**Projet FADDEV 31-21-02**

**Rotten potatoes**

**Site communautaire de notation et de critique de jeux vidéos**

Par Sébastien Leyrissoux

## **Introduction**

This project is based on the following fictive scenario : The website Rotten tomatoes want to expand his universe with a new website dedicated to the notation and review of video games by the community. To achive this objective, they wish to make a whole new interface while keeping the original website indentity, with the new name Rotten potatoes. This new website will be realised from scratch with the following languages : HTML, CSS, javascript, PHP and will be linked to a mySQL database.



## **REAC**

|  |  |
| --- | --- |
| Maquetter une application | Projet |
| Réaliser une interface utilisateur web statique et adaptable | Projet |
| Développer une interface utilisateur web dynamique | Projet |
| Réaliser une interface utilisateur avec une solution de gestion de contenu ou e-commerce | Dossier professionnel |
| Créer une base de données | Projet |
| Développeur les composants d’accès aux données | Projet |
| Développer la partie back-end d’une application web ou web mobile | Projet |
| Elaborer et mettre en œuvre des composants dans une application de gestion de contenu ou e-commerce | Dossier professionnel |

## **Analyse du besoin**

Présentation de l’entreprise

Rotten tomatoes est un site web dédié aux informations et aux critiques de films et de séries, séparées en critiques de la communauté et critiques professionnelles. On y retrouve des informations sur les dernières sorties, les films/séries les plus populaires du moment, ainsi que des listes diverses et variées comme les films par réalisateur, par année de sortie, par acteur… et divers articles écrits par les membres de l’équipe en charge du site.

Contexte – besoin

Comme cité dans l’introduction, ce projet représente une demande fictive du site Rotten tomatoes, le besoin étant d’étendre l’univers des médias traités par le site via un tout nouveau site dédié à la critique de jeux vidéo. Le but étant de retrouver les fonctionnalités présentes sur le site original, on pourra donc y visualiser les dernières sorties, les sorties prochaines, noter et poster une critique des jeux

Contraintes techniques

Temps : C’est un grand projet pour un développeur en apprentissage, réussir à mener l’intégralité du projet à son terme dans le temps impartit sera compliqué

Interface responsive : Une interface adapté aux différents formats d’écrans est devenue la norme, il faudra donc mettre en place une solution responsive.

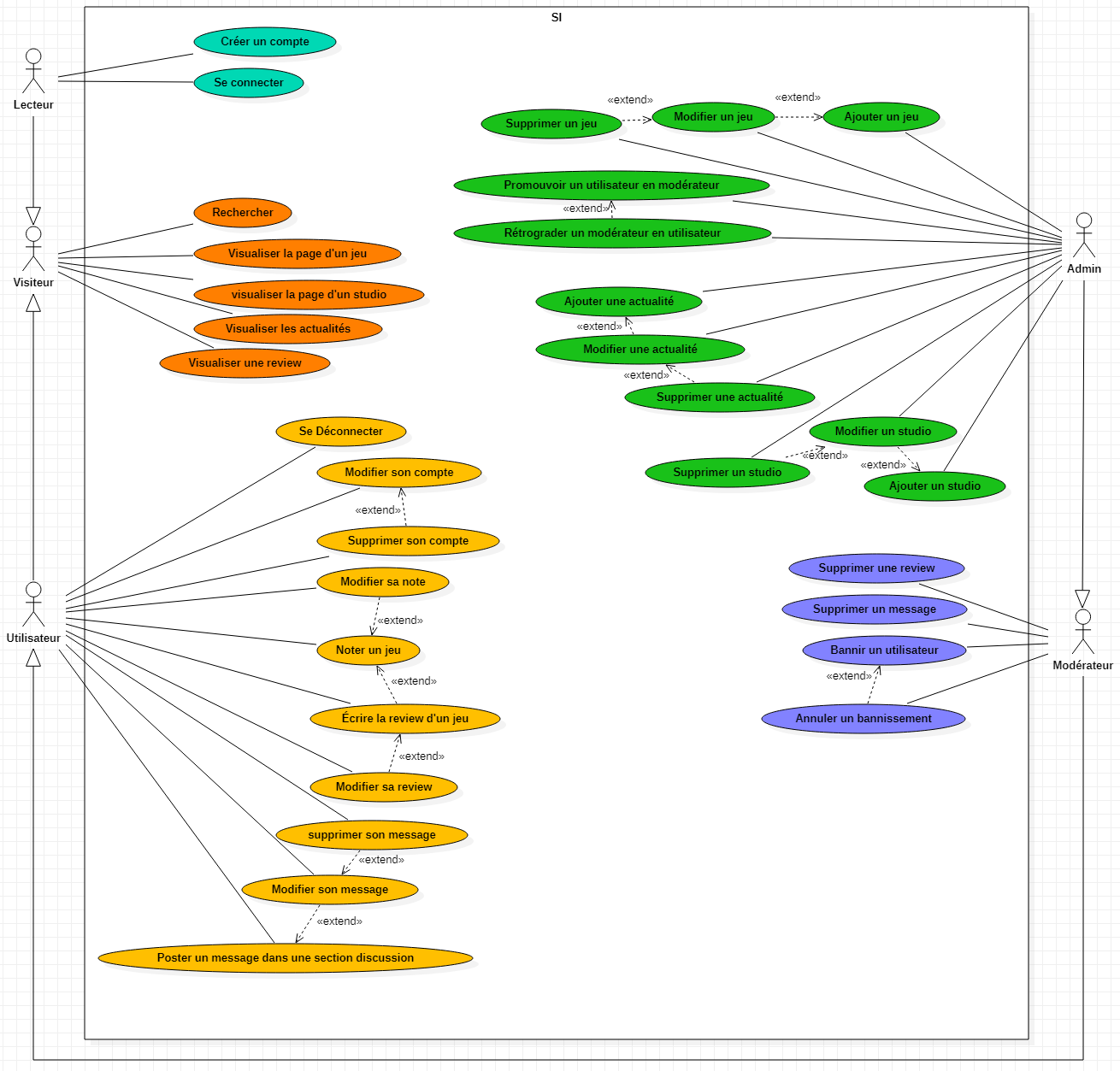
Français Anglais : Rotten tomatoes étant un site anglophone, les langues Anglais et Français serait nécessaire, mais vont nécessiter une traduction entière du site.

Charte graphique à respecter : il faudra respecter un minimum la charte graphique de rotten tomatoes pour créer un lien visuel entre les deux sites, ce qui limitera les choix créatifs possibles.

Langages choisis : le site étant développés avec les langages HTML, CSS, Javascript, PHP, il faudra s’adapter aux contraintes de ces langages pour développer les fonctionnalités du site.

## **Spécifications fonctionnelles**

Use case



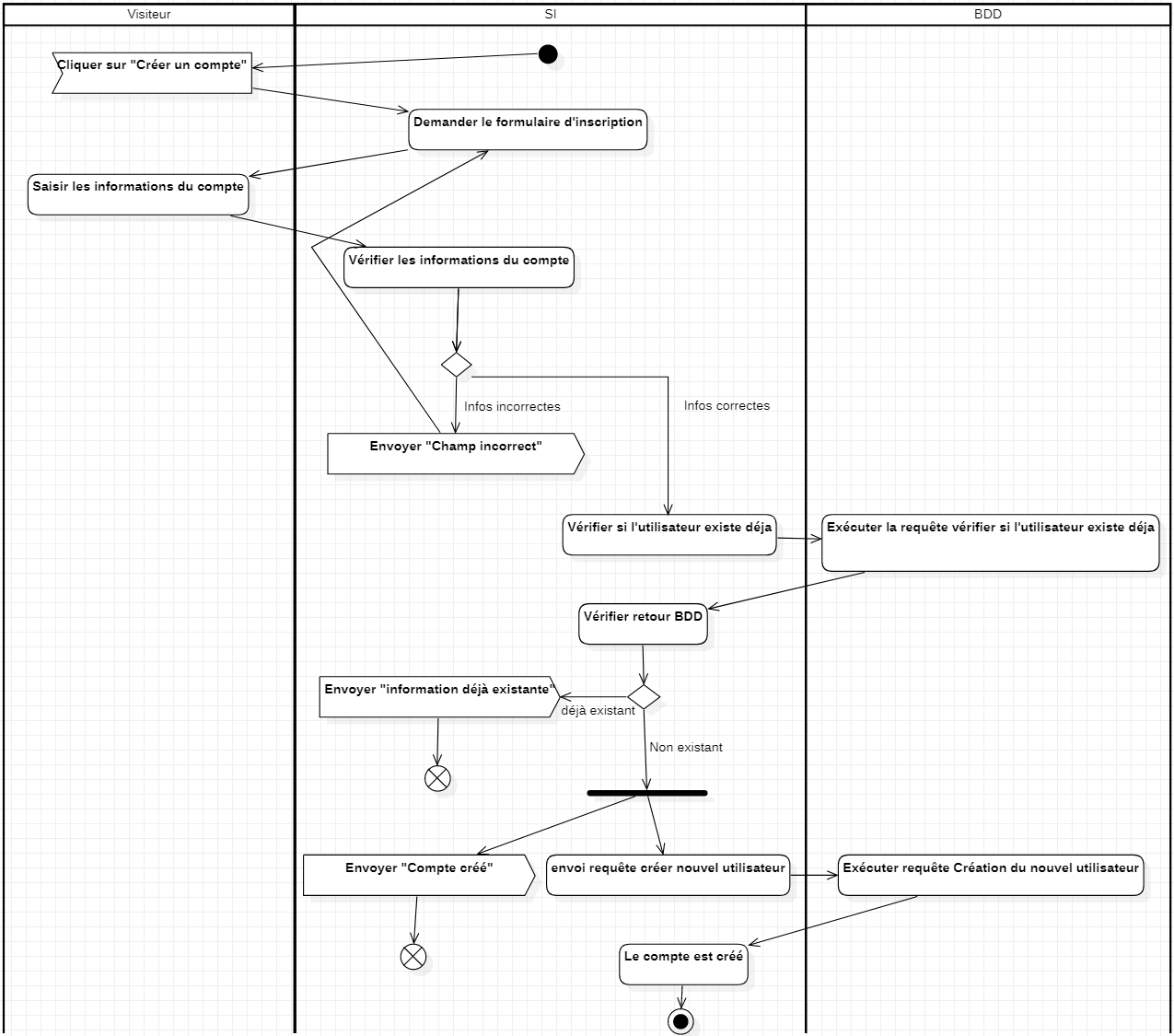
Les fonctions du diagramme des cas d’utilisations sont réparties sur 5 acteurs : Le visiteur, qui représente toute personne navigant sur le site sans compte, l’utilisateur, toute personne qui navigue sur le site en étant connectée à son compte, le lecteur, qui représente les fonctions de création de compte et de connexion auxquelles le visiteur a accès mais pas l’utilisateur, le modérateur, un utilisateur qui dispose des fonctions de modération de la communauté et l’administrateur qui est en charge de la gestion du contenu du site.

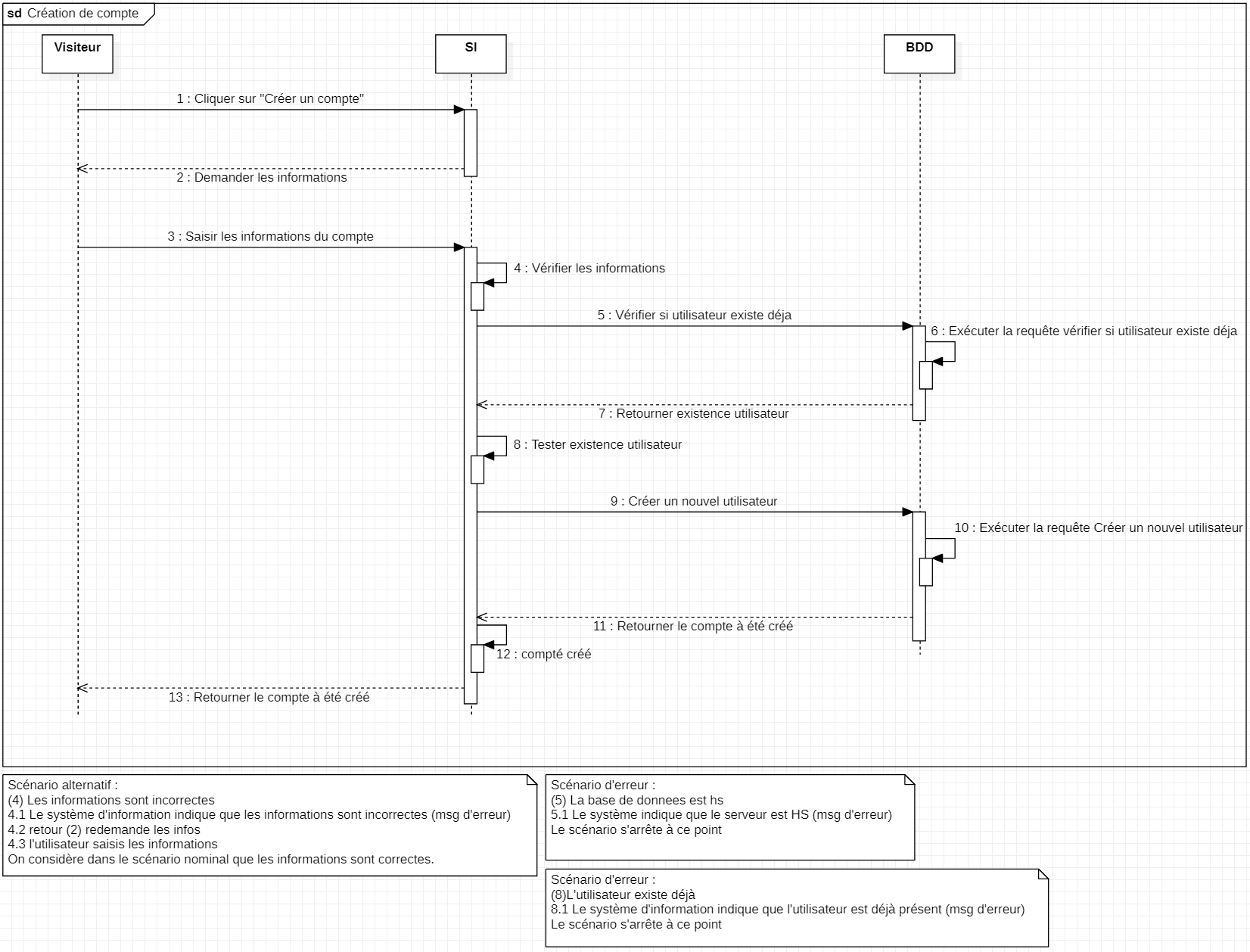
Dans la partie suivante seront présentés les diagrammes d’activité et de séquence des fonctions suivantes : création de compte, connexion, déconnexion, modification du compte et suppression du compte.

Diagrammes d’activité et de séquence

Création de compte

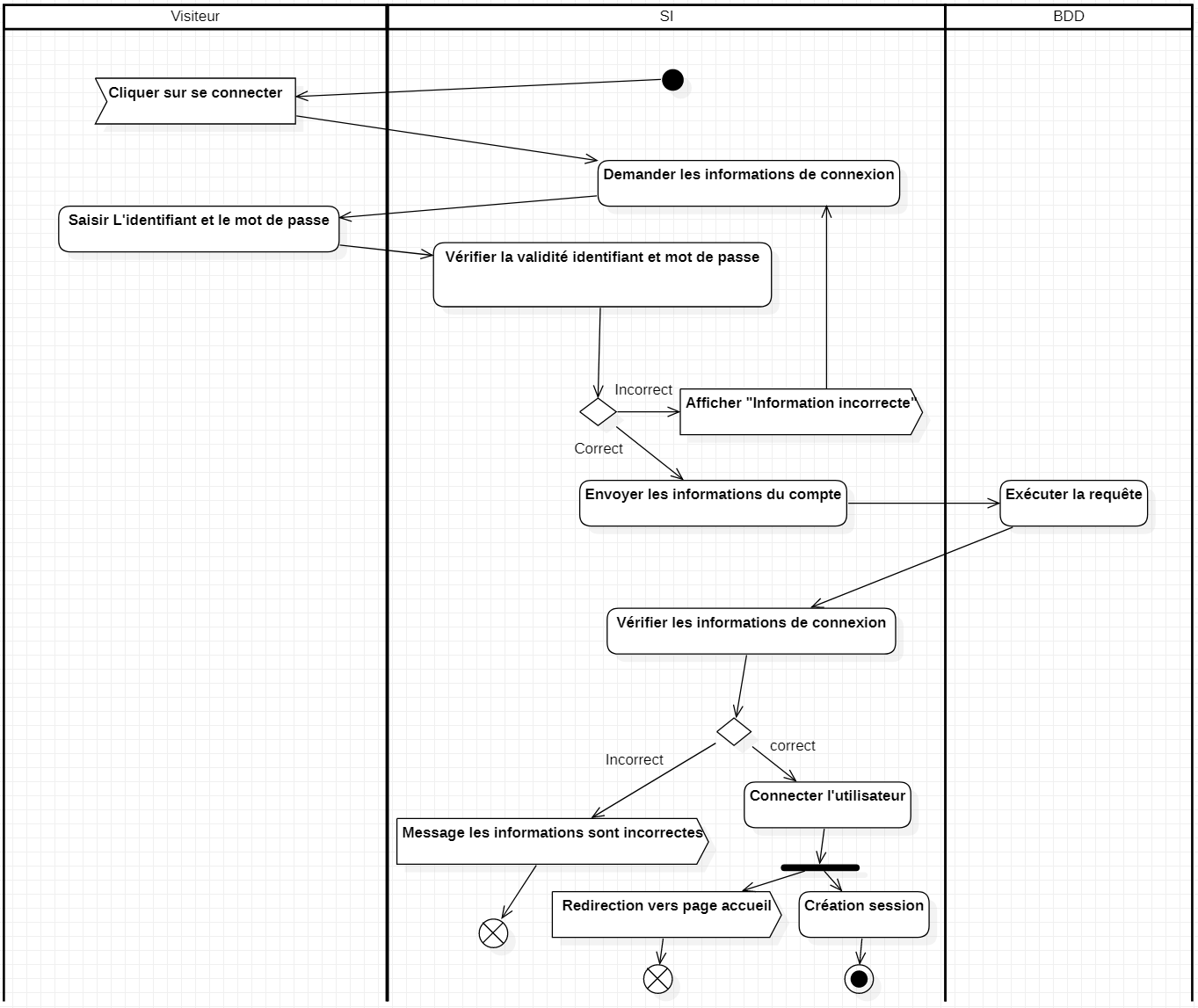
La création de compte, avec la validation des données entrés par l’utilisateur, la comparaison avec la BDD pour éviter les doublons et la création dans la BDD.

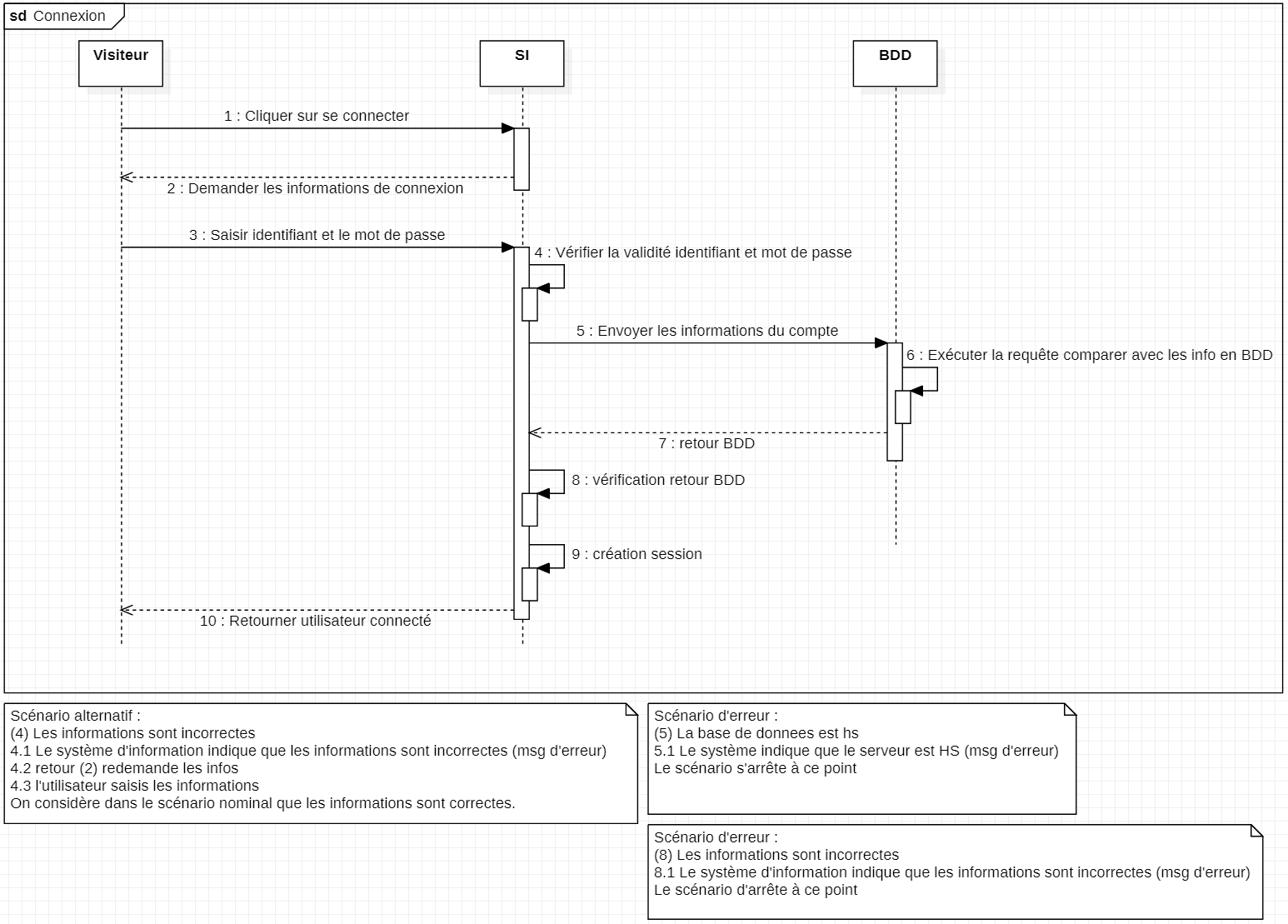




Connexion

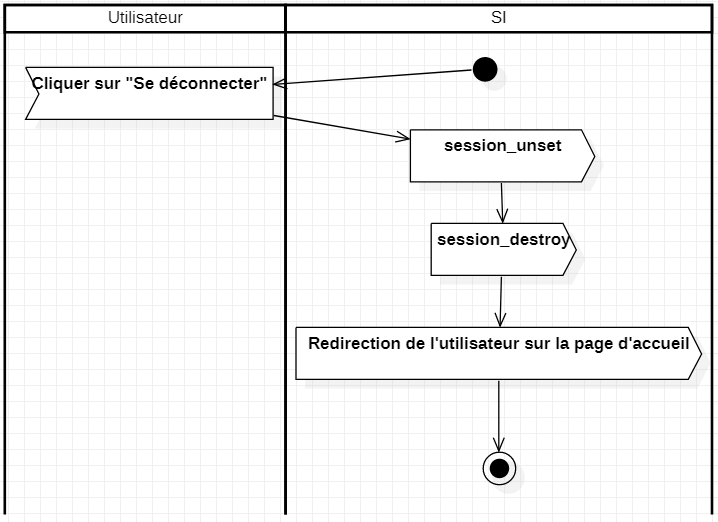
La connexion avec la validation des données entrés par l’utilisateur, la comparaison avec la BDD pour vérifier l’existence de l’utilisateur et la création de la session.

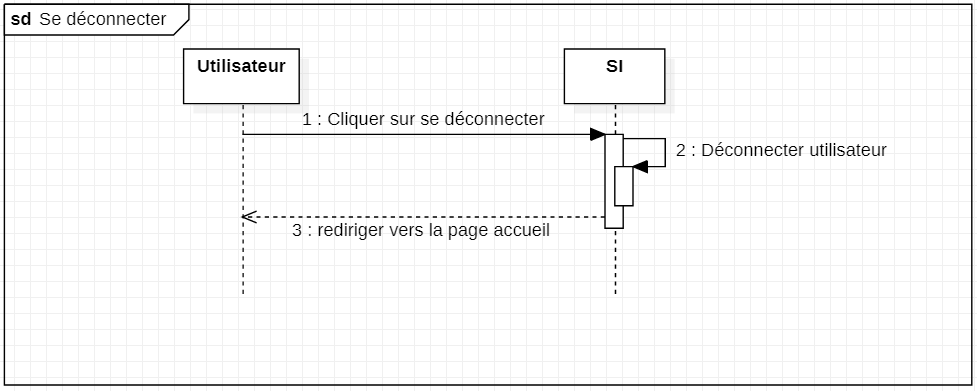




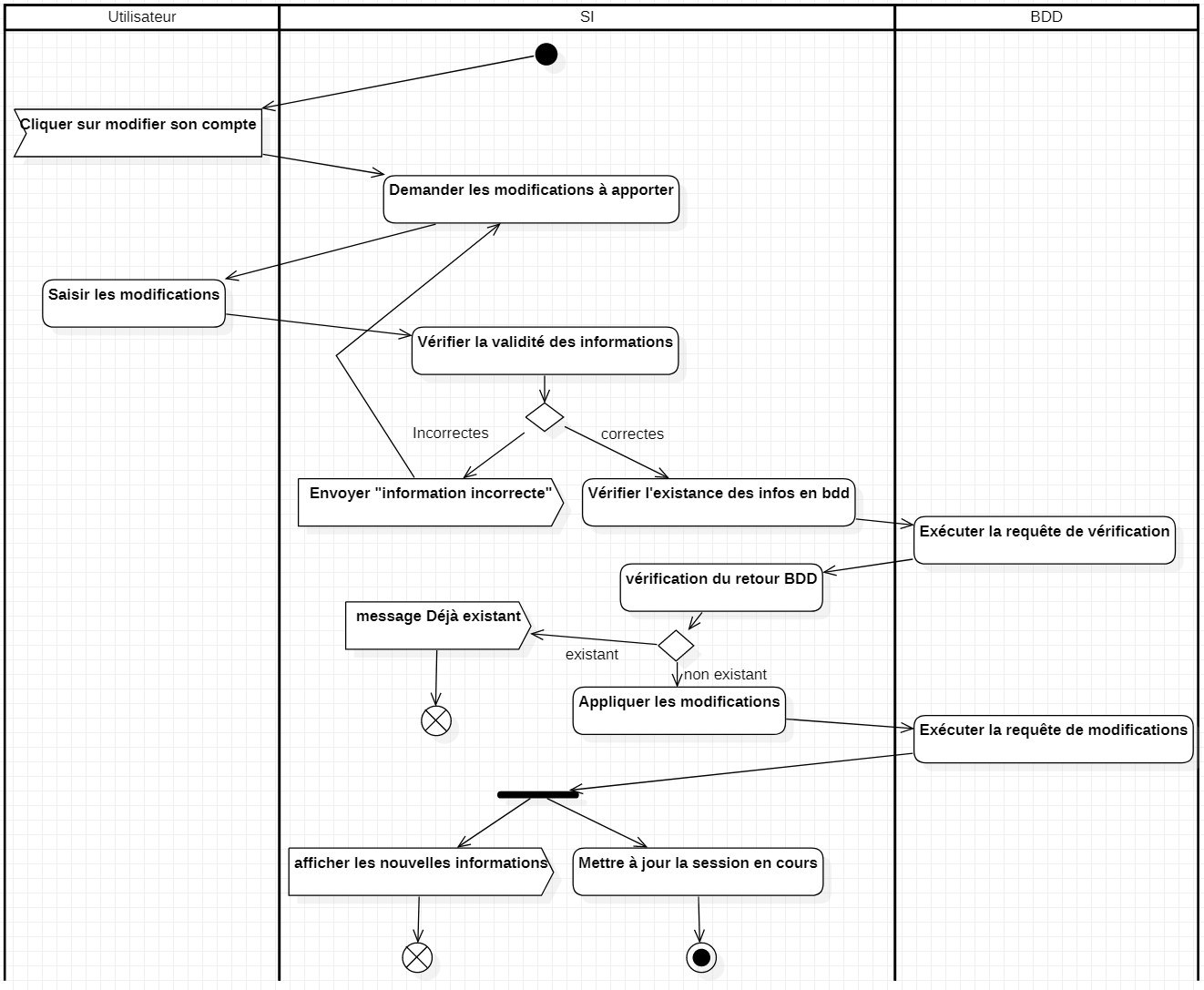
Déconnexion

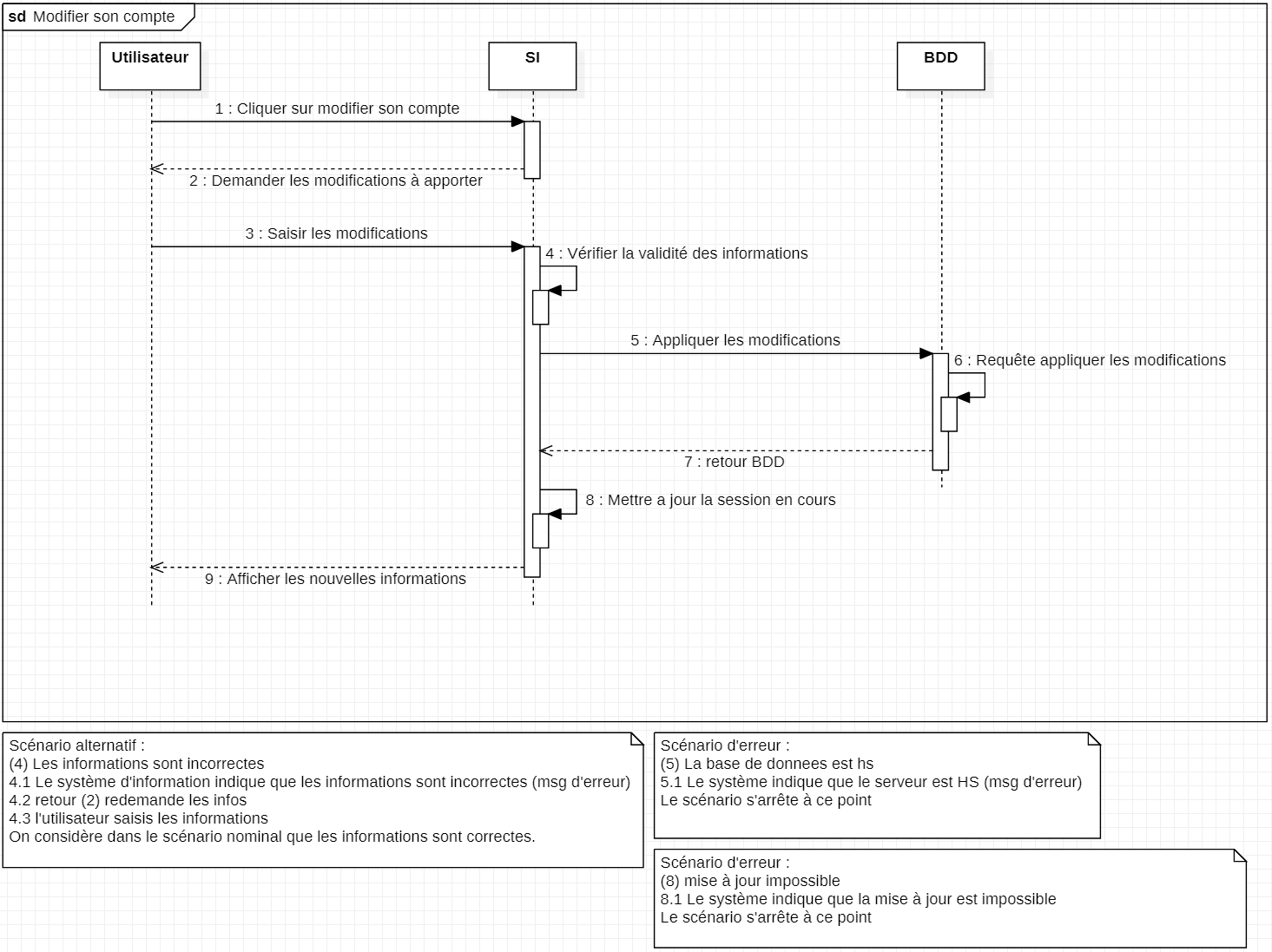
La déconnexion avec session\_unset et session\_destroy pour effacer les données de la session utilisateur du navigateur.





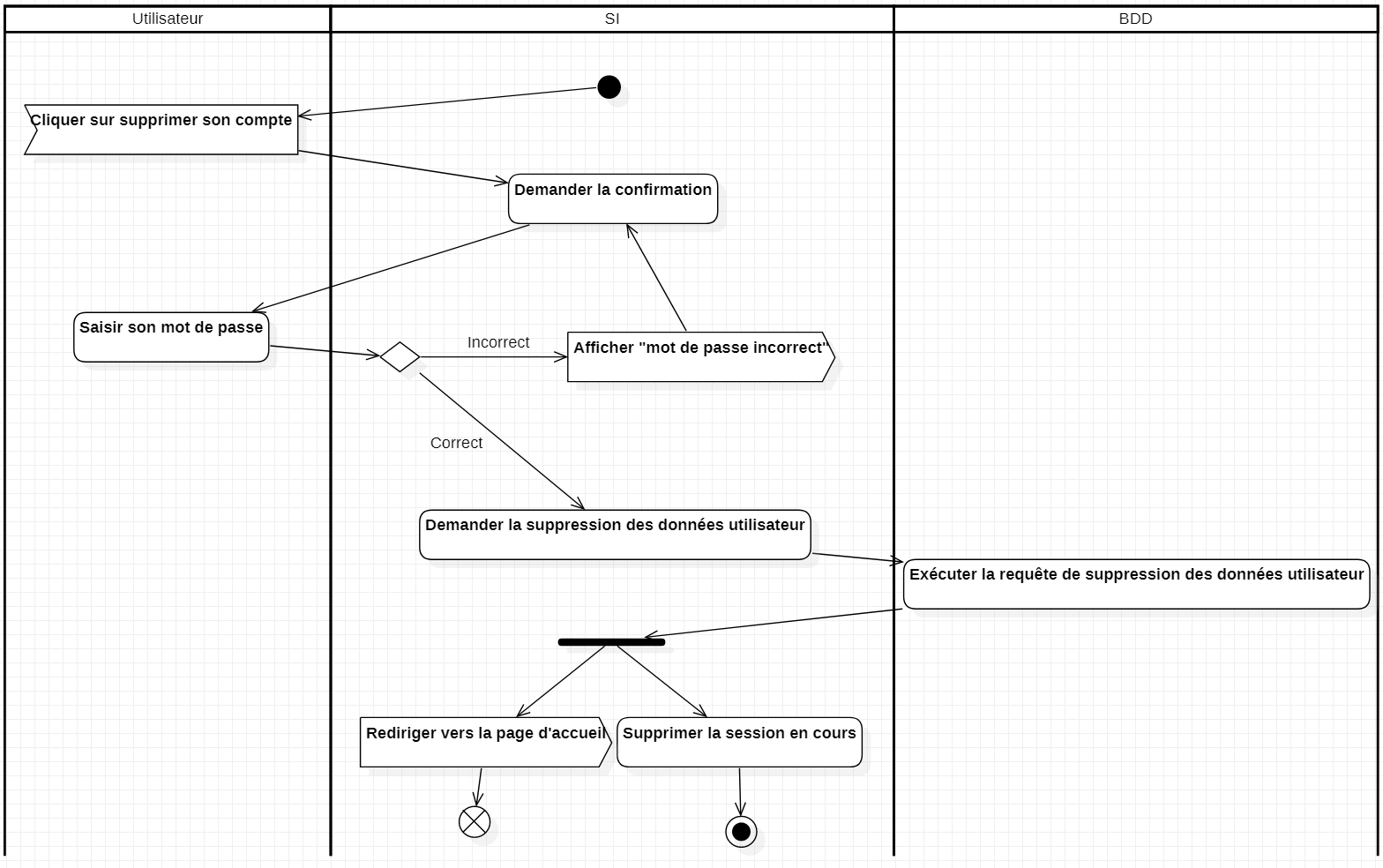
Modification

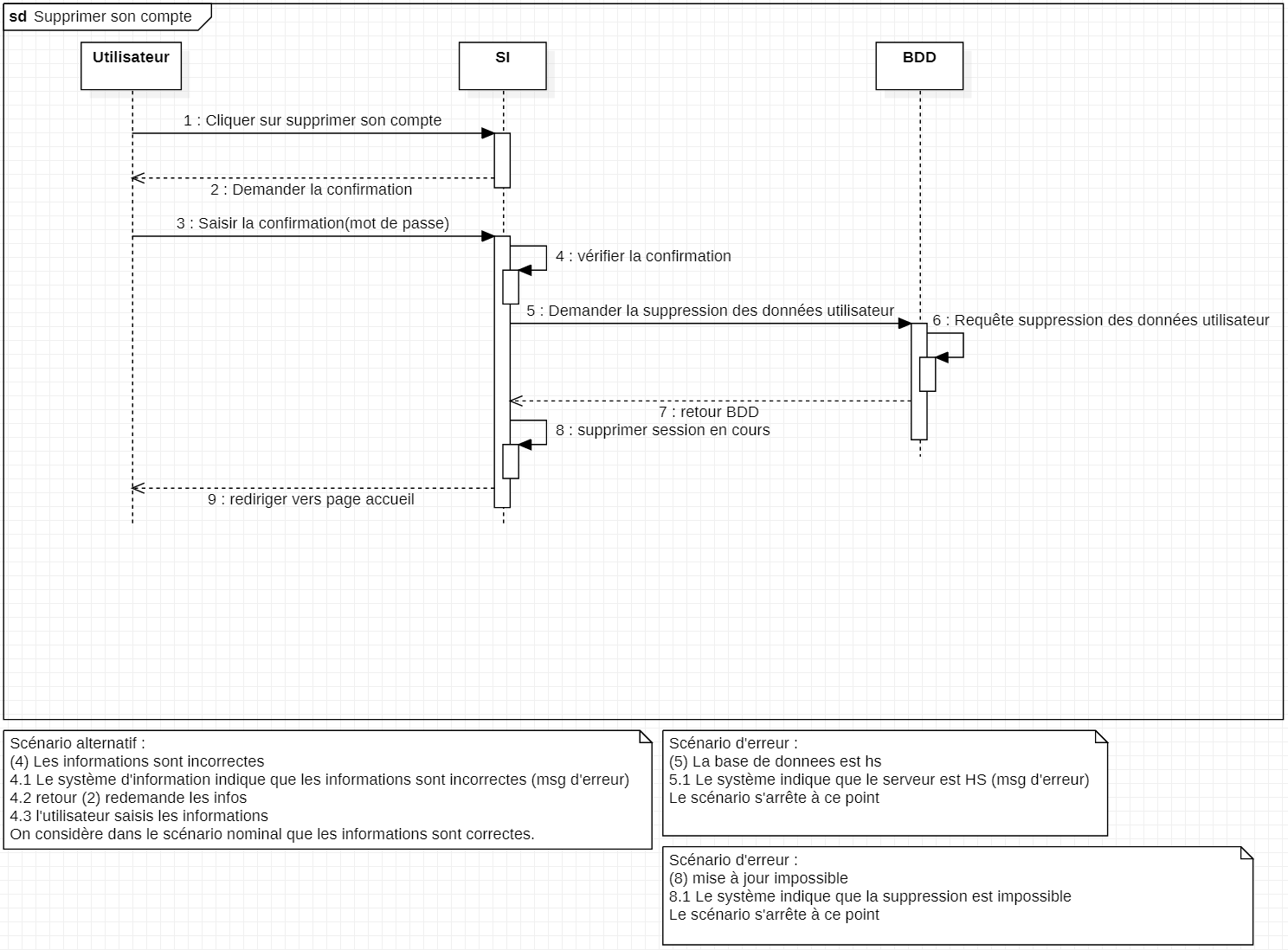
La modification, avec la validation des données entrés par l’utilisateur, la comparaison avec la BDD pour éviter les doublons, la création dans la BDD et la mise à jour de la session.



Suppression

La suppression, avec la vérification du mot de passe de confirmation, la suppression en BDD et la suppression de la session en cours.

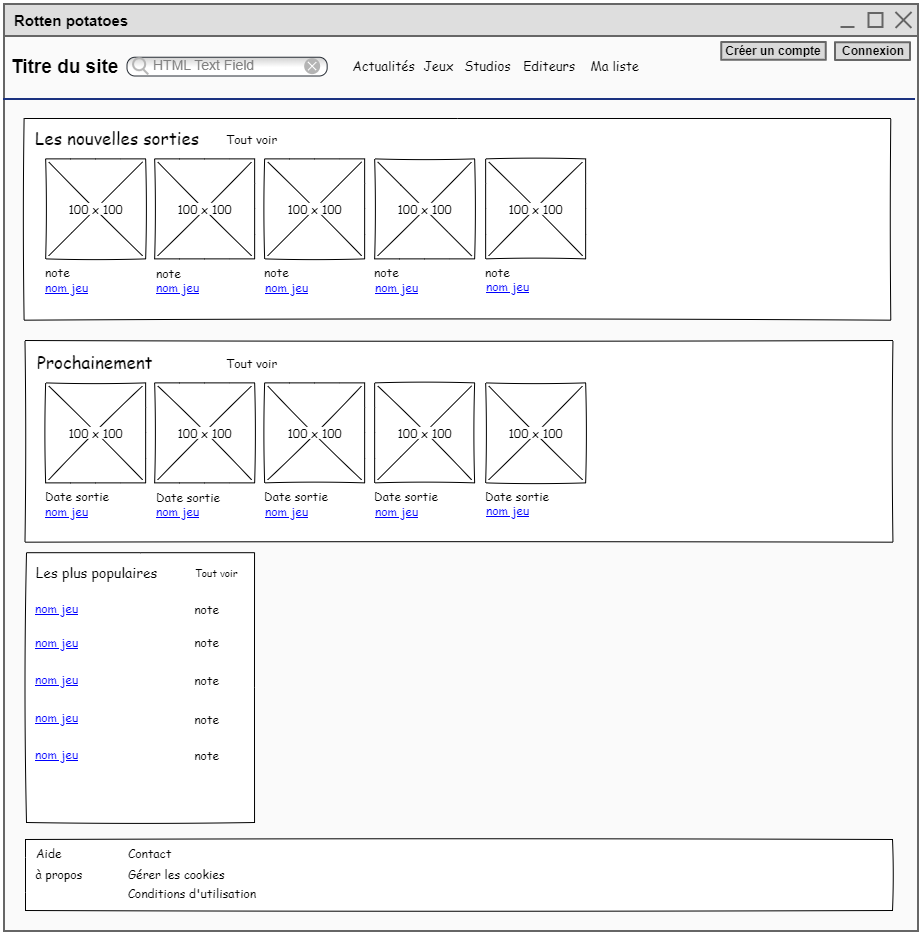




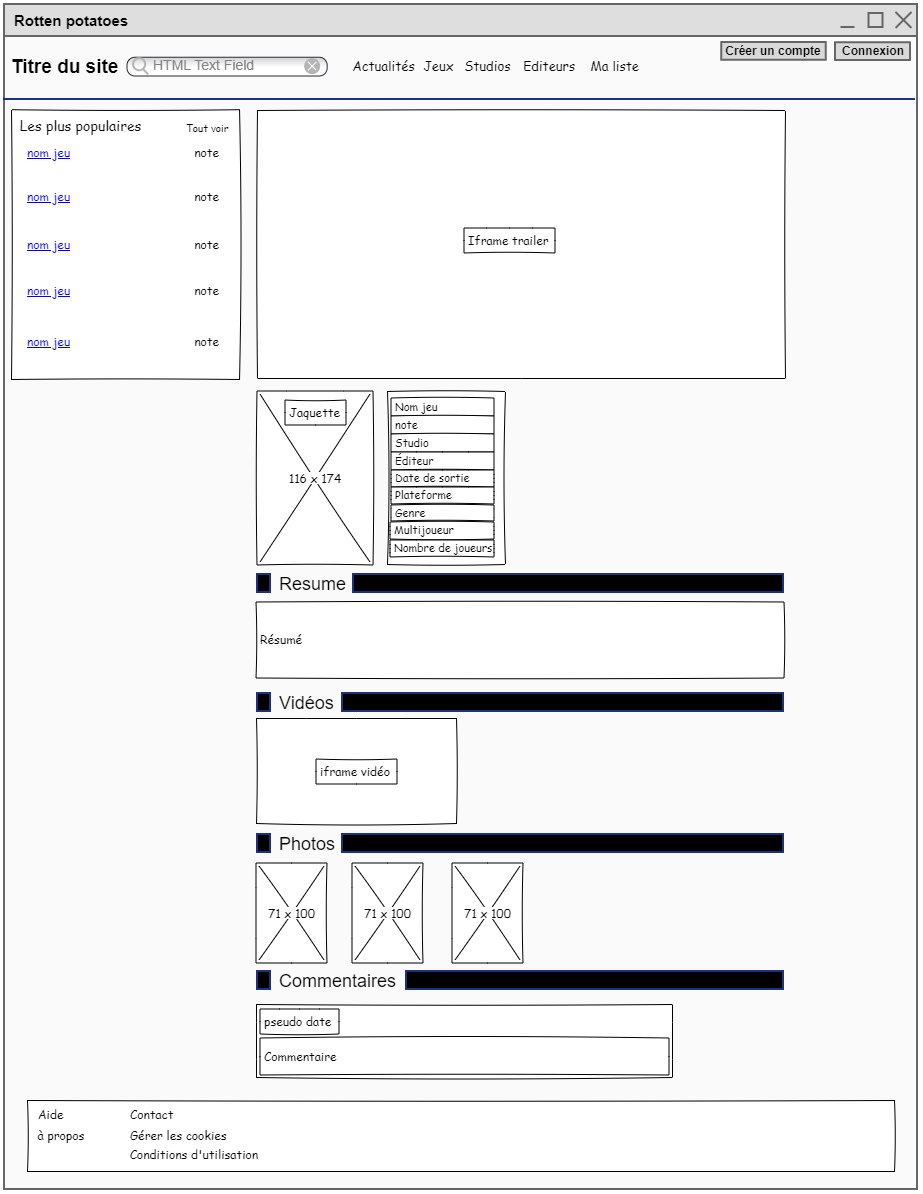
Maquettes

Par soucis de pertinence j’ai choisi de montrer les maquettes des pages accueil et jeu, qui sont pour moi les deux pages et plus importantes du site.

Le wireframe de la page d’accueil



Le wireframe de la page jeu

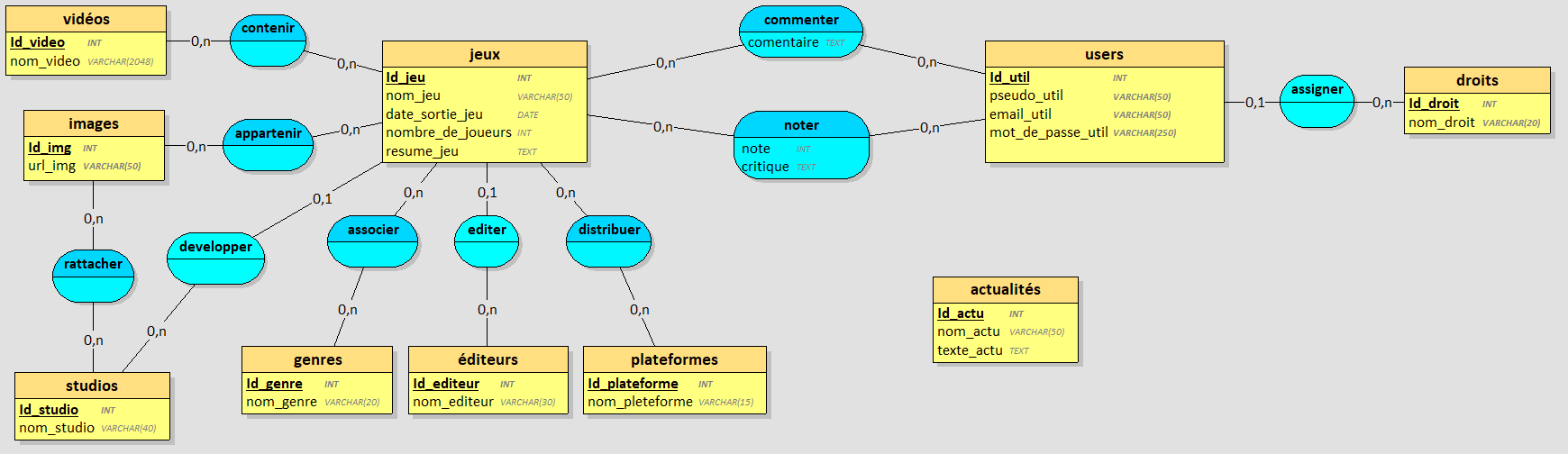


## **Conception**

Voici le modèle logique de la base de donnée de Rotten potatoes

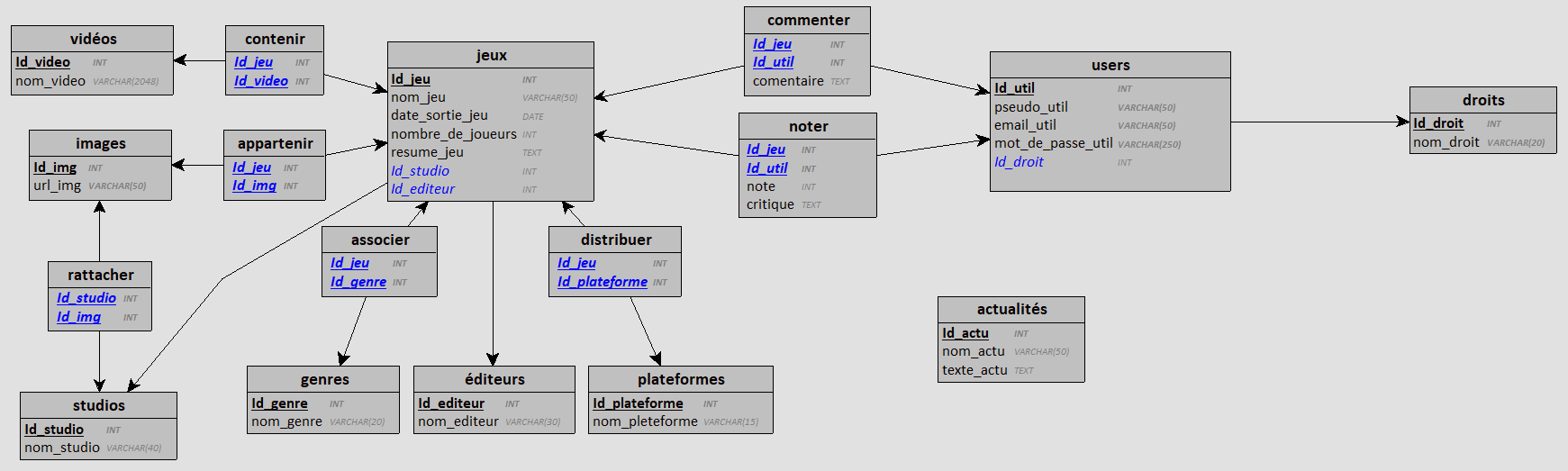
MCD

Le mcd de la BDD, où l’on peut voir la table principale du site, jeux, reliée aux autres tables mineures par des associations majoritairement avec des cardinalités en 0,n, 0,n, à l’exception de certaines en 0,1, 0,n et des deux liaisons avec la table user qui contiennent des champs. Nous avons enfin la table studio reliée à la table images, droits relié à users et actualité toute seule.

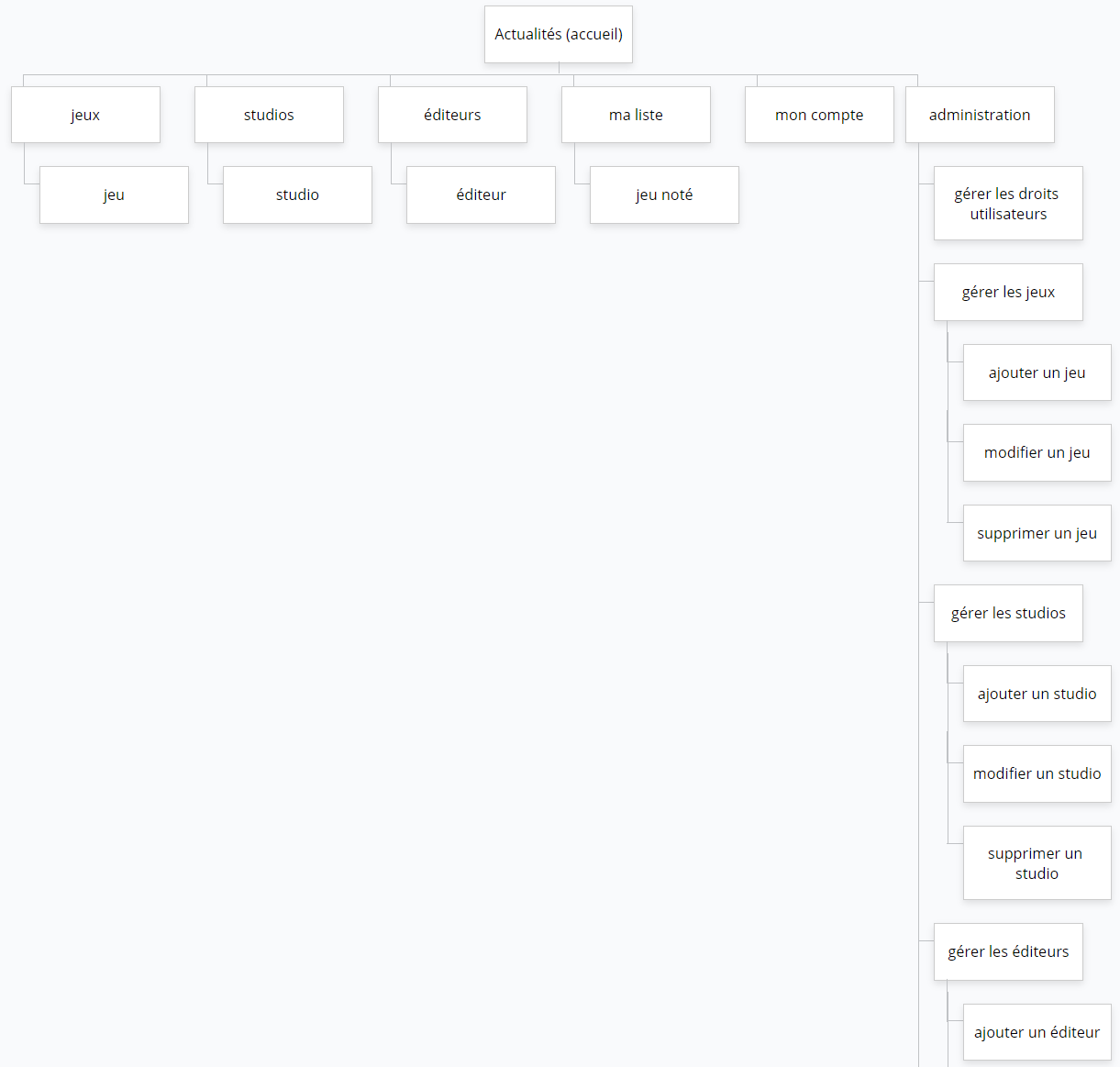
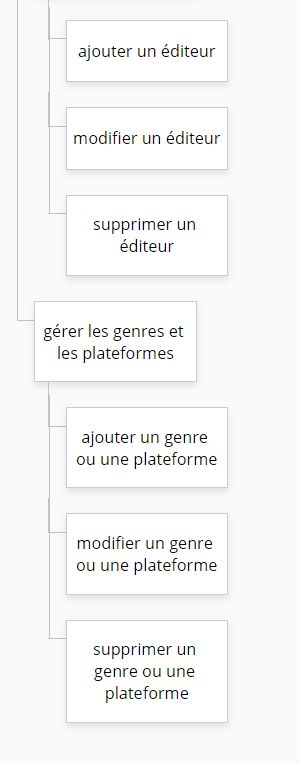


MLD

Le mld permet de voir la disposition réelle de la BDD, avec les associations en 0,n qui deviennent les tables d’association et les 0,1, 0,n qui se transforment en absorbions de clé étrangère.



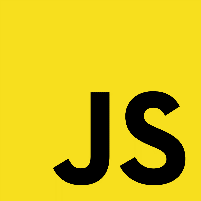
Arborescence

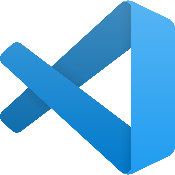


Outils techniques utilisés

Pour mettre en œuvre ce projet, la partie front-end sera réalisée en **HTML** **CSS**, avec des éléments en **Javascript** pour gérer certains éléments dynamiques. La partie back-end sera quant à elle réalisée avec une structure suivant le modèle **MVC** en **PHP**. Le tout sera connecté à une base de donnée **SQL** accessible via des requêtes en **MySQL** et administrable grâce à **phpMyAdmin**.

Le code sera développé via **Visual Studio Code**, sauvegardé et versionné avec **Git**. Les requêtes de création de la base de donnée seront faites via **MySQLWorkbench**. Le site sera hébergé sur un serveur local créé via **Wamp**.

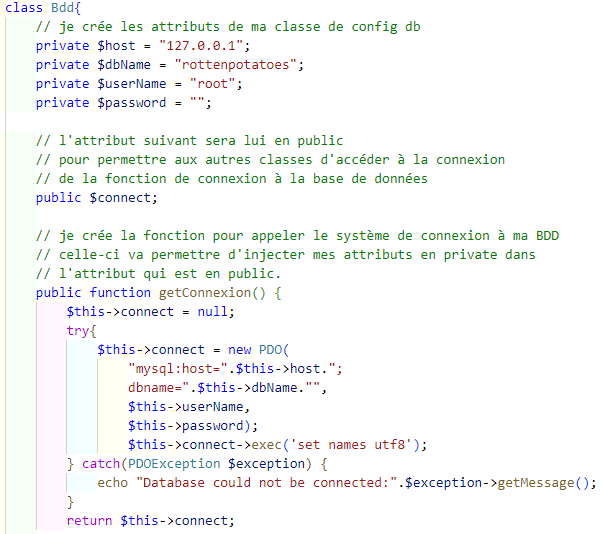
    

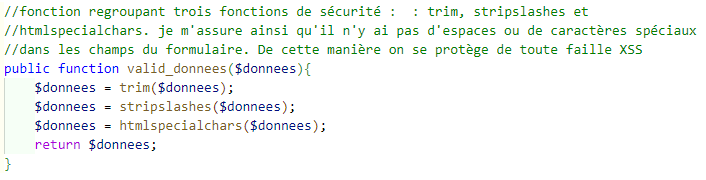
Fonctionnalités

Les fonctionnalités choisies correspondent aux fonctions présentées précédemment dans les diagrammes, mais il faut au préalable présenter deux fonctions utilitaires nécessaires au fonctionnement de reste du projet : la connexion à la base de donnée et la validation des données.

La connexion à la base de donnée est dans un fichier à part puis intégrée aux modèles pour éviter la répétition du code :

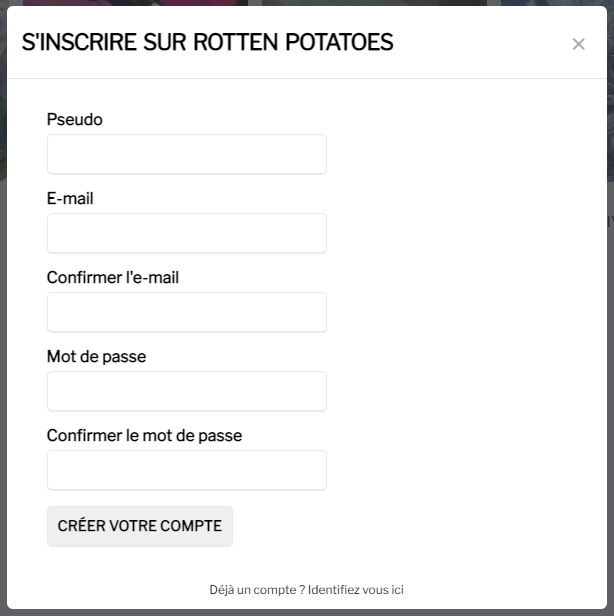


La fonction valid\_donnees quant à elle regroupe plusieurs fonctions de sécurité nécessaires au traitement des données envoyées par les utilisateurs au site, on y retrouve trim, pour enlever les espaces, stripslashes pour enlever les slashs et htmlspecialchars pour convertir les caractères spéciaux comme les quotes ou les chevrons en entités html.



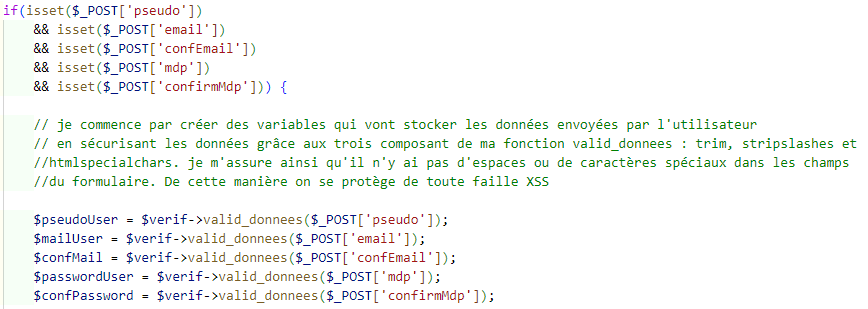
Création de compte

La création de compte passe tout d’abord par un formulaire disponible dans une modale depuis le header du site

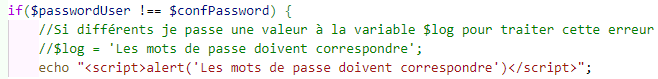


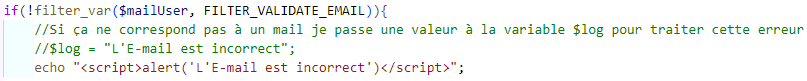


Qui va envoyer les données au controller, qui une fois la présence des champs obligatoires vérifiée, va les faire passer dans la fonction valid\_donnees pour les sécuriser.

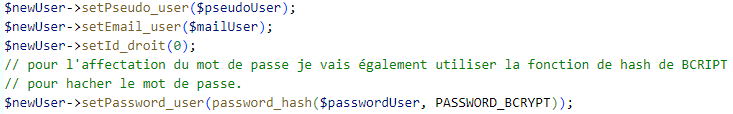


Ensuite il va comparer le champ mot de passe avec sa confirmation et renvoyer une erreur s’ils sont différents, le même processus sera appliqué à l’email et sa confirmation avec en plus la fonction FILTER\_VALIDATE\_EMAIL qui vérifie que le format de l’email est correct.

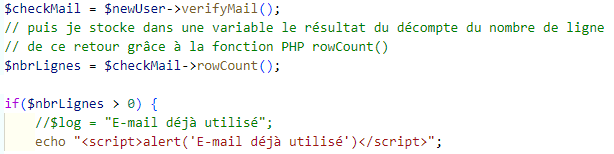




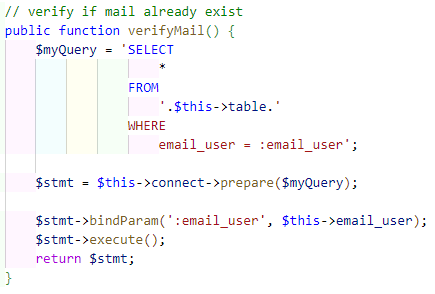
Si ces étages de vérifications se passe correctement, le controller va ensuite utiliser les setters de la classe User pour affecter les valeurs des variables aux attributs de la classe, tout en utilisant la fonction password\_hash avec PASSWORD\_BCRYPT sur le mot de passe pour le hacher et ainsi éviter l’insertion en clair dans la base de donnée.



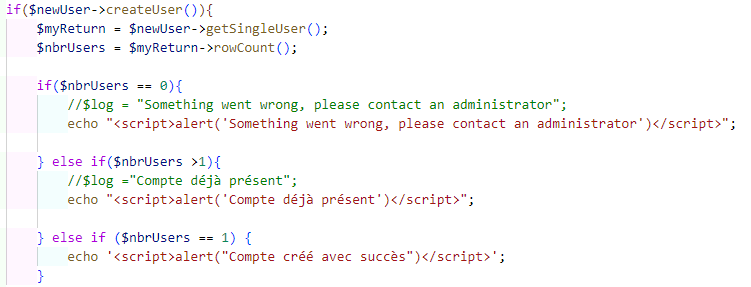
Une fois fait, le controller va pouvoir vérifier si le pseudo et le mail sont déjà présents en BDD, et renvoyer une erreur si c’est le cas. Pour cela les fonctions verifyMail et verifyPseudo seront utilisées en combinaison avec un rowCount pour vérifier la présence de champs dans le retour de la requête.



verifyMail du modèle user :



Pour finir le controller va exécuter la fonction createUser pour créer l’utilisateur en BDD grâce à des requêtes préparées. Si la création se passe comme prévu, il va vérifier qu’il y a bien un seul utilisateur avec le pseudo choisi dans la BDD avec la fonction getSingleUser de User et rowCount, et renvoyer une erreur si ce n’est pas le cas.

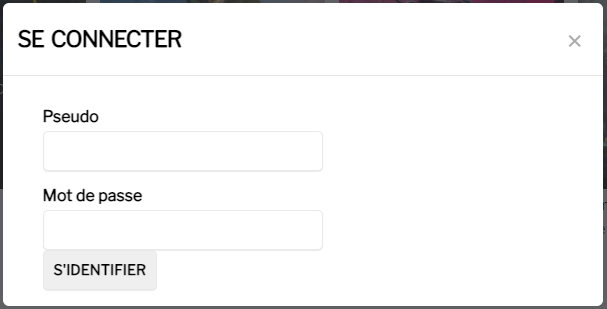


createUser du modèle user :

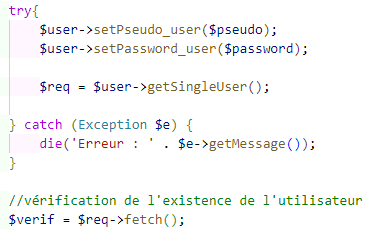


Une fois la création terminée, l’utilisateur est renvoyé sur la page d’accueil.

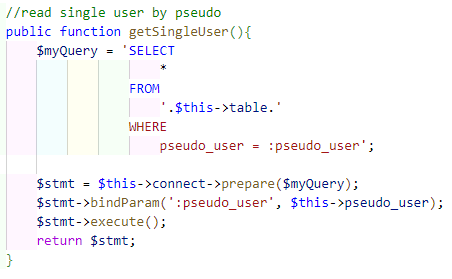
Connexion

La connexion passe aussi en premier lieu par un formulaire dans une modale.

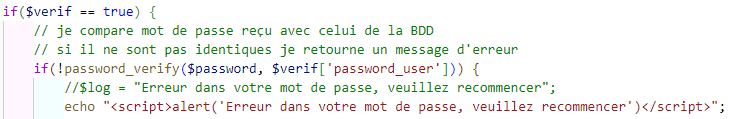
Une fois la présence des champs vérifié et la fonction valid\_donnees utlisée, on utilise le setPseudo\_user pour affecter la valeur du pseudo à l’attribut correspondant et ainsi utiliser getSingleUser pour vérifier la présence de l’utilisateur dans la BDD.



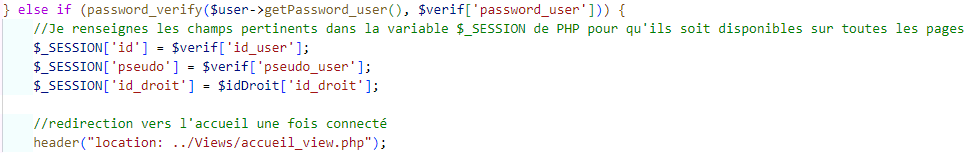
getSingleUser du modèle user :



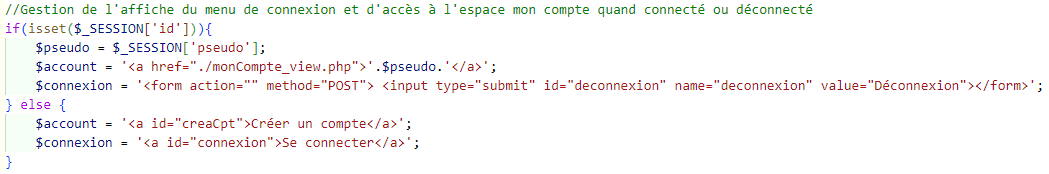
Si l’utilisateur existe bien, on va comparer le mot de passe entré par le visiteur avec celui en BDD en utilisant password\_verify (pour comparer les hash).



Si les mots de passes correspondent, le controller initialise la session en entrant l’id, le pseudo et l’id\_droit de l’utilisateur dans $\_SESSION, qui sera ensuite accessible grâce à un session\_start présent sur chaque view du site.

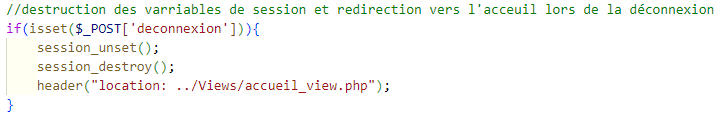


Une fois l’utilisateur connecté le controller va aussi changer les boutons créer un compte et se connecter du header par un bouton d’accès au compte qui affiche le pseudo de l’utilisateur connecté et un bouton déconnexion.



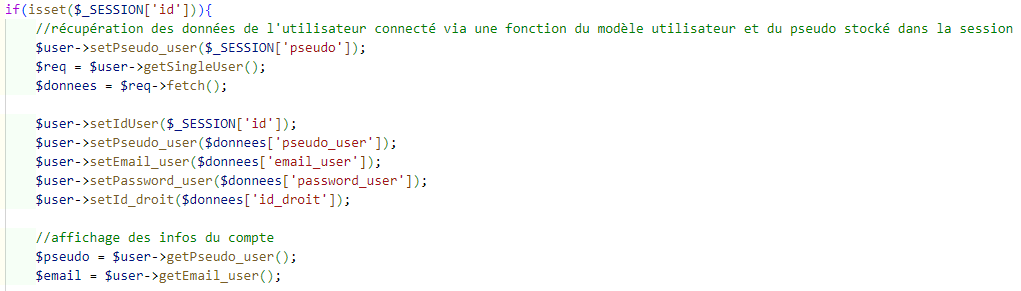
Déconnexion

La Déconnexion d’un utilisateur se fait à partir du bouton déconnexion du header, qui est en fait un submit de formulaire sous forme de bouton. Si l’utilisateur clique dessus et ainsi valide le isset du controller, ce dernier lance un session\_unset pour supprimer les variables de la session, puis un session\_destroy pour supprimer la session. L’utilisateur est ensuite redirigé vers la page d’accueil.

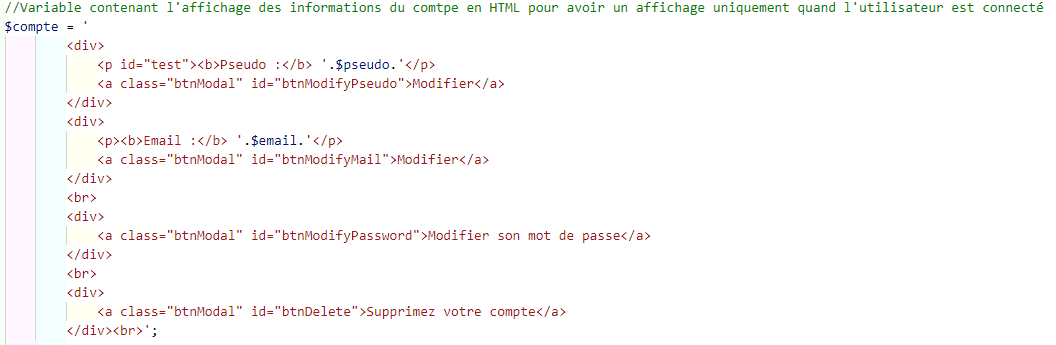


Modification

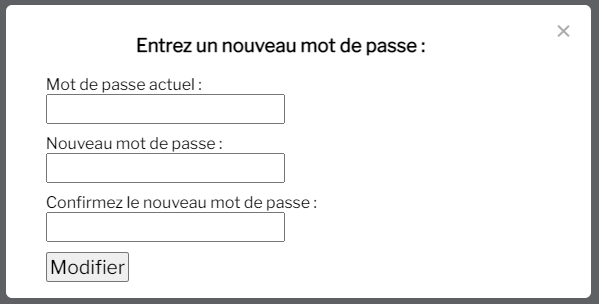
La modification commence par la vérification de la connexion de l’utilisateur par le controller. Si la session est bien initialisée le controller va récupérer les données de l’utilisateur en BDD via un getSingleUser et utiliser les setters pour affecter les valeurs récupérées.



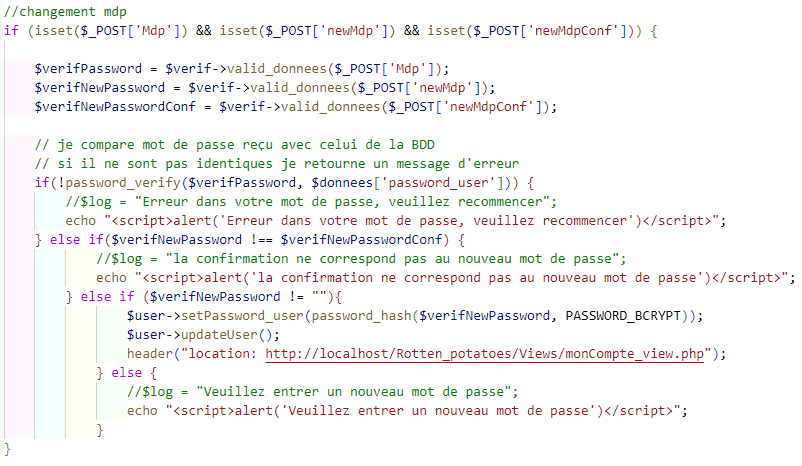
Le controller peut maintenant afficher les informations du compte et les boutons pour les modifier.



La modification de chaque champ passe par un formulaire qui demande le champ en question, à l’exception du mot de passe où le mot de passe actuel et une confirmation du nouveau mot de passe sont demandées.



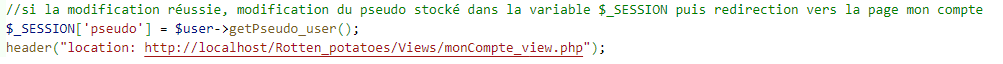
Lors de la demande de modification l’utilisateur, le controller suit le même cheminement que pour la création de compte, vérification de la présence du champ, fonction valid\_donnees, setter et vérification de l’absence du nouveau champ dans la BDD. Une fois toutes ces étapes validées le controller utilise la fonction updateUser de la classe user qui met à jour tous les champs de l’utilisateur, il va donc remplacer tous les champs non modifiés par leur valeur actuelle et les champs modifiés par leur nouvelle valeur.



updateUser du modèle user :

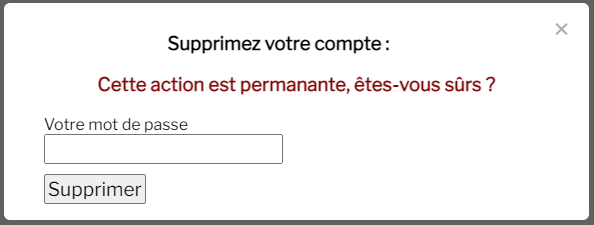


Si nécessaire la session en cours est mise à jour et l’utilisateur est ensuite redirigé vers la page mon compte.



Suppression

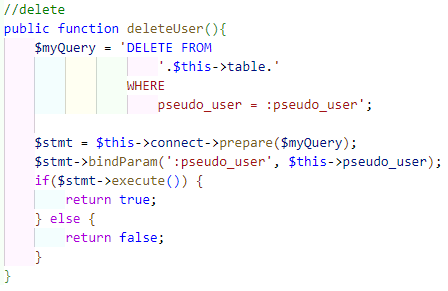
La suppression est disponible sur la même page que la modification avec un formulaire qui demande le mot de passe en confirmation.



Si la vérification du mot de passe via password\_verify est validée, le controller va utiliser la fonction deleteUser pour supprimer l’utilisateur de la BDD, puis supprimer la session en cours comme lors de la déconnexion et enfin rediriger l’utilisateur vers la page d’accueil.



deleteUser du modèle user :



**Conclusion**

Au moment de la rédaction de ce mémoire le projet est loin d’être fini. Côté front, les pages jeu, studio, studios (liste des studios) sont terminées. L’accueil aussi à l’exception du carrousel de jeux, éditeurs est exactement comme studios et est donc presque faite, mon compte contient la modification de compte mais pas les activités de l’utilisateur, l’administration contient uniquement les ajouts de jeu, studio, éditeur, plateforme et genres mais les pages ne sont pas séparées. Enfin les pages jeux, ma liste et le reste de l’administration ne sont pas faites. Il manque aussi la majorité du responsive.

Coté back l’affichage de jeu, studio, et de la liste des studios sont fonctionnels, la création, modification et suppression de compte ainsi que la connexion et la déconnexion aussi. Comme dit au-dessus les ajouts de jeu, studio, éditeur, plateforme et genre sont présents et fonctionnels, mais pas la modification et suppression. Le système de notation et de commentaire et tous les affichages qui en découlent (ma liste, les jeux populaires) ne sont pas prêts et les actualités ne sont pas du tout présentes.

En conclusion, malgré l’avancement actuel j’ai beaucoup aimé travailler sur ce projet, je pense avoir rempli mon principal objectif qui était d’apprendre à coder et mettre en place le plus de choses possibles, autant en back-end qu’en front-end. Je pense que la principale difficulté que j’aurais rencontré aura été les problèmes de base de données, que j’ai eu du mal à détecter et qui m’ont fait perdre plusieurs jours, à côté de ça je pense avoir grandement sous-estimé le temps nécessaire à la mise en place des plusieurs pages en HTML CSS, travail qui a représenté une grande partie du temps de projet, d’autant plus qu’étant l’une des premières étapes, mes compétences étaient très limitées à ce moment-là.

Grâce aux TP en court de formation et au stage le développement de certaines de fonctions s’est avéré plutôt simple, notamment pour les fonctions utilisateur.

Mon principal regret est de ne pas avoir eu le temps de finir le projet, mais étant donné le temps que j’ai eu et mon niveau de compétences je suis conscient que c’était un objectif extrèmement dur à atteindre.

Concernant l’avenir de ce projet je ne pense pas qu’il sera fini un jour, mis à part peut-être durant mon temps libre à titre personnel étant donné la nature fictive de la demande.

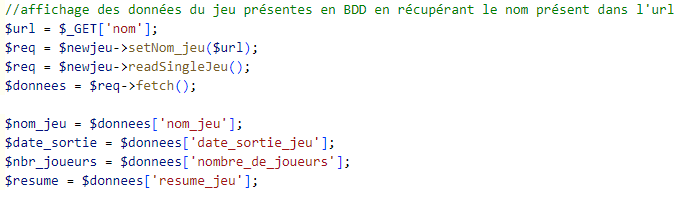
A l’heure actuelle j’ai décidé de poursuivre mes études vers un bac+5, plus précisément un bachelor en développement suivi d’un mastère d’expert en développement web, le tout en alternance. J’estime que si je veux faire du milieu du développement mon métier il est plus avantageux de pouvoir présenter un bac+5.

Et enfin je tiens à adresser mes remerciements tout d’abord à l’équipe pédagogique de l’ADRAR pour ses enseignements et sa précieuse aide durant le projet, ainsi qu’à mes camarades pour l’aide des plus à l’aise d’entre eux en cas de difficulté.

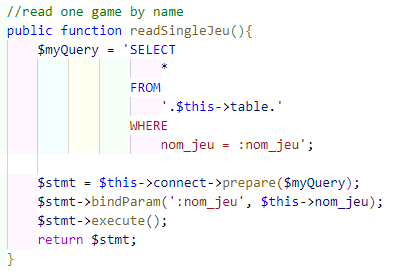
**Annexe**

Je pense qu’il est pertinent de présenter en plus des fonctionnalités ci-dessus l’affichage d’un jeu :

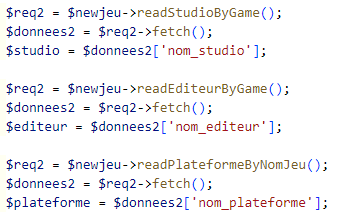
Une fois le lien d’un jeu sélectionné par un utilisateur, le controller récupère le nom du jeu dans l’url de la page et va chercher les données du jeu dans la BDD avec la fonction readSingleGame de la classe Jeu.



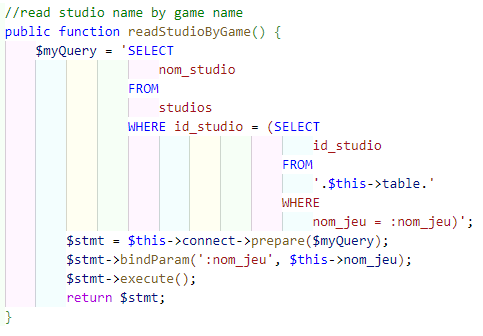
La fonction readSingleGame du modèle :



Une fois ces informations récupérées, il faut aller chercher les informations nécessaires dans les autres tables : nom\_studio dans la table studios, nom\_editeur dans editeur, etc. Pour cela le controller va faire appel aux fonction imbriquées de la classe Jeu qui présentent toutes la même structure, avec un inner join si une table d’association est présente.



readStudioByGame du modèle jeu :



readPlateformeByNomJeu avec l’inner join vers la table d’association distribuer :



Une fois toutes les données récupérées et insérées dans des variables, la vue va pouvoir les afficher en récupérant ces variables depuis le controller.

