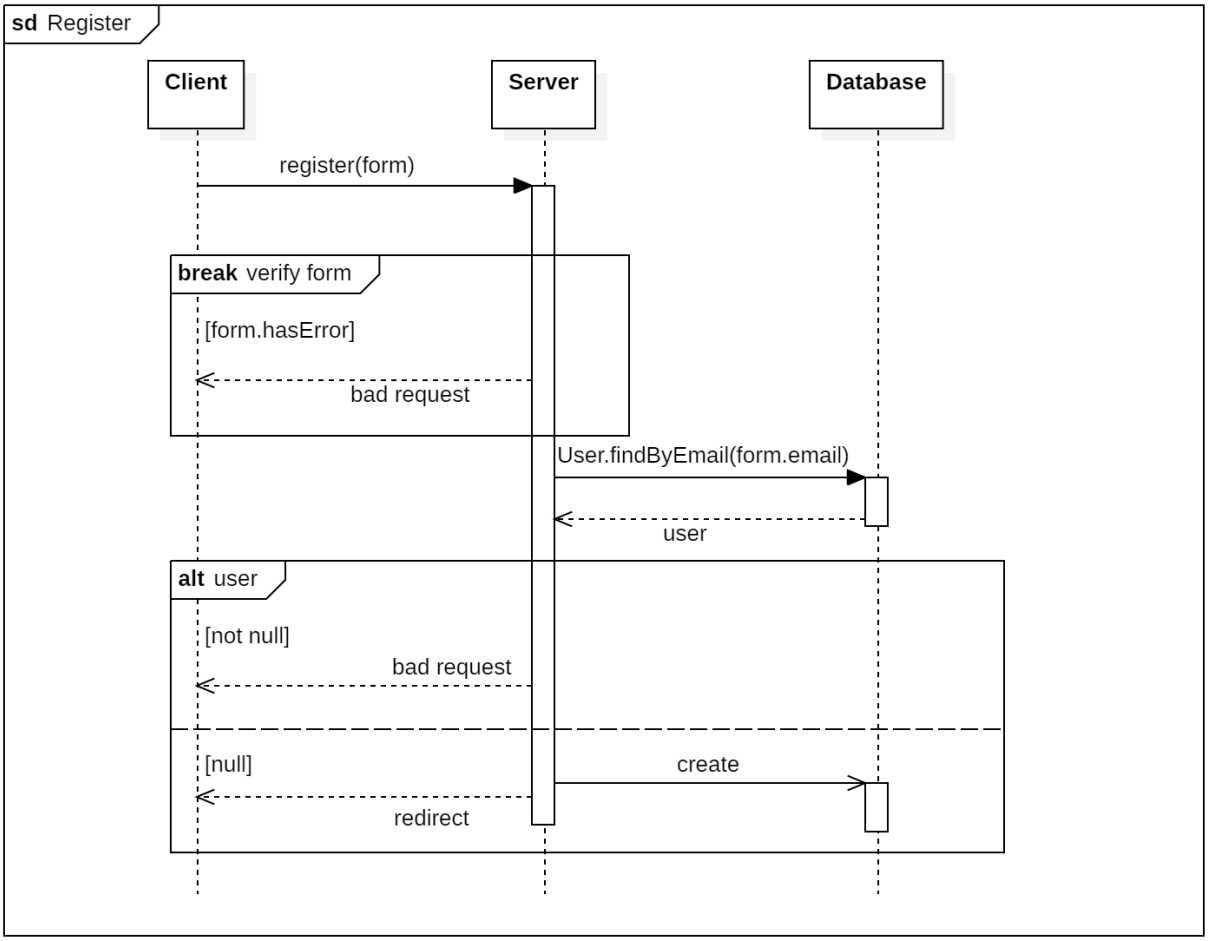

Figure 2 Diagramme de séquence de l'inscription

Le paramètre « form » représente le formulaire envoyé, avec tous les champs nécessaires à l'authentification.

2 Gestion de projets

Toutes les routes concernant les projets sont protégées. Il faut être authentifié afin d'avoir accès à la route. La vérification d'authentification n'est donc pas mise sur les diagrammes qui suivent afin de les alléger.

2.1 Projet

Afin de favoriser l'expérience utilisateur, toutes les interactions de gestion de projets sont faites en asynchrones, depuis du code JavaScript présent dans les vues. Cela permet de ne pas recharger la page à chaque modification effectuée.

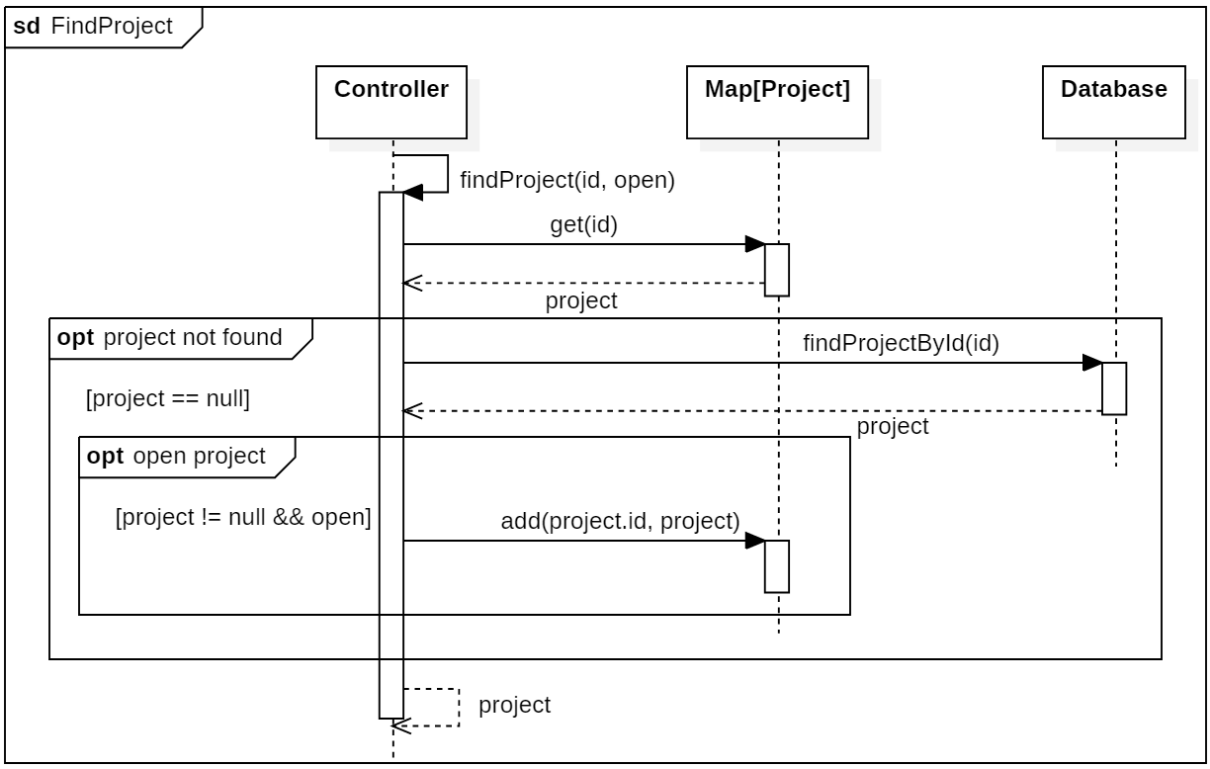

Figure 3 Diagramme de séquence de la recherche/ouverture d'un diagramme

Une map de projets ouverts permet d'effectuer les modifications sur le projet correspondant et que la modification se répercute pour tous les utilisateurs en instantané. Ceci est tout d'abord utilisé pour le websocket, mais afin de limiter les appels à la base de données, cette liste est également utilisée pour les autres fonctionnalités.

Un projet n'étant pas déjà ouvert, mais existant dans la base de données peut être ajouté à la map ou non en fonction d'un paramètre. Le diagramme du projet est également chargé lors de son ouverture, et donc cela est fait une unique fois.

Afin d'arriver à cette solution, je me suis inspiré d'un tutoriel qui utilise les Akka Actor pour travailler avec les websockets, afin de créer un serveur de messagerie en Scala.[2]

De plus, lorsqu'un projet s'ouvre, un timer est lancé afin d'effectuer des actions à un certain intervalle. Ces interactions consistent à :

- Sauvegarder le diagramme dans un fichier XML
- Vérifier si des utilisateurs sont connectés
    - Si aucun utilisateur n'est connecté, le projet se sauvegarde et se ferme.

[2] https://medium.com/@nnnsadeh/building-a-reactive-distributed-messaging-server-in-scala-and-akka-with-websockets-c70440c494e3

2.1.1 Création

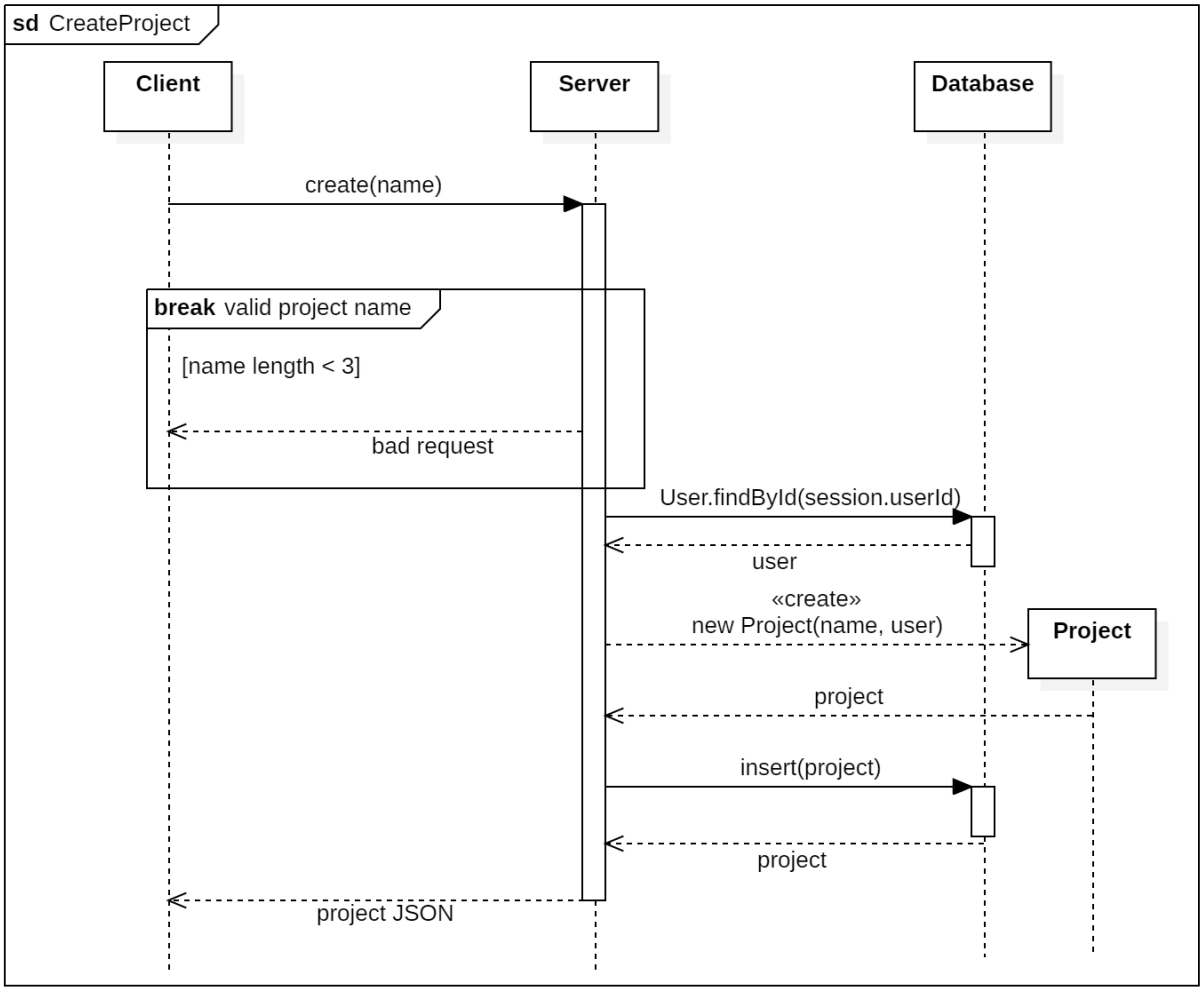

Figure 4 : Diagramme de séquence de la création de projets

Une validation est effectuée afin que le nom d'un projet ne soit pas plus petit que 3 caractères.

L'utilisateur authentifié est alors ajouté en tant que propriétaire du projet avec tous les droits.

2.1.2 Mise à jour

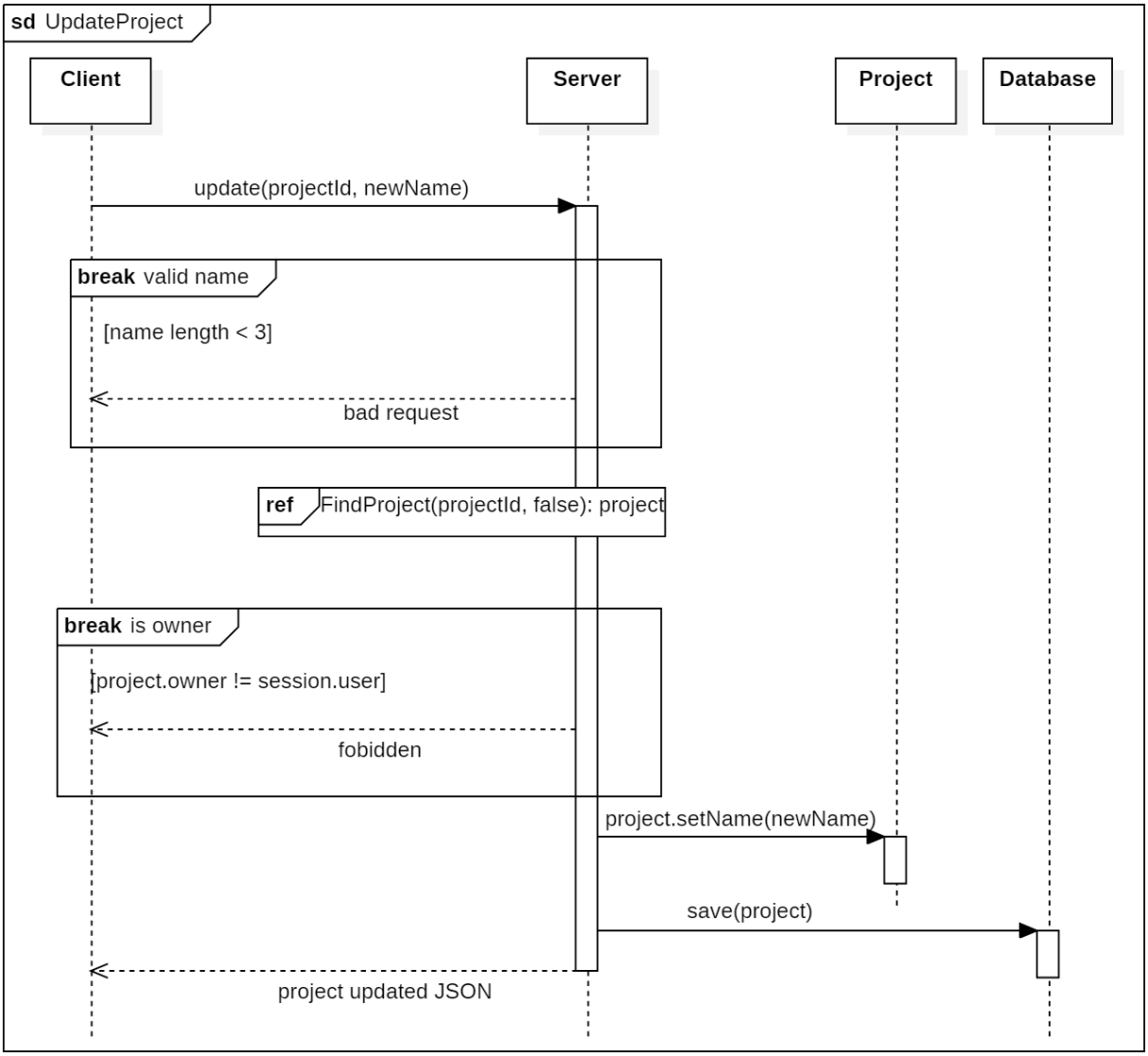

Figure 5 : Diagramme de séquence de la mise à jour d'un projet

On vérifie que le nom est correct, et que le projet a modifié appartient bien à l'utilisateur connecté avant de le modifier.

2.1.3 Suppression

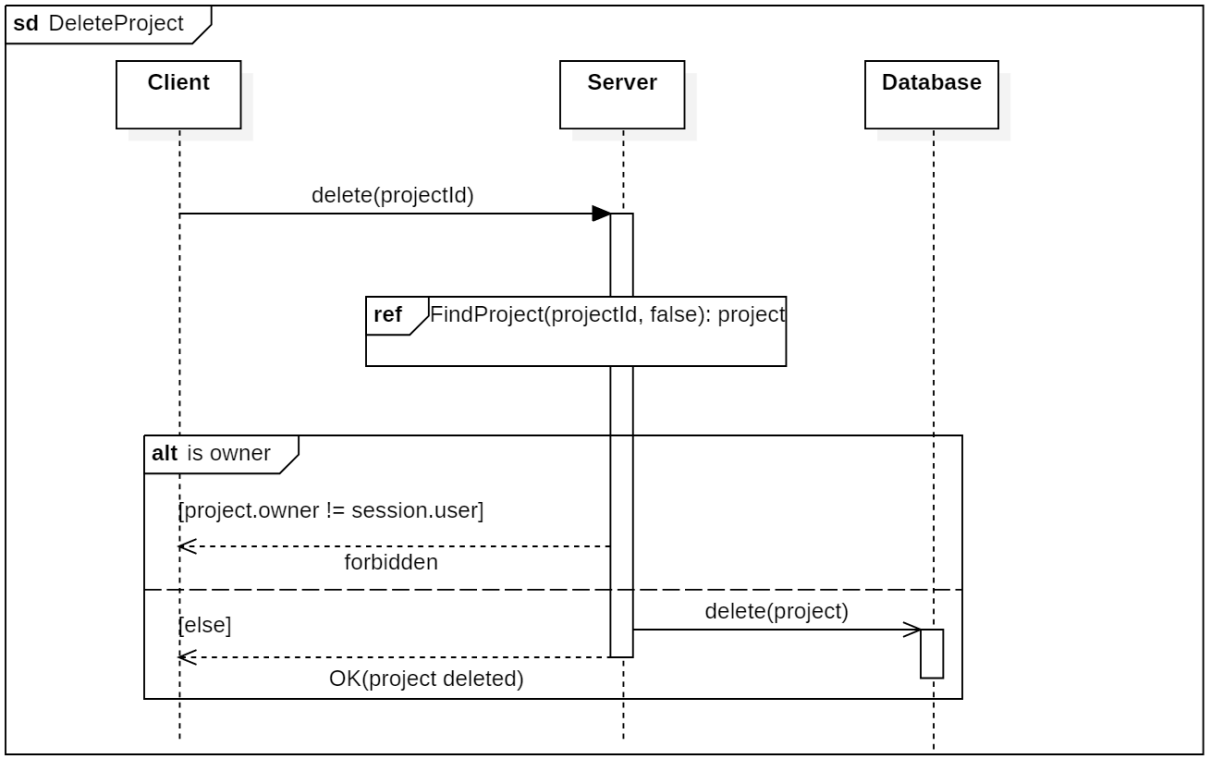

Figure 6 : Diagramme de séquence de la suppression d'un projet

On vérifie que le projet appartient bien à l'utilisateur connecté avant de le supprimer.

2.1.4 Récupération des projets d'un membre

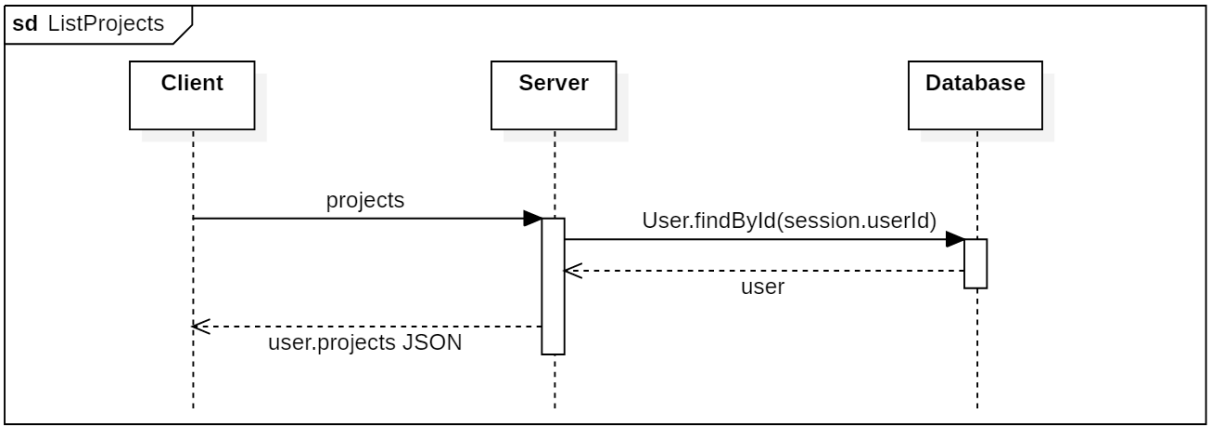

Figure 7 : Diagramme de séquence de la récupération des projets de l'utilisateur connecté

2.1.5 Récupération d'un projet spécifique et accès à la page du projet

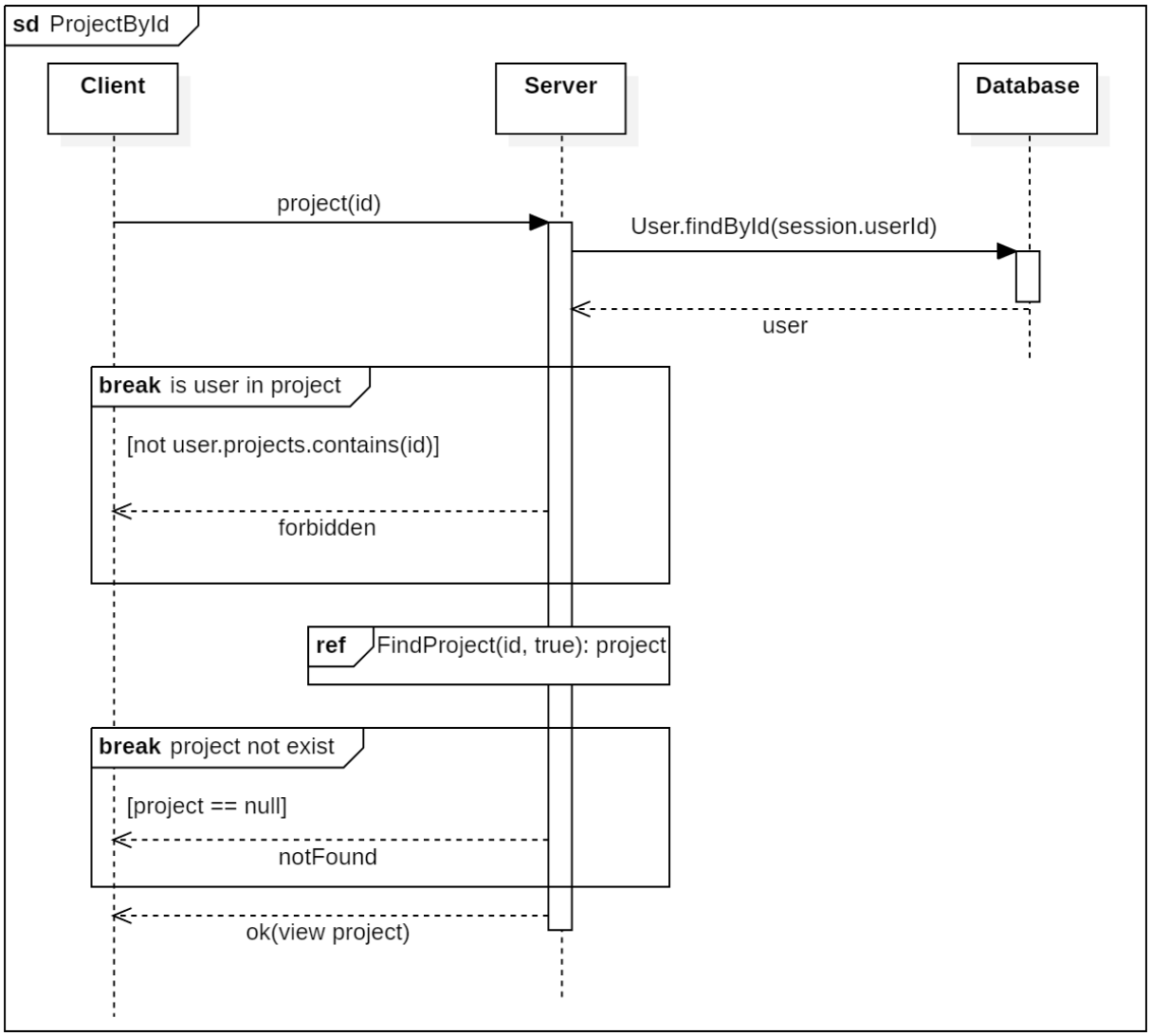

Figure 8 : Diagramme de séquence de la récupération d'un projet spécifique et de sa vue

On vérifie que l'utilisateur connecté appartient bien au projet. Si le projet n'est pas dans la map des projets ouverts, il y est ajouté.

2.2 Collaborateurs

2.2.1 Ajout d'un collaborateur

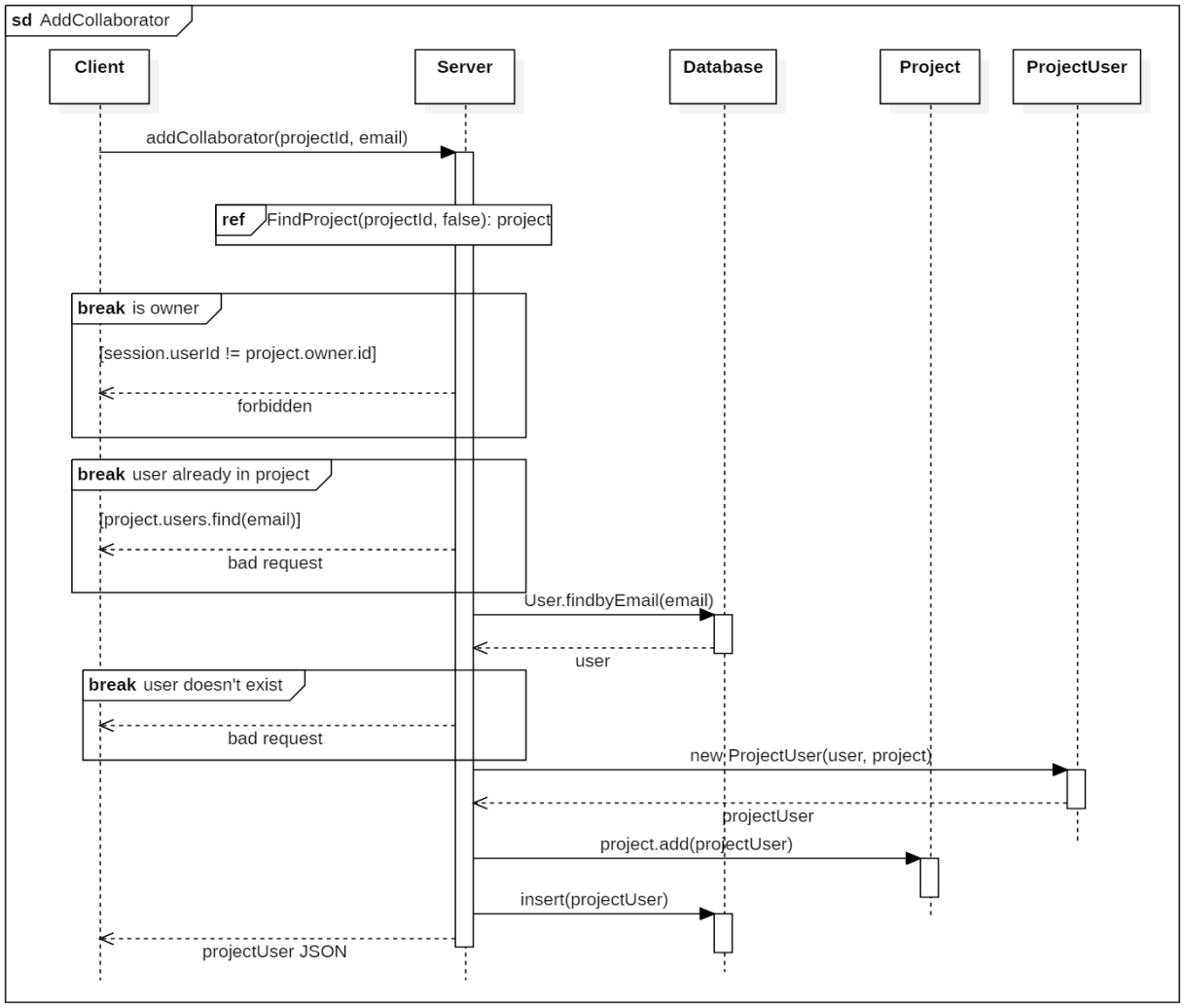

Figure 9 : Diagramme de séquence de l'ajout d'un collaborateur

Seul le propriétaire d'un projet peut ajouter un collaborateur.

Un collaborateur déjà dans le projet ne peut pas être ajouté une seconde fois.

2.2.2 Modification des droits d'un utilisateur

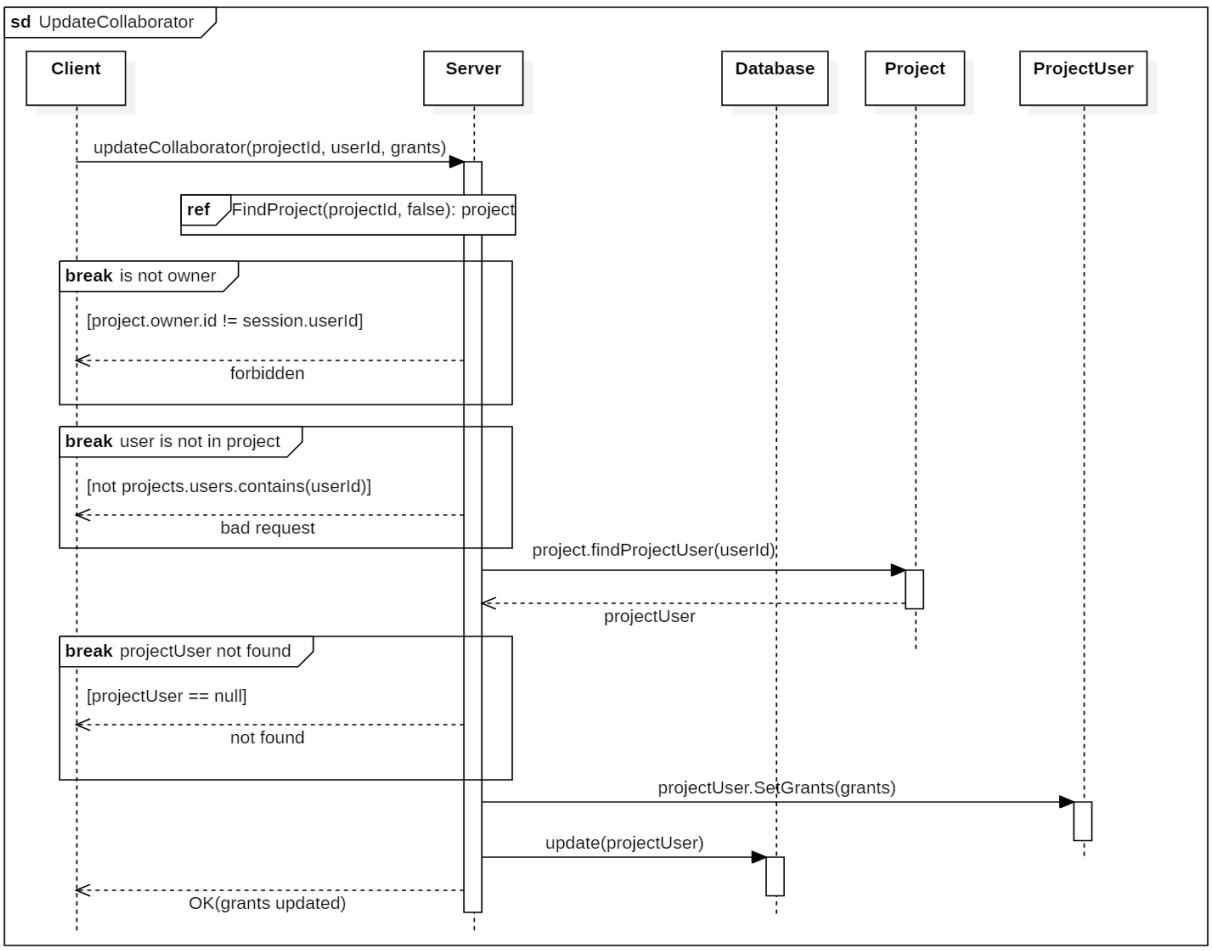

Figure 10 : Diagramme de séquence de la modification des droits d'un utilisateur

On vérifie que l'utilisateur connecté est bien le propriétaire, et que l'utilisateur a modifié appartient bien au projet.

2.2.3 Suppression d'un collaborateur

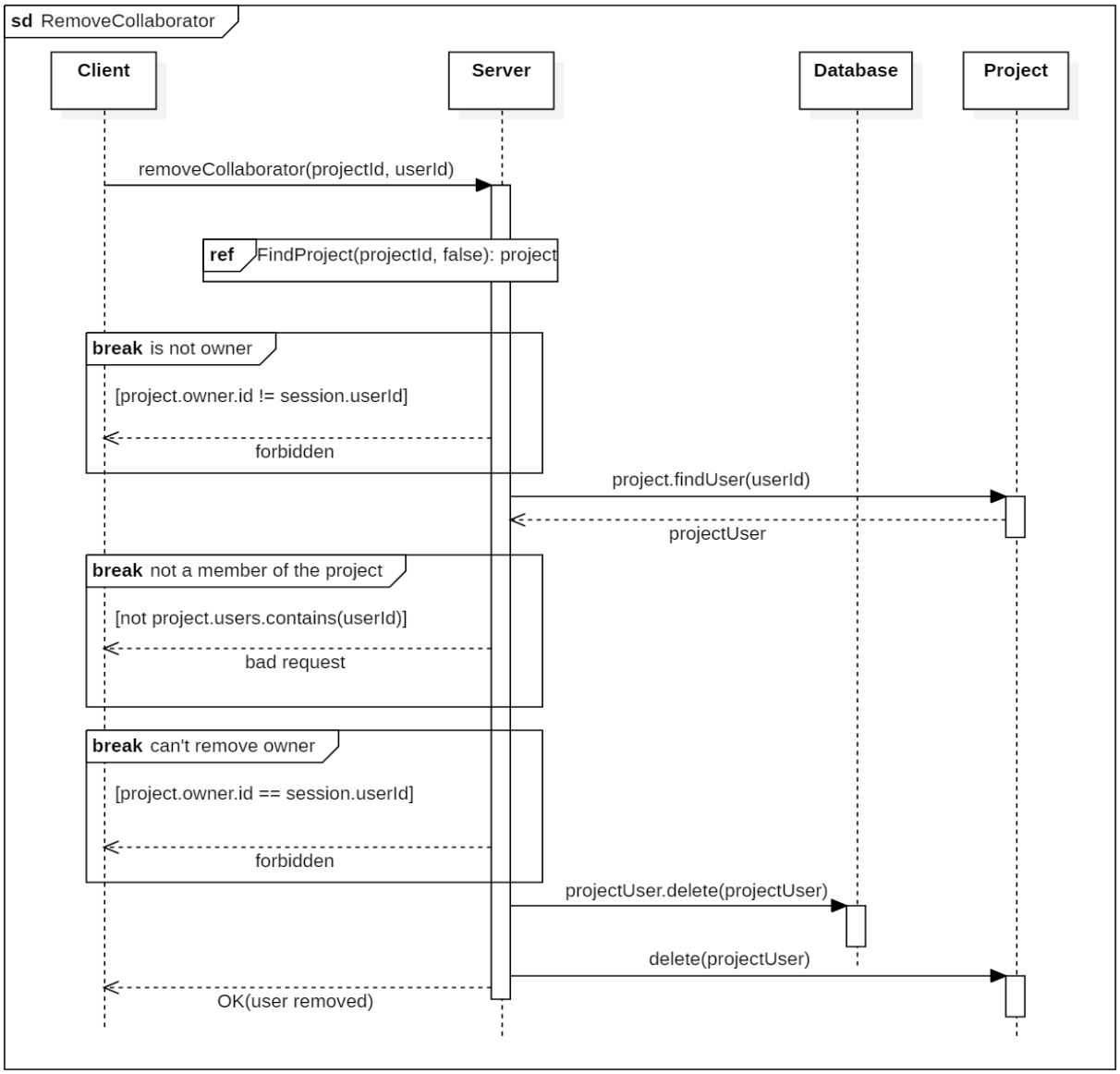

Figure 11 : Diagramme de séquence de la suppression d'un collaborateur

On vérifie que l'utilisateur connecté est le propriétaire du projet, et que le collaborateur a supprimé est bien un membre du projet et qu'il n'est pas le propriétaire du projet.

3 Collaboration sur les projets

3.1 Websocket

3.1.1 Connexion

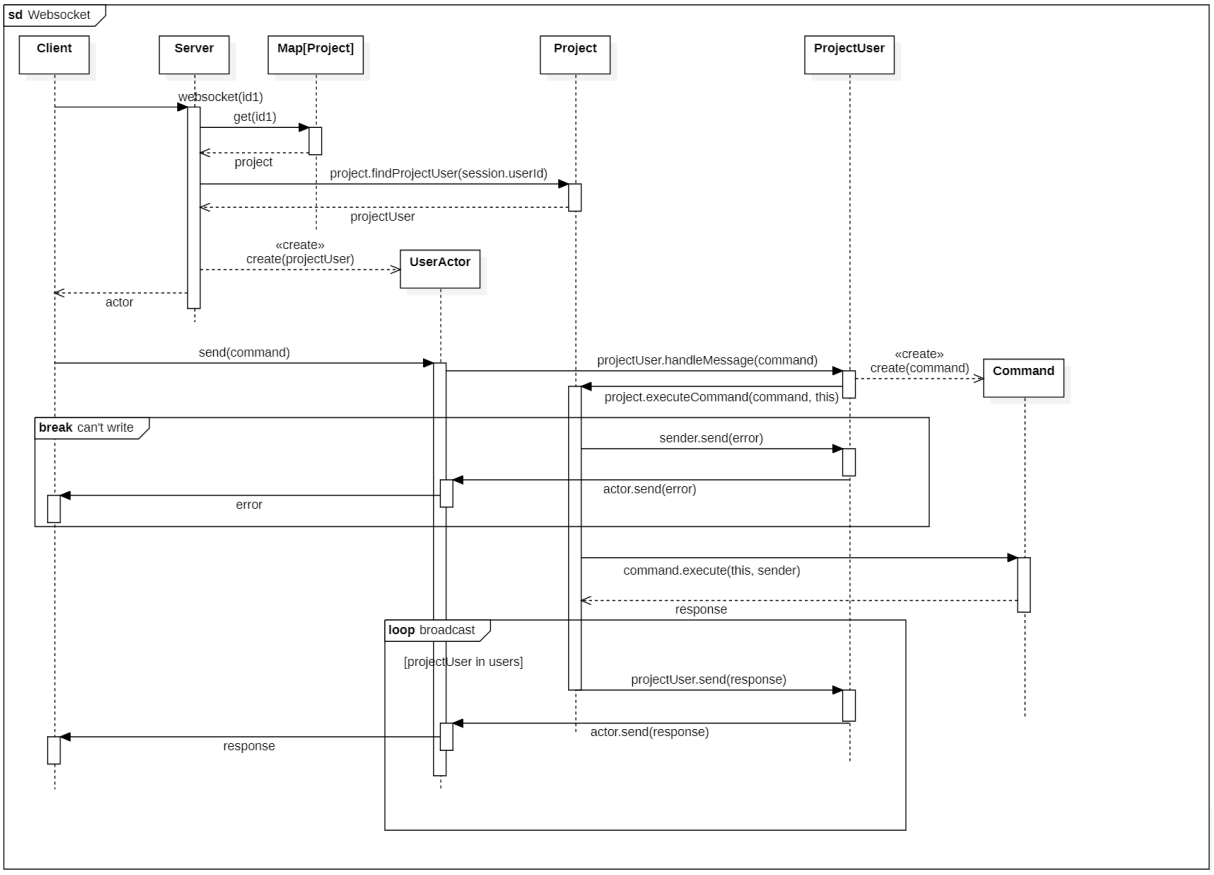

Figure 12 : Diagramme de séquence du websocket

Afin de pouvoir collaborer en instantané, il est nécessaire d'utiliser les websockets.

Les websockets sont gérés par des Akka.Actor. Ils permettent d'avoir des fonctionnalités, tels que la détection de déconnexion ou encore d'associer des objets métiers à un Actor.[3]

Dans le cadre de ce projet, lors de la création de l'Actor, on lui associe un ProjectUser, symbolisant un utilisateur d'un certain projet. De cette manière, il est possible d'ouvrir différents websockets sans pour autant avoir des conflits. Chaque connexion aura son propre ProjectUser, qui aura une référence sur le bon projet.

3.1.2 Déconnexion

Une déconnexion d'un websocket peut arriver dans 2 cas.

3.1.3 Le client se déconnecte

L'Akka Actor va exécuter une fonction, qui permettra de notifier le projet qu'un utilisateur s'est déconnecté.

[3] https://www.playframework.com/documentation/2.8.x/JavaWebSockets

Si tous les utilisateurs sont déconnectés, le projet se ferme.

3.1.3.1 Le serveur déconnecte un client

Lorsqu'un projet est supprimé, ou qu'un collaborateur est retiré d'un projet, le serveur termine la connexion avec les utilisateurs concernés.

3.2 Commandes

Les utilisateurs connectés à un websocket peuvent envoyer des messages JSON via ce dernier, qui seront traduits en Command. Ces commandes seront ensuite exécutées dans le contexte du projet en question.

Un message résultant de l'exécution de la commande est ensuite envoyé à tous les utilisateurs connectés.

Les messages échangés sont créés via des DTO.

3.2.1 ChatMessage

Cette commande crée simplement une réponse contenant le nom de l'utilisateur l'ayant exécuté, ainsi que le message de ce dernier.

Tous utilisateurs, même ceux n'ayant pas les droits de modifications peuvent utiliser cette commande.

3.2.2 Select

Cette commande sert à récupérer des informations sur le diagramme. Notamment, lors d'une connexion au websocket, il est nécessaire de récupérer le diagramme dans son entièreté.

Tous les utilisateurs peuvent utiliser cette commande.

3.2.3 Create

Cette commande crée une entité sur le serveur. Si plus d'une entité doit être créée, la commande de création de chaque entité est envoyée à chaque utilisateur connecté.

Seuls les utilisateurs ayant le droit de modification peuvent utiliser cette commande.

3.2.4 Update

Cette commande est la plus complexe. Elle modifie non seulement l'entité souhaitée, mais également les entités lui faisant référence. Par exemple, une méthode ayant un paramètre de type « Person », « Person » étant une classe du diagramme, si le nom de la classe « Person » change, la modification du paramètre doit se faire également. Cela est fait automatiquement sur le serveur, ce qui veut dire qu'on recharge la page, le nom est bien mis à jour à tous les endroits.

J'ai mis en place un mécanisme qui permet que le nom soit directement modifié, sans avoir à recharger la page. Pour cela, j'ai utilisé une liste de « Subscribers », qui lors d'une modification du nom d'un Type, on crée également une commande signifiant leur mise à jour à envoyé à tous les utilisateurs.

Seuls les utilisateurs ayant le droit de modification peuvent utiliser cette commande

3.2.5 Remove

Cette commande sert simplement à retirer une entité du diagramme.

Seuls les utilisateurs ayant le droit de modifications peuvent utiliser cette commande.