

# Rotten potatoes

Site communautaire de notation et de critique de jeux vidéos

Par Sébastien Leyrissoux

## Table des matières

<b>Introduction</b> .....	3
<b>REAC</b> .....	4
<b>Analyse du besoin</b> .....	4
Présentation de l'entreprise.....	4
Contexte – besoin.....	5
Contraintes techniques .....	5
Use case .....	6
Diagrammes d'activité et de séquence.....	7
Maquettes .....	16
<b>Conception</b> .....	18
MCD .....	18
MLD .....	18
Arborescence .....	19
Outils techniques utilisés .....	20
Fonctionnalités .....	20
<b>Conclusion</b> .....	36
<b>Annexe</b> .....	38

## 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.

The logo for Rotten Tomatoes is displayed in a bold, orange-red font. The word "Rotten" is on the top line, and "Tomatoes" is on the bottom line. The letter 'o' in "Rotten" is replaced with a white asterisk-like flower shape. The letter 'o' in "Tomatoes" is replaced with a stylized tomato icon, complete with a green stem and a small leaf.

## 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 imparti 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.

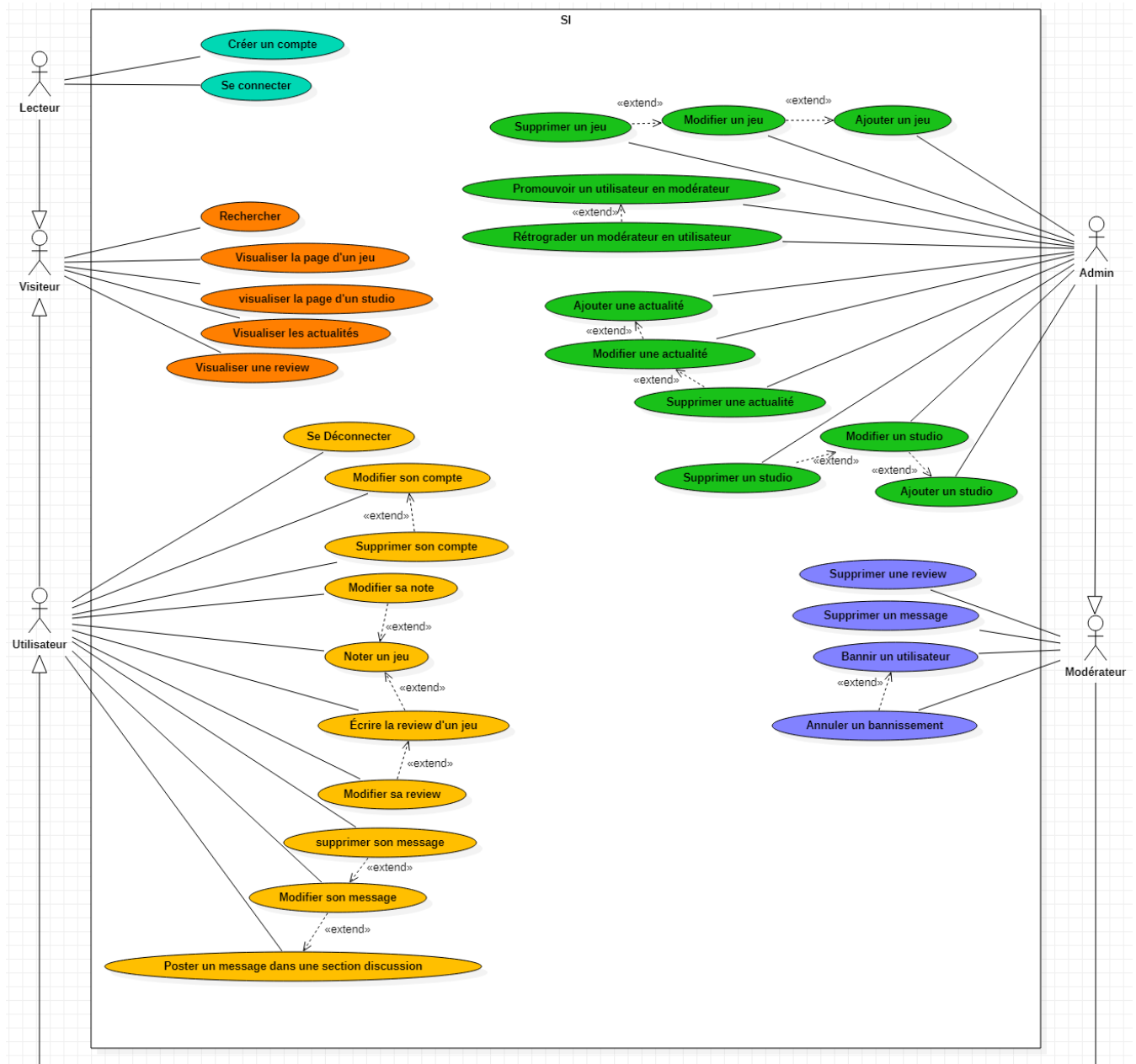
Français Anglais : Rotten tomatoes étant un site anglophone, les langues Anglais et Français serait nécessaire, mais vont nécessiter une traduction entière du site.

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.

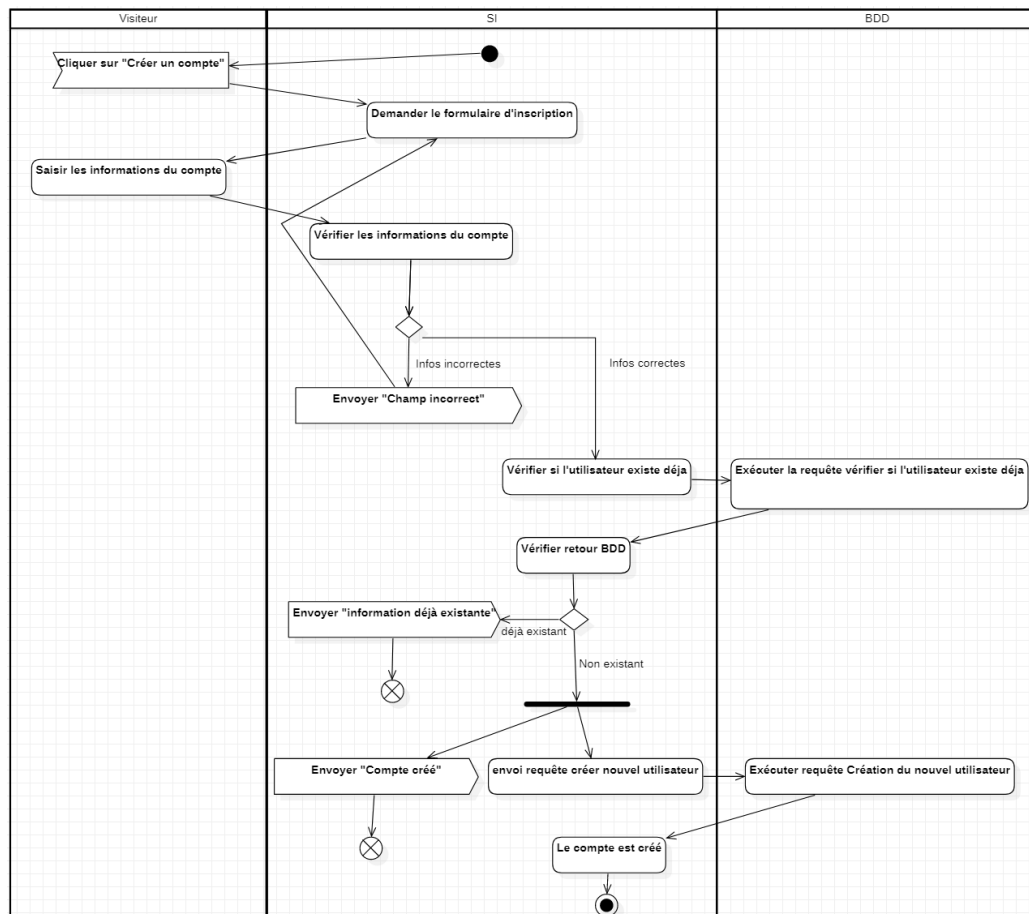
Dans la partie suivante seront présentés les diagrammes d'activité et de séquence des fonctions suivantes : création de compte, connexion, déconnexion, modification du compte et suppression du compte.

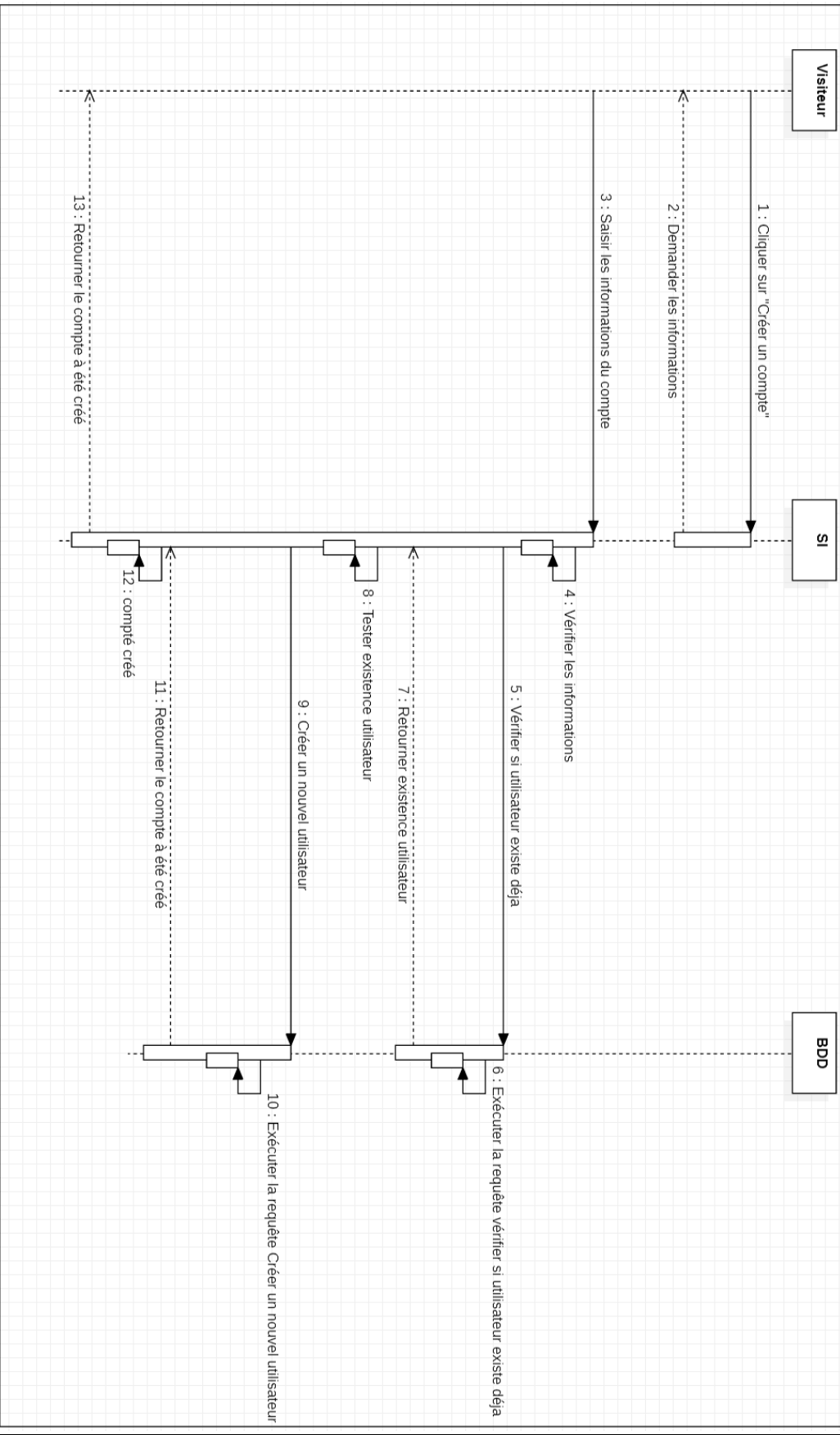
## Diagrammes d'activité et de séquence

### Création de compte

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

Le diagramme suivant est présent en annexe si trop petit.





Scénario alternatif :  
(4) Les informations sont incorrectes  
4.1 Le système d'information indique que les informations sont incorrectes (msg d'erreur)  
4.2 retour (2) redemande les infos  
4.3 l'utilisateur saisit les informations  
On considère dans le scénario nominal que les informations sont correctes.

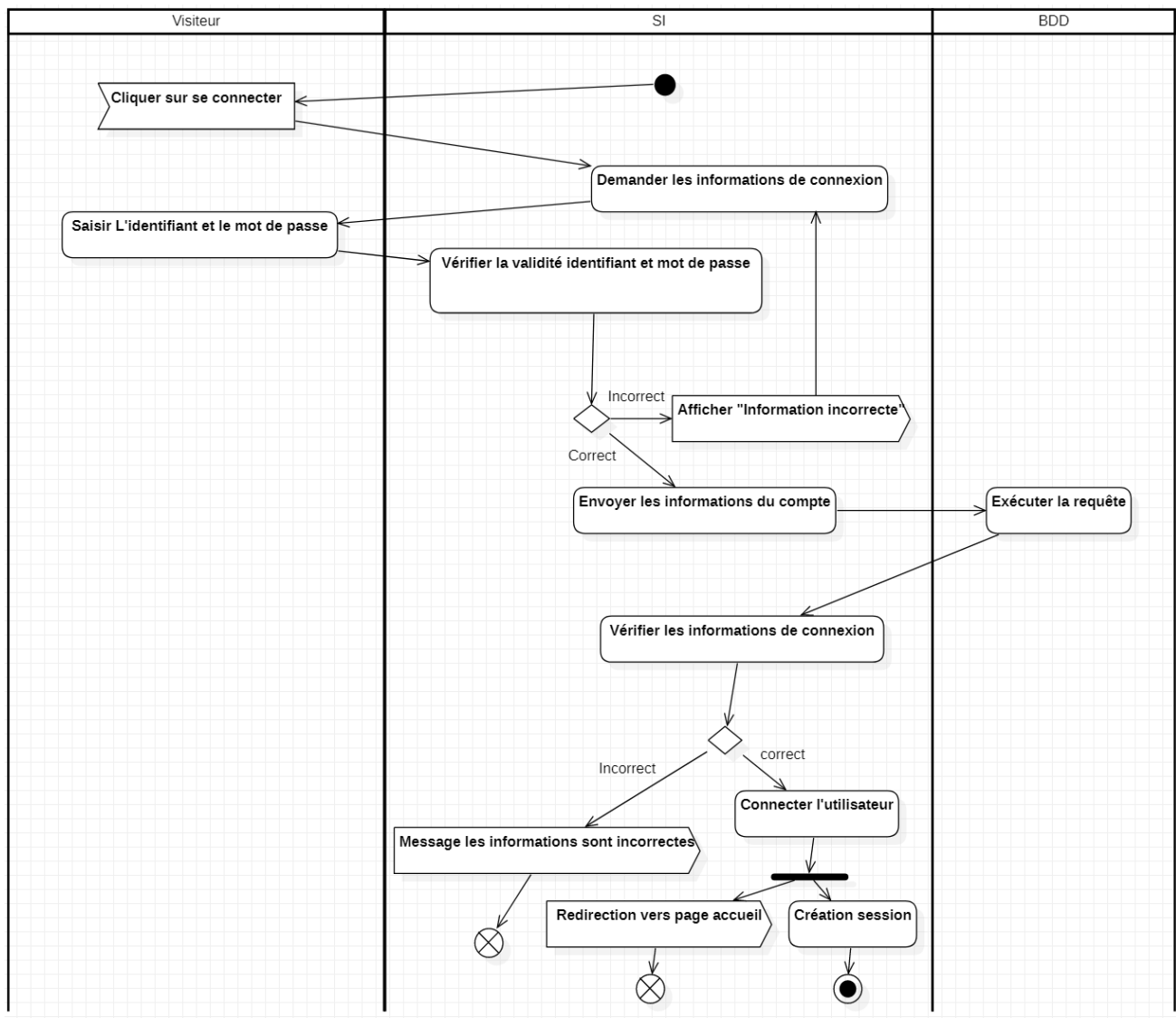
Scénario d'erreur :  
(5) La base de données est HS  
5.1 Le système indique que le serveur est HS (msg d'erreur)  
Le scénario s'arrête à ce point

Scénario d'erreur :  
(8) L'utilisateur existe déjà  
8.1 Le système d'information indique que l'utilisateur est déjà présent (msg d'erreur)  
Le scénario s'arrête à ce point

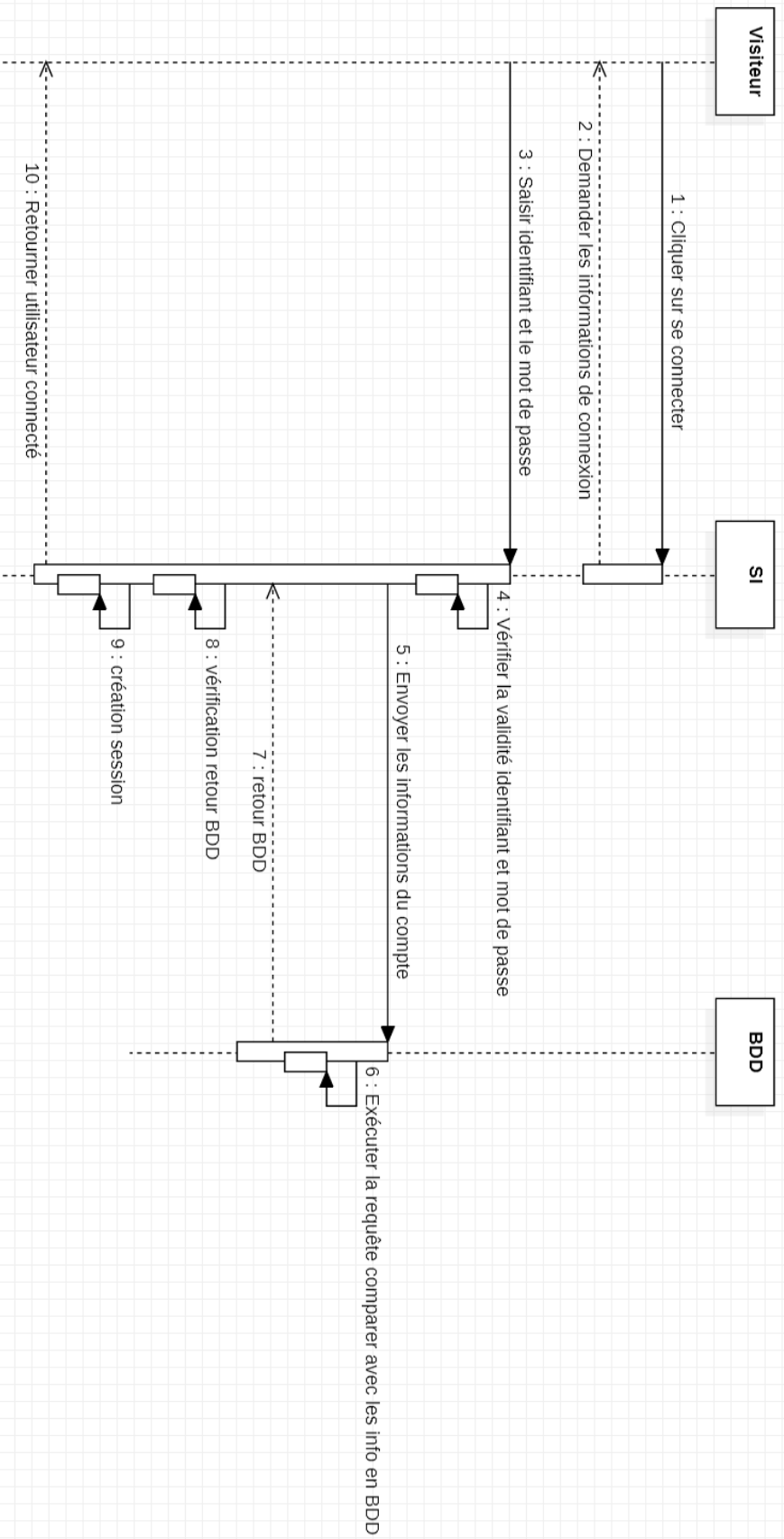


## Connexion

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



# sd Connexion



## Scénario alternatif :

- (4) Les informations sont incorrectes
  - 4.1 Le système d'information indique que les informations sont incorrectes (msg d'erreur)
  - 4.2 retour (2) redemande les infos
  - 4.3 l'utilisateur saisit les informations
- On considère dans le scénario nominal que les informations sont correctes.

## Scénario d'erreur :

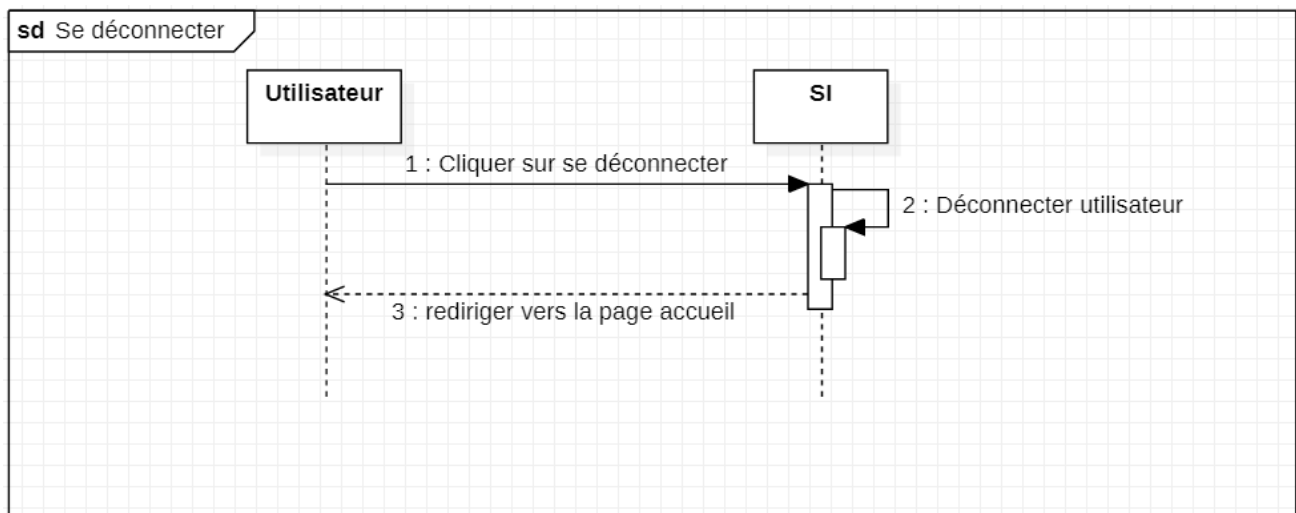
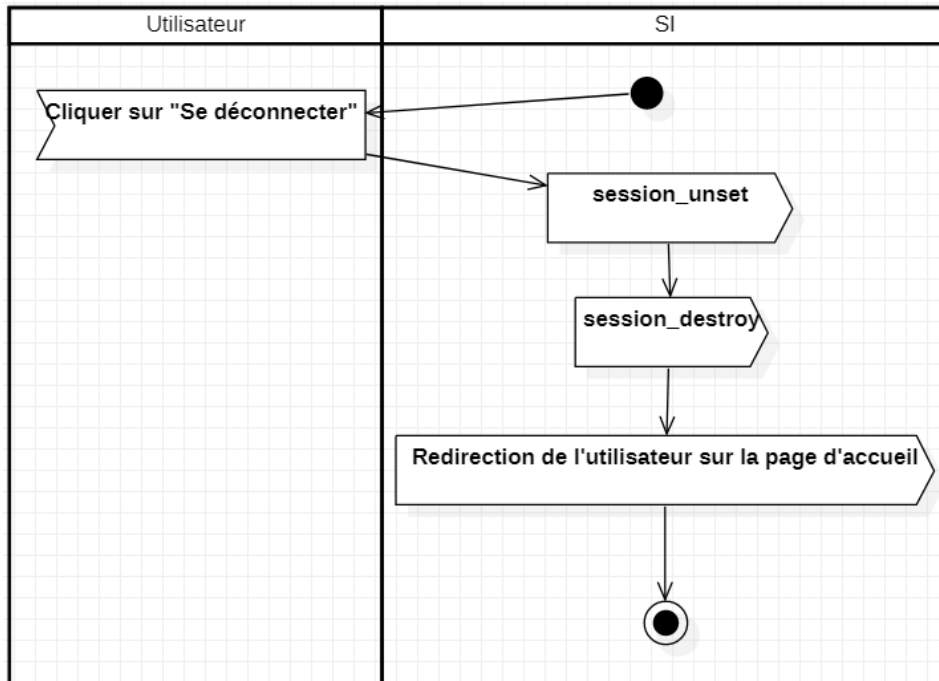
- (5) La base de données est HS
  - 5.1 Le système indique que le serveur est HS (msg d'erreur)
- Le scénario s'arrête à ce point

## Scénario d'erreur :

- (8) Les informations sont incorrectes
  - 8.1 Le système d'information indique que les informations sont incorrectes (msg d'erreur)
- Le scénario s'arrête à ce point

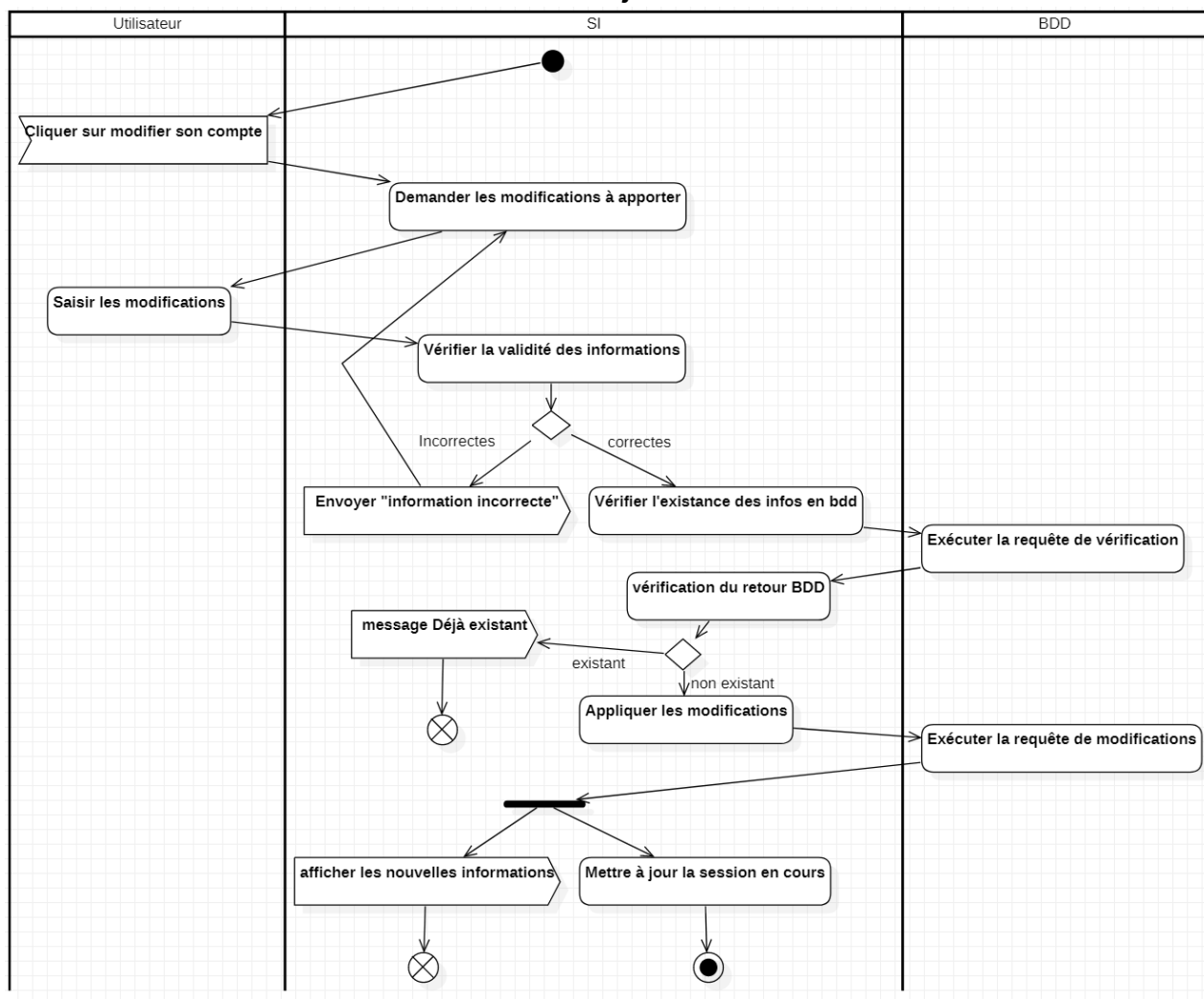
## Déconnexion

La déconnexion avec `session_unset` et `session_destroy` pour effacer les données de la session utilisateur du navigateur.

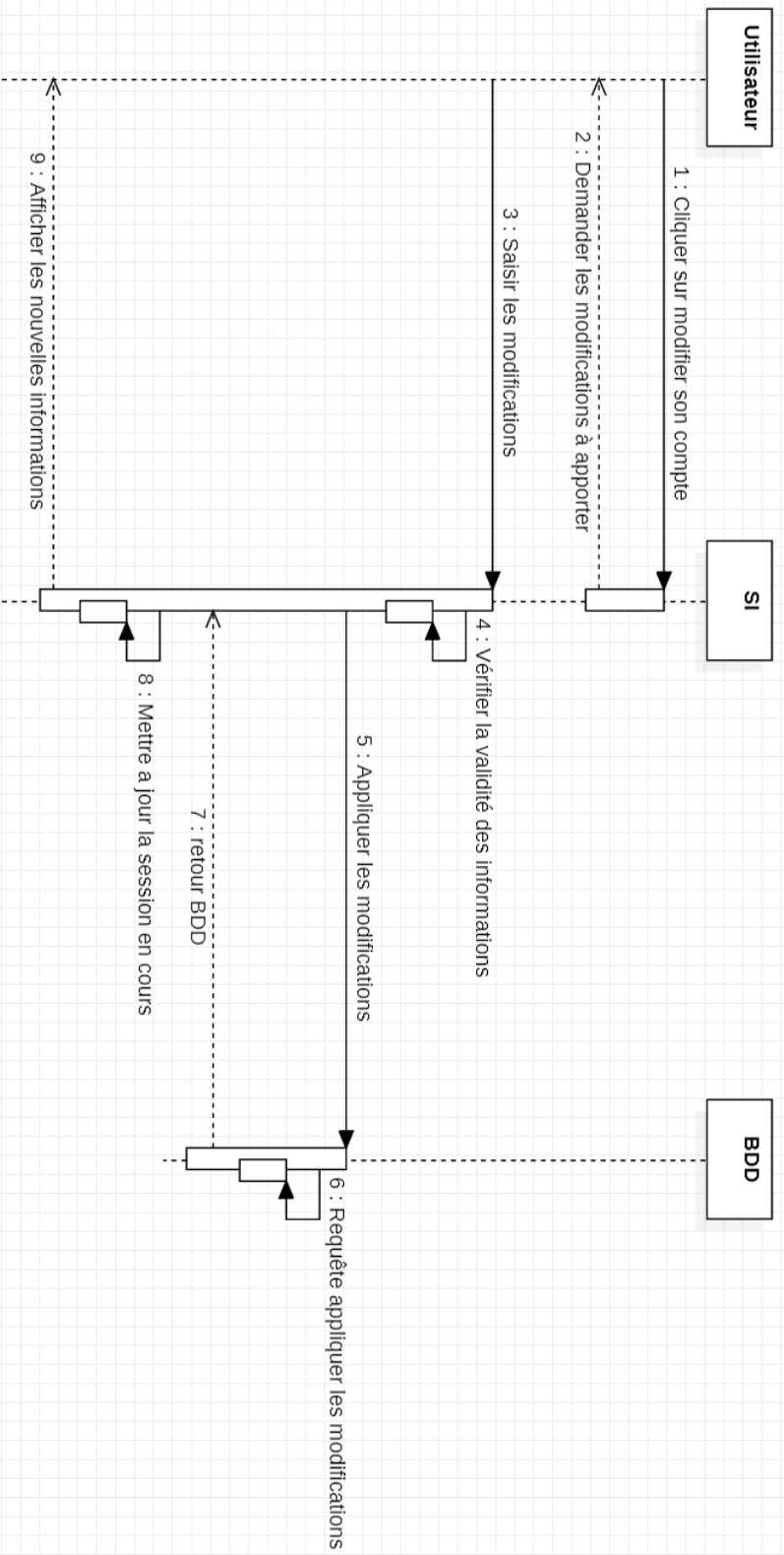


## Modification

La modification, avec la validation des données entrées 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.



# sd Modifier son compte



## Scénario alternatif :

- (4) Les informations sont incorrectes
  - 4.1 Le système d'information indique que les informations sont incorrectes (msg d'erreur)
  - 4.2 retour (2) redemande les infos
  - 4.3 l'utilisateur saisit les informations
- On considère dans le scénario nominal que les informations sont correctes.

## Scénario d'erreur :

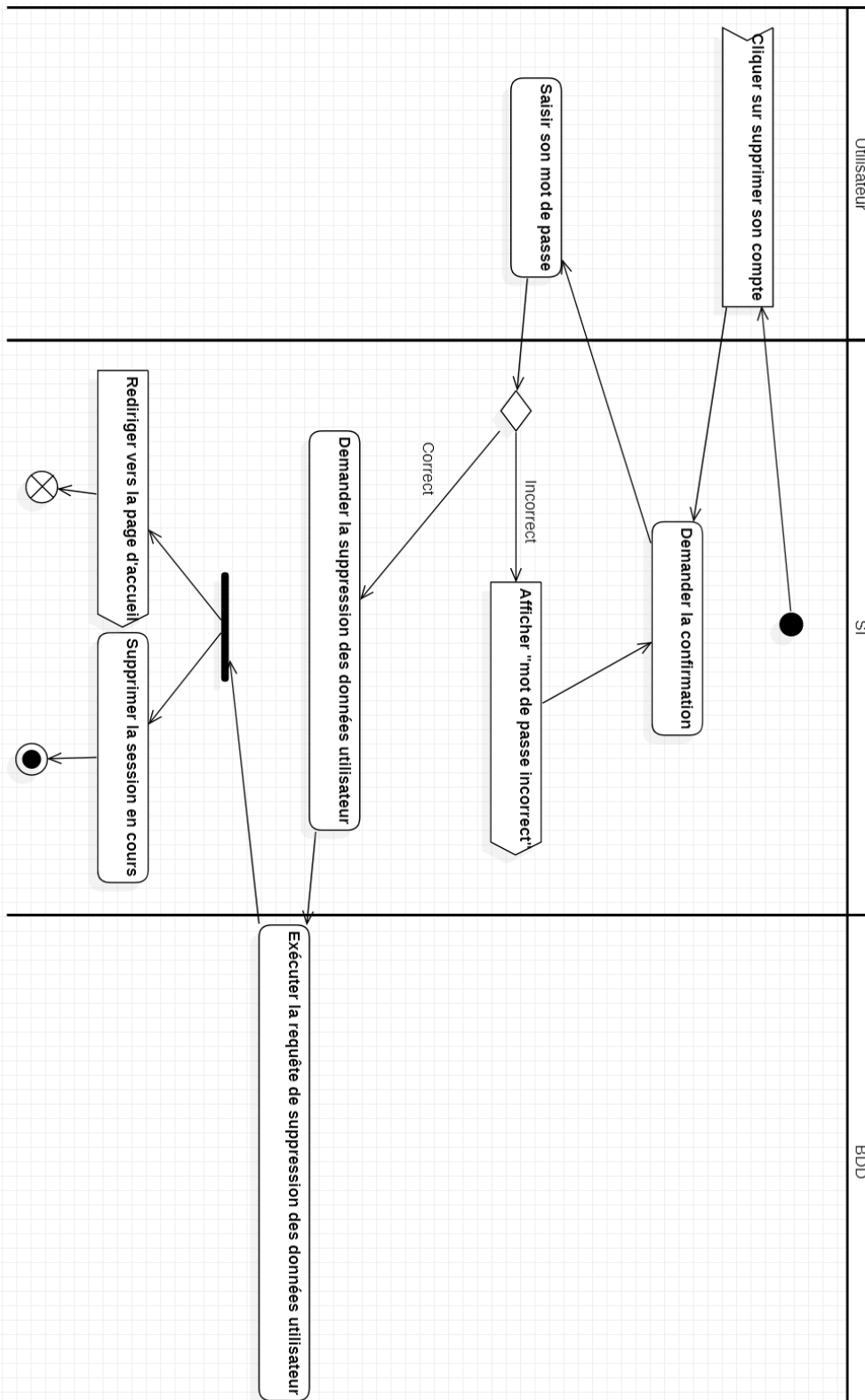
- (5) La base de données est hs
  - 5.1 Le système indique que le serveur est HS (msg d'erreur)
- Le scénario s'arrête à ce point

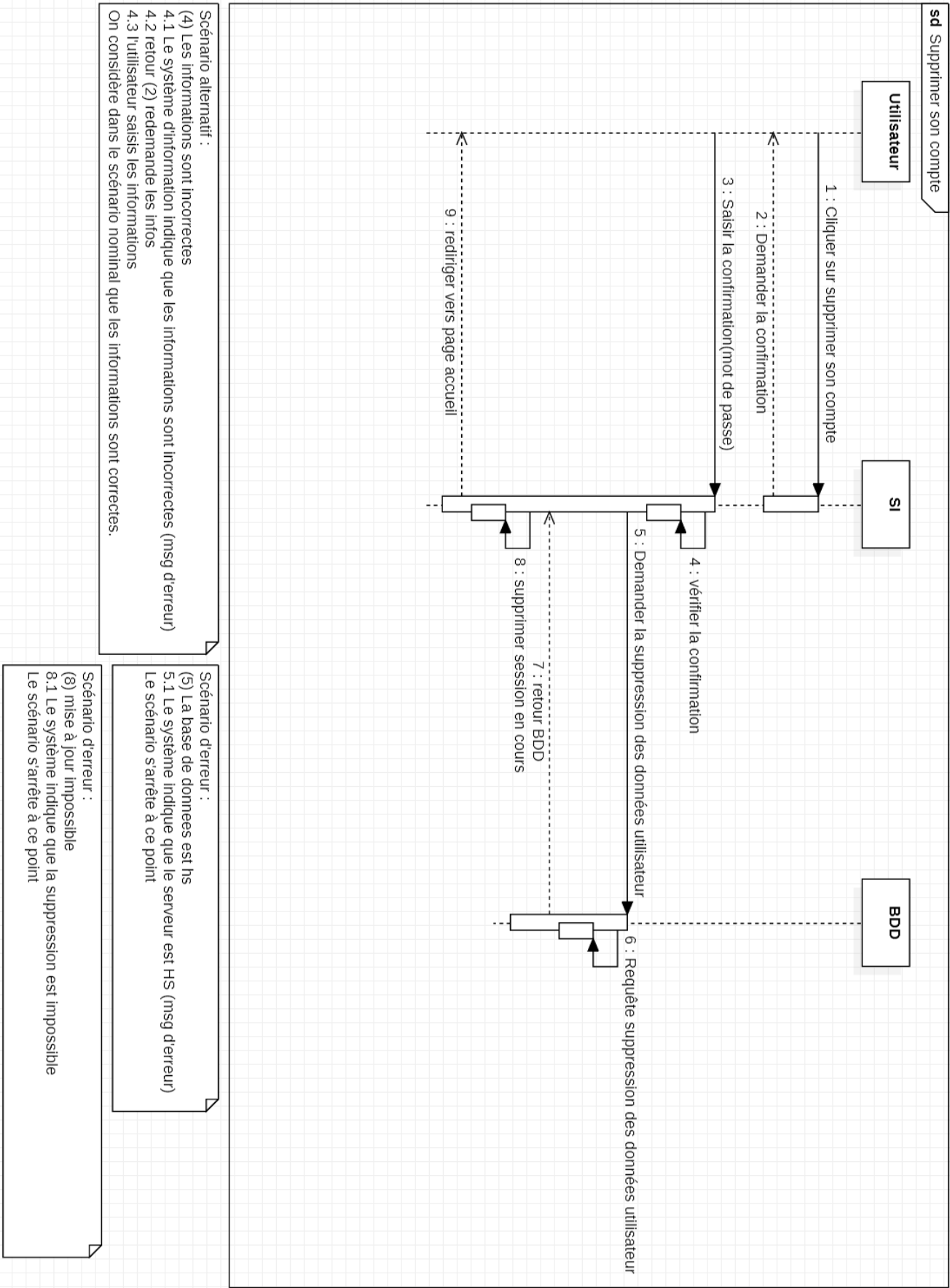
## Scénario d'erreur :

- (8) mise à jour impossible
  - 8.1 Le système indique que la mise à jour est impossible
- Le scénario s'arrête à ce point

## Suppression

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

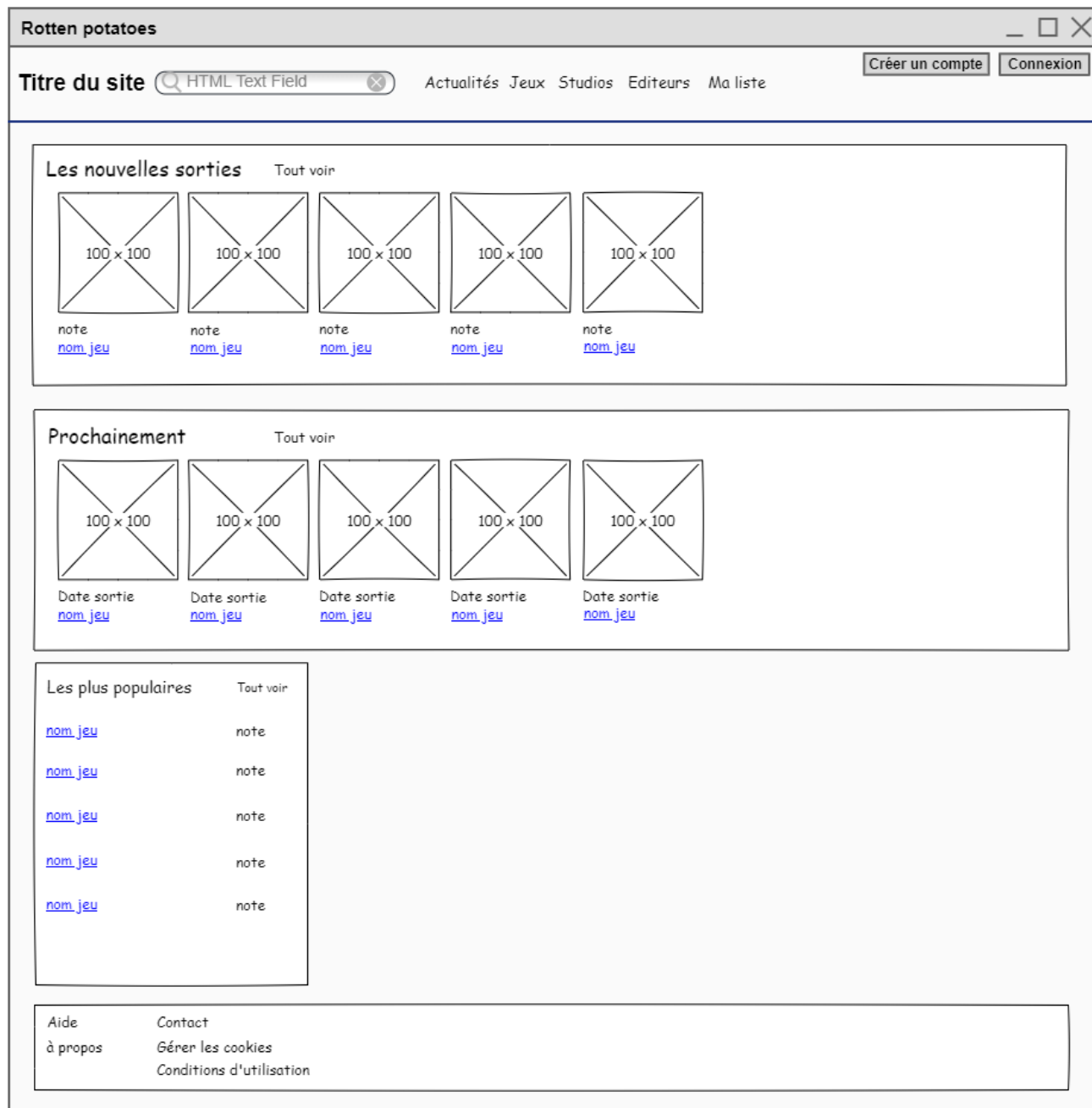




## Maquettes

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



Rotten potatoes

Titre du site

Actualités
Jeux
Studios
Editeurs
Ma liste

Créer un compte
Connexion

Les plus populaires

[nom jeu](#)

Tout voir

note

[nom jeu](#)

note

[nom jeu](#)

note

[nom jeu](#)

note

[nom jeu](#)

note

Jaquette

116 x 174

Nom jeu

note

Studio

Editeur

Date de sortie

Plateforme

Genre

Multijoueur

Nombre de joueurs

Resume

Résumé

Vidéos

iframe vidéo

Photos

71 x 100

71 x 100

71 x 100

Commentaires

pseudo date

Commentaire

Aide

Contact

à propos

Gérer les cookies

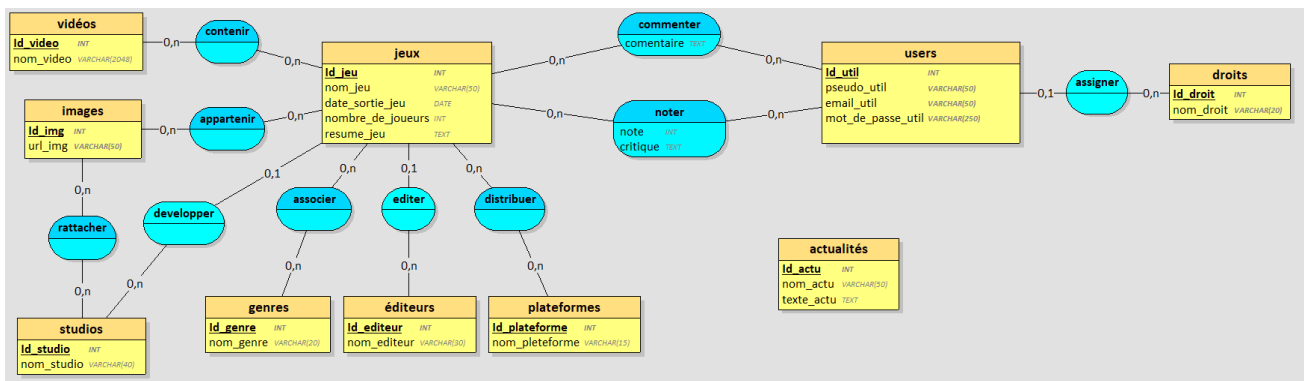
Conditions d'utilisation

# Conception

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

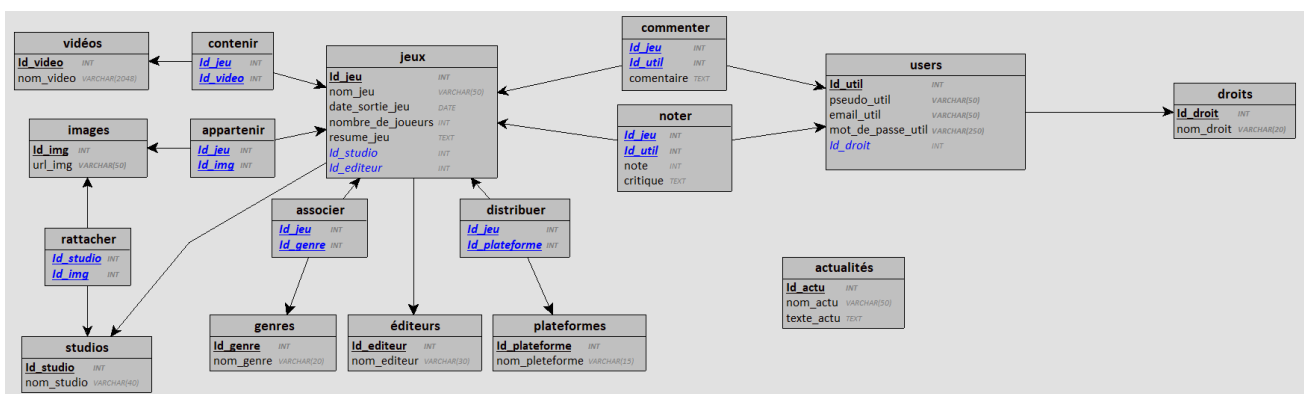
## MCD

Le modèle conceptuel de données de la BDD, où l'on peut voir l'entité principale du site, jeux, reliée aux autres entités 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 l'entité user qui contiennent des champs. Nous avons enfin l'entité studio reliée à l'entité images, droits relié à users et actualité toute seule.

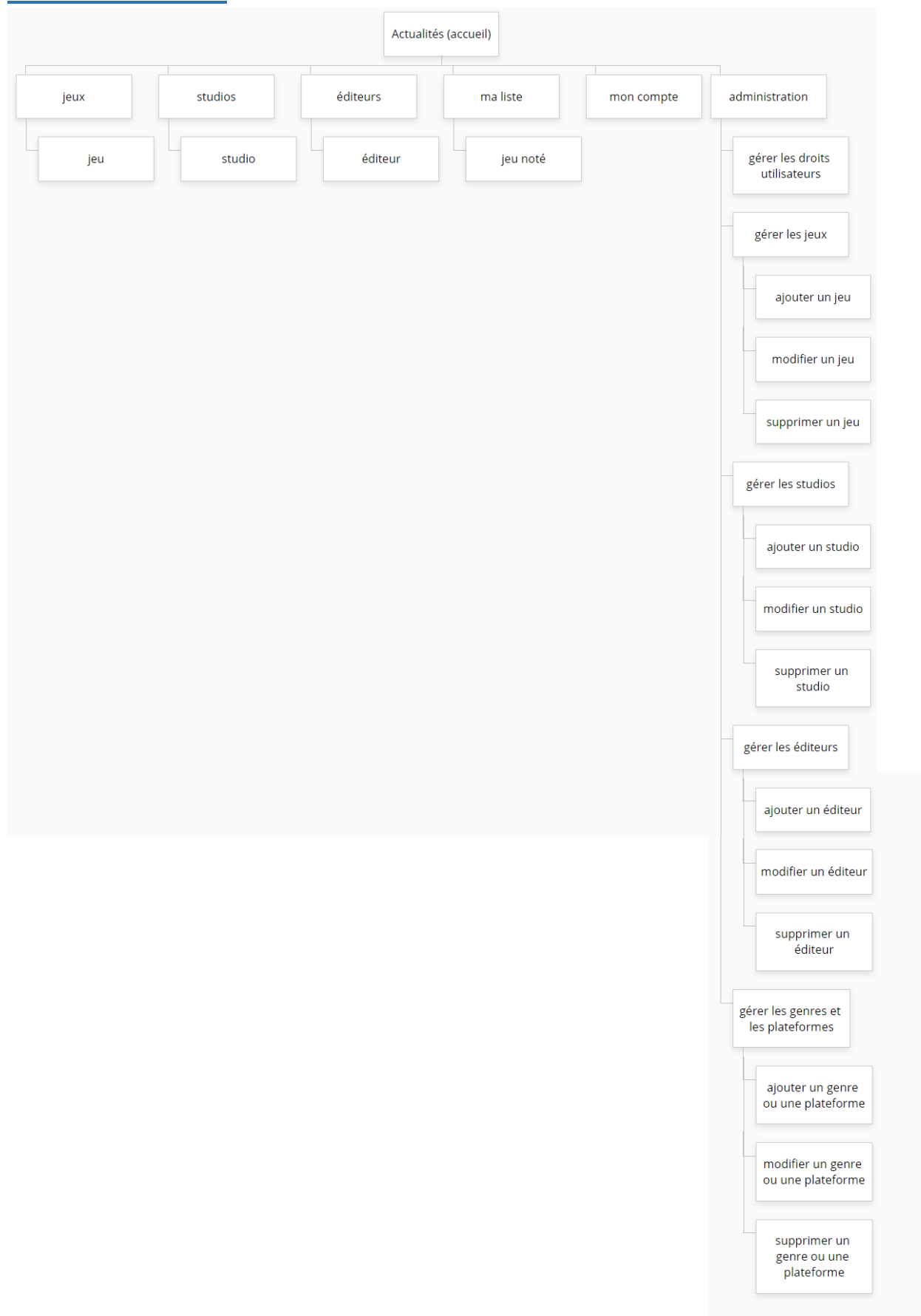


## MLD

Le modèle logique de données 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 absorptions de clé étrangère.



# Arborescence



## Outils techniques utilisés

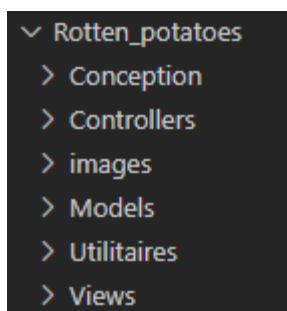
Pour mettre en œuvre ce projet, la partie front-end sera réalisée en **HTML CSS**, avec des éléments en **Javascript** pour gérer certains éléments dynamiques. La partie back-end sera quant à elle réalisée avec une structure suivant le modèle **MVC** en **PHP**. Le tout sera connecté à une base de donnée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

Le projet a été conçu en suivant le modèle MVC : le modèle contient les fonctions d'accès à la BDD, la vue qui gère l'affichage et les interactions de l'utilisateur, le controller qui synchronise la vue et le modèle, en exécutant les actions demandées par l'utilisateur et les fonctions du modèle quand nécessaire. Cela nous donne un répertoire de travail comme cela :



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 :

```
class Bdd{
    // je crée les attributs de ma classe de config db
    private $host = "127.0.0.1";
    private $dbName = "rottenpotatoes";
    private $userName = "root";
    private $password = "";

    // l'attribut suivant sera lui en public
    // pour permettre aux autres classes d'accéder à la connexion
    // de la fonction de connexion à la base de données
    public $connect;

    // je crée la fonction pour appeler le système de connexion à ma BDD
    // celle-ci va permettre d'injecter mes attributs en private dans
    // l'attribut qui est en public.
    public function getConnexion() {
        $this->connect = null;
        try{
            $this->connect = new PDO(
                "mysql:host=".$this->host.";
                dbname=".$this->dbName.",
                $this->userName,
                $this->password);
            $this->connect->exec('set names utf8');
        } catch(PDOException $exception) {
            echo "Database could not be connected:".$exception->getMessage();
        }
        return $this->connect;
    }
}
```

Sachant que le projet étant en phase de pré production et étant géré sur un serveur local aucun mot de passe n'a été mis pour faciliter l'accès.

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 slashes et htmlspecialchars pour

convertir les caractères spéciaux comme les quotes ou les chevrons en entités html.

```
//fonction regroupant trois fonctions de sécurité : : trim, stripslashes et
//htmlspecialchars. je m'assure ainsi qu'il n'y ai pas d'espaces ou de caractères spéciaux
//dans les champs du formulaire. De cette manière on se protège de toute faille XSS
public function valid_donnees($donnees){
    $donnees = trim($donnees);
    $donnees = stripslashes($donnees);
    $donnees = htmlspecialchars($donnees);
    return $donnees;
}
```

Pour les fonctionnalités suivantes, création de compte, connexion, déconnexion, un tutoriel en anglais a servis en plus des connaissances déjà acquises sur le sujet (<https://www.tutorialrepublic.com/php-tutorial/php-mysql-login-system.php>) et sera traduit ici :

## Building the Registration System

In this section we'll build a registration system that allows users to create a new account by filling out a web form. But, first we need to create a table that will hold all the user data.

### Step 1: Creating the Database Table

Execute the following SQL query to create the *users* table inside your MySQL database.

La première section concerne la création du système de création de compte en utilisant un formulaire, avec en premier lieu la création de la table sql pour stocker les utilisateurs.

### Step 2: Creating the Config File

After creating the table, we need create a PHP script in order to connect to the MySQL database server. Let's create a file named "config.php" and put the following code inside it.

On a la requête pour créer une table utilisateur classique suivi du fichier de connexion à la BDD, cette partie étant déjà réalisée elle a été inutile à l'avancée du projet.

### Step 3: Creating the Registration Form

Let's create another PHP file "register.php" and put the following example code in it. This example code will create a web form that allows user to register themselves.

This script will also generate errors if a user tries to submit the form without entering any value, or if username entered by the user is already taken by another user.

In the above example, we have used the PHP's inbuilt `password_hash()` function to create a password hash from the password string entered by the user (*line no-78*). This function creates a password hash using a strong one-way hashing algorithm. It also generates and applies a random salt automatically when hashing the password; this basically means that even if two users have the same passwords, their password hashes will be different.

At the time of login we'll verify the given password with the password hash stored in the database using the PHP `password_verify()` function, as demonstrated in the next example.

We've used the Bootstrap framework to make the [form layouts](#) quickly and beautifully. Please, checkout the [Bootstrap tutorial](#) section to learn more about this framework.

L'étape suivante concerne la création de compte, avec le fichier qui contient le formulaire et la vérification des données envoyées par l'utilisateur, pour avoir des messages d'erreurs si le formulaire est envoyé avec un champ vide ou si l'utilisateur essaye d'utiliser un pseudo déjà utilisé, avec une utilisation des requêtes préparées pour l'envoi en BDD sécurisé. Le tutoriel précise aussi l'utilisation de la fonction `password_hash` pour sécuriser le mot de passe.

La partie front du formulaire a été stylisée avec bootstrap.

## Building the Login System

In this section we'll create a login form where user can enter their username and password. When user submit the form these inputs will be verified against the credentials stored in the database, if the username and password match, the user is authorized and granted access to the site, otherwise the login attempt will be rejected.

### Step 1: Creating the Login Form

Let's create a file named "login.php" and place the following code inside it.

Il y a ensuite la partie connexion, qui consiste en un formulaire de connexion avec une comparaison des champs dans la BDD pour autoriser ou refuser la connexion.

Le code contient le formulaire stylisé avec bootstrap ainsi que la comparaison avec la BDD, les messages d'erreurs en cas d'information erronée et le stockage des informations de l'utilisateur dans la variable `$_SESSION` de PHP pour créer la session.

If data comes from external sources like form filled in by anonymous users, there is a risk that it may contain malicious script indented to launch cross-site scripting (XSS) attacks. Therefore, you must escape this data using the PHP `htmlspecialchars()` function before displaying it in the browser, so that any HTML tag it contains becomes harmless.

For example, after escaping special characters the string `<script>alert("XSS")</script>` becomes `&lt;script&gt;alert("XSS")&lt;/script&gt;` which is not executed by the browser.

La seconde étape de cette partie explique comment créer la page de bienvenue qui affiche le pseudo de l'utilisateur en utilisant la fonction `htmlspecialchars` pour se protéger des injections XSS.



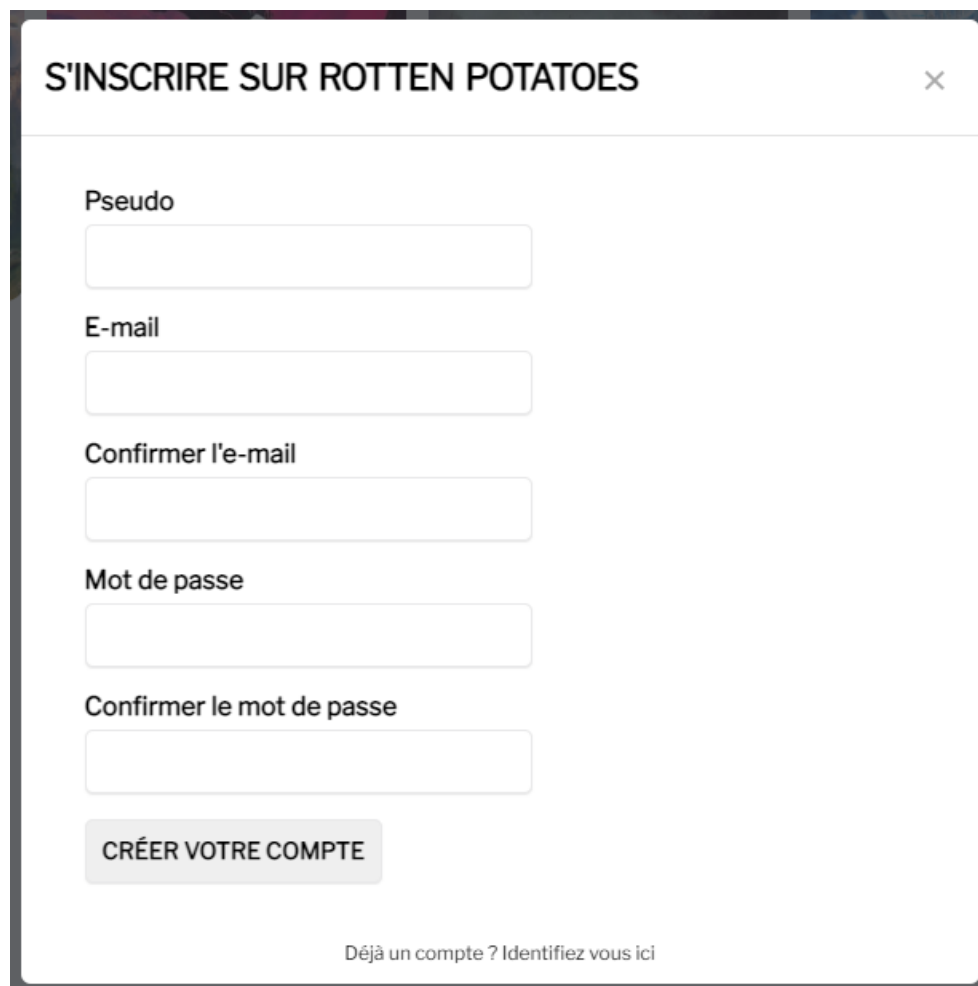
### Step 3: Creating the Logout Script

Now, let's create a "logout.php" file. When the user clicks on the log out or sign out link, the script inside this file destroys the session and redirect the user back to the login page.

La troisième étape montre la création de la déconnexion, avec un script pour détruire la session en cours et rediriger vers la page de connexion.

### Création de compte

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



The image shows a modal window titled "S'INSCRIRE SUR ROTTEN POTATOES" with a close button (X) in the top right corner. The form contains five input fields with labels: "Pseudo", "E-mail", "Confirmer l'e-mail", "Mot de passe", and "Confirmer le mot de passe". Below the fields is a button labeled "CRÉER VOTRE COMPTE". At the bottom of the modal, there is a link that says "Déjà un compte ? Identifiez vous ici".

```

<!--Modale création de compte-->
<div id="creaCompte" class="modal">
  <div id="modalCrea" class="modal-container">
    <header>
      <span id="close">&times;</span>
      <h3>S'inscrire sur Rotten Potatoes</h3>
    </header>

    <div>
      <form id="createCompte" method="POST" action="">
        <ul>
          <li>
            <label for="pseudo">Pseudo </label>
            <input type="text" name="pseudo" maxlength="20" id="pseudo" required/>
          </li>
          <li>
            <label for="email">E-mail </label>
            <input type="email" name="email" id="email" required/>
          </li>
          <li>
            <label for="confEmail">Confirmer l'e-mail </label>
            <input type="email" name="confEmail" id="confEmail" required/>
          </li>
          <li>
            <label for="mdp">Mot de passe </label>
            <input type="password" name="mdp" minlength="8" maxlength="25" id="mdp" required/>
          </li>
          <li>
            <label for="confirmMdp">Confirmer le mot de passe </label>
            <input type="password" name="confirmMdp" minlength="8" maxlength="25" id="confirmMdp" required/>
          </li>
          <li>
            <p><strong><?php echo $log; ?></strong></p>
            <input type="submit" name="Crea" value="Créer votre compte">
          </li>
        </ul>
      </form>
    </div>
    <footer>
      <p>Déjà un compte ? Identifiez vous <a id="connexionCrea">ici</a></p>
    </footer>
  </div>
</div>
<!------->

```

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.

```

if(isset($_POST['pseudo']))
  && isset($_POST['email'])
  && isset($_POST['confEmail'])
  && isset($_POST['mdp'])
  && isset($_POST['confirmMdp'])) {

  // je commence par créer des variables qui vont stocker les données envoyées par l'utilisateur
  // en sécurisant les données grâce aux trois composant de ma fonction valid_donnees : trim, stripslashes et
  // htmlspecialchars. je m'assure ainsi qu'il n'y ai pas d'espaces ou de caractères spéciaux dans les champs
  // du formulaire. De cette manière on se protège de toute faille XSS

  $pseudoUser = $verif->valid_donnees($_POST['pseudo']);
  $mailUser = $verif->valid_donnees($_POST['email']);
  $confMail = $verif->valid_donnees($_POST['confEmail']);
  $passwordUser = $verif->valid_donnees($_POST['mdp']);
  $confPassword = $verif->valid_donnees($_POST['confirmMdp']);

```

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.

```
if($passwordUser !== $confPassword) {  
    //Si différents je passe une valeur à la variable $log pour traiter cette erreur  
    //$log = 'Les mots de passe doivent correspondre';  
    echo "<script>alert('Les mots de passe doivent correspondre')</script>";  
  
if(!filter_var($mailUser, FILTER_VALIDATE_EMAIL)){  
    //Si ça ne correspond pas à un mail je passe une valeur à la variable $log pour traiter cette erreur  
    //$log = "L'E-mail est incorrect";  
    echo "<script>alert('L'E-mail est incorrect')</script>";  
}
```

Si ces étapes 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.

```
$newUser->setPseudo_user($pseudoUser);  
$newUser->setEmail_user($mailUser);  
$newUser->setId_droit(0);  
// pour l'affectation du mot de passe je vais également utiliser la fonction de hash de BCRYPT  
// pour hacher le mot de passe.  
$newUser->setPassword_user(password_hash($passwordUser, PASSWORD_BCRYPT));
```

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.

```
$checkMail = $newUser->verifyMail();  
// puis je stocke dans une variable le résultat du décompte du nombre de ligne  
// de ce retour grâce à la fonction PHP rowCount()  
$nbrLignes = $checkMail->rowCount();  
  
if($nbrLignes > 0) {  
    //$log = "E-mail déjà utilisé";  
    echo "<script>alert('E-mail déjà utilisé')</script>";  
}
```

verifyMail du modèle user :

```
// verify if mail already exist
public function verifyMail() {
    $myQuery = 'SELECT
                *
                FROM
                '.$this->table.'
                WHERE
                email_user = :email_user';

    $stmt = $this->connect->prepare($myQuery);

    $stmt->bindParam(':email_user', $this->email_user);
    $stmt->execute();
    return $stmt;
}
```

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.

```
if($newUser->createUser()){
    $myReturn = $newUser->getSingleUser();
    $nbrUsers = $myReturn->rowCount();

    if($nbrUsers == 0){
        //$log = "Something went wrong, please contact an administrator";
        echo "<script>alert('Something went wrong, please contact an administrator')</script>";
    } else if($nbrUsers >1){
        //$log = "Compte déjà présent";
        echo "<script>alert('Compte déjà présent')</script>";
    } else if ($nbrUsers == 1) {
        echo '<script>alert("Compte créé avec succès")</script>';
    }
}
```

createUser du modèle user :

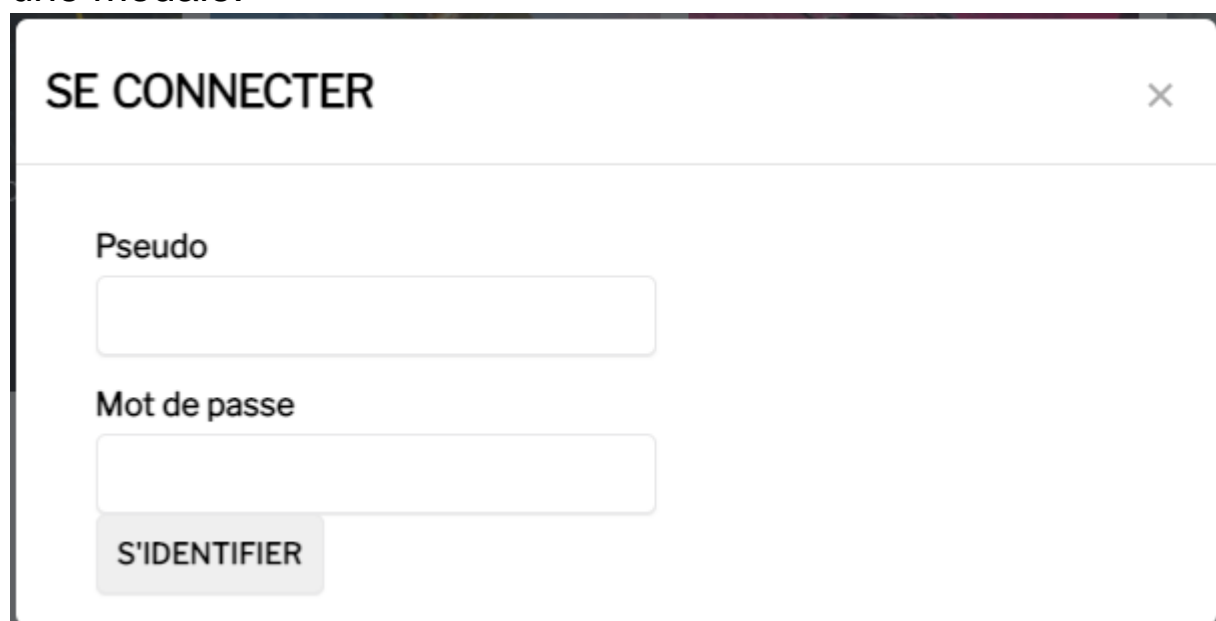
```
//create
public function createUser(){
    $myQuery = 'INSERT INTO
                ' . $this->table . '
                SET
                pseudo_user = :pseudo_user,
                email_user = :email_user,
                password_user = :password_user,
                id_droit = :id_droit';

    $stmt = $this->connect->prepare($myQuery);
    $stmt->bindParam(':pseudo_user', $this->pseudo_user);
    $stmt->bindParam(':email_user', $this->email_user);
    $stmt->bindParam(':password_user', $this->password_user);
    $stmt->bindParam(':id_droit', $this->id_droit);
    return $stmt->execute();
}
```

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.



The image shows a modal window titled "SE CONNECTER" with a close button (X) in the top right corner. Inside the modal, there are two input fields: "Pseudo" and "Mot de passe". Below the "Mot de passe" field is a button labeled "S'IDENTIFIER".

Une fois la présence des champs vérifiée et la fonction `valid_donnees` utilisée, 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. L'appel de la fonction se trouve dans un `try catch` pour qu'en cas de bug on puisse extraire l'exception et poursuivre l'exécution du code. Une fois la requête exécutée la fonction `fetch` récupère le résultat de la requête.

```
try{
    $user->setPseudo_user($pseudo);
    $user->setPassword_user($password);

    $req = $user->getSingleUser();
} catch (Exception $e) {
    die('Erreur : ' . $e->getMessage());
}

//vérification de l'existence de l'utilisateur
$verif = $req->fetch();
```

`getSingleUser` du modèle `user` :

```
//read single user by pseudo
public function getSingleUser(){
    $myQuery = 'SELECT
        *
    FROM
        '.$this->table.'
    WHERE
        pseudo_user = :pseudo_user';

    $stmt = $this->connect->prepare($myQuery);
    $stmt->bindParam(':pseudo_user', $this->pseudo_user);
    $stmt->execute();
    return $stmt;
}
```

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).

```

if($verif == true) {
    // je compare mot de passe reçu avec celui de la BDD
    // si il ne sont pas identiques je retourne un message d'erreur
    if(!password_verify($password, $verif['password_user'])) {
        //$log = "Erreur dans votre mot de passe, veuillez recommencer";
        echo "<script>alert('Erreur dans votre mot de passe, veuillez recommencer')</script>";
    }
}

```

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.

```

} else if (password_verify($user->getPassword_user(), $verif['password_user'])) {
    //Je renseigne les champs pertinents dans la variable $_SESSION de PHP pour qu'ils soit disponibles sur toutes les pages
    $_SESSION['id'] = $verif['id_user'];
    $_SESSION['pseudo'] = $verif['pseudo_user'];
    $_SESSION['id_droit'] = $idDroit['id_droit'];

    //redirection vers l'accueil une fois connecté
    header("location: ../Views/accueil_view.php");
}

```

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.

```

//Gestion de l'affiche du menu de connexion et d'accès à l'espace mon compte quand connecté ou déconnecté
if(isset($_SESSION['id'])){
    $pseudo = $_SESSION['pseudo'];
    $account = '<a href="/monCompte_view.php">'.$pseudo.'</a>';
    $connexion = '<form action="" method="POST"> <input type="submit" id="deconnexion" name="deconnexion" value="Déconnexion"></form>';
} else {
    $account = '<a id="creaCpt">Créer un compte</a>';
    $connexion = '<a id="connexion">Se connecter</a>';
}

```

## 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.

```
//destruction des varriables de session et redirection vers l'accueil lors de la déconnexion
if(isset($_POST['deconnexion'])){
    session_unset();
    session_destroy();
    header("location: ../Views/accueil_view.php");
}
```

## 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.

```
if(isset($_SESSION['id'])){
    //récupération des données de l'utilisateur connecté via une fonction du modèle utilisateur et du pseudo stocké dans la session
    $user->setPseudo_user($_SESSION['pseudo']);
    $req = $user->getSingleUser();
    $donnees = $req->fetch();

    $user->setIdUser($_SESSION['id']);
    $user->setPseudo_user($donnees['pseudo_user']);
    $user->setEmail_user($donnees['email_user']);
    $user->setPassword_user($donnees['password_user']);
    $user->setId_droit($donnees['id_droit']);

    //affichage des infos du compte
    $pseudo = $user->getPseudo_user();
    $email = $user->getEmail_user();
}
```

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

```
//Variable contenant l'affichage des informations du compte en HTML pour avoir un affichage uniquement quand l'utilisateur est connecté
$compte = '
    <div>
        <p id="test"><b>Pseudo :</b> '.$pseudo.'</p>
        <a class="btnModal" id="btnModifyPseudo">Modifier</a>
    </div>
    <div>
        <p><b>Email :</b> '.$email.'</p>
        <a class="btnModal" id="btnModifyMail">Modifier</a>
    </div>
    <br>
    <div>
        <a class="btnModal" id="btnModifyPassword">Modifier son mot de passe</a>
    </div>
    <br>
    <div>
        <a class="btnModal" id="btnDelete">Supprimez votre compte</a>
    </div><br>';
```

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.



×

Entrez un nouveau mot de passe :

Mot de passe actuel :

Nouveau mot de passe :

Confirmez le nouveau mot de passe :

Modifier

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.

```
//changement mdp
if (isset($_POST['Mdp']) && isset($_POST['newMdp']) && isset($_POST['newMdpConf'])) {

    $verifPassword = $verif->valid_donnees($_POST['Mdp']);
    $verifNewPassword = $verif->valid_donnees($_POST['newMdp']);
    $verifNewPasswordConf = $verif->valid_donnees($_POST['newMdpConf']);

    // je compare mot de passe reçu avec celui de la BDD
    // si il ne sont pas identiques je retourne un message d'erreur
    if(!password_verify($verifPassword, $donnees['password_user'])) {
        //$log = "Erreur dans votre mot de passe, veuillez recommencer";
        echo "<script>alert('Erreur dans votre mot de passe, veuillez recommencer')</script>";
    } else if($verifNewPassword !== $verifNewPasswordConf) {
        //$log = "la confirmation ne correspond pas au nouveau mot de passe";
        echo "<script>alert('la confirmation ne correspond pas au nouveau mot de passe')</script>";
    } else if ($verifNewPassword !== ""){
        $user->setPassword_user(password_hash($verifNewPassword, PASSWORD_BCRYPT));
        $user->updateUser();
        header("location: http://localhost/Rotten_potatoes/Views/monCompte_view.php");
    } else {
        //$log = "Veuillez entrer un nouveau mot de passe";
        echo "<script>alert('Veuillez entrer un nouveau mot de passe')</script>";
    }
}
```

updateUser du modèle user :

```
//update
public function updateUser(){
    $myQuery = 'UPDATE
                '.$this->table.'
                SET
                pseudo_user = :pseudo_user,
                email_user = :email_user,
                password_user = :password_user,
                id_droit = :id_droit
                WHERE
                id_user = :id_user';

    $stmt = $this->connect->prepare($myQuery);
    $stmt->bindParam(':pseudo_user', $this->pseudo_user);
    $stmt->bindParam(':email_user', $this->email_user);
    $stmt->bindParam(':password_user', $this->password_user);
    $stmt->bindParam(':id_droit', $this->id_droit);
    $stmt->bindParam(':id_user', $this->id_user);
    if($stmt->execute()) {
        return true;
    } else {
        return false;
    }
}
```

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

```
//si la modification réussie, modification du pseudo stocké dans la variable $_SESSION puis redirection vers la page mon compte
$_SESSION['pseudo'] = $user->getPseudo_user();
header("location: http://localhost/Rotten\_potatoes/Views/monCompte\_view.php");
```

## Suppression

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

Supprimez votre compte :

Cette action est permanente, êtes-vous sûrs ?

Votre mot de passe

Supprimer

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.

```
//suppression compte
if(isset($_POST['passwordSuppr'])) {

    $verifPassword = $verif->valid_donnees($_POST['passwordSuppr']);
    if(!password_verify($verifPassword, $donnees['password_user'])) {
        // $log = "Erreur dans votre mot de passe, veuillez recommencer";
        echo "<script>alert('Erreur dans votre mot de passe, veuillez recommencer')</script>";
    } else {
        //si le mot de passe est correct, utilisation de la fonction de utilisateur pour supprimer le compte
        //dans la BDD et suppression de la session avant de rediriger vers l'accueil
        $user->deleteUser();
        session_unset();
        session_destroy();
        header("location: http://localhost/Rotten\_potatoes/Views/accueil\_view.php");
    }
}
```

deleteUser du modèle user :

```
//delete
public function deleteUser(){
    $myQuery = 'DELETE FROM
                ' . $this->table . '
                WHERE
                pseudo_user = :pseudo_user';

    $stmt = $this->connect->prepare($myQuery);
    $stmt->bindParam(':pseudo_user', $this->pseudo_user);
    if($stmt->execute()) {
        return true;
    } else {
        return false;
    }
}
```

## 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 la page 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'ai 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 cours 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 extrêmement 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 master 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`.

```
//affichage des données du jeu présentes en BDD en récupérant le nom présent dans l'url
$url = $_GET['nom'];
$req = $newjeu->setNom_jeu($url);
$req = $newjeu->readSingleJeu();
$donnees = $req->fetch();

$nom_jeu = $donnees['nom_jeu'];
$date_sortie = $donnees['date_sortie_jeu'];
$nbr_joueurs = $donnees['nombre_de_joueurs'];
$resume = $donnees['resume_jeu'];
```

La fonction `readSingleGame` du modèle :

```
//read one game by name
public function readSingleJeu(){
    $myQuery = 'SELECT
                *
                FROM
                '.$this->table.'
                WHERE
                nom_jeu = :nom_jeu';

    $stmt = $this->connect->prepare($myQuery);
    $stmt->bindParam(':nom_jeu', $this->nom_jeu);
    $stmt->execute();
    return $stmt;
}
```

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.

```
$req2 = $newjeu->readStudioByGame();  
$donnees2 = $req2->fetch();  
$studio = $donnees2['nom_studio'];  
  
$req2 = $newjeu->readEditeurByGame();  
$donnees2 = $req2->fetch();  
$editeur = $donnees2['nom_editeur'];  
  
$req2 = $newjeu->readPlateformeByNomJeu();  
$donnees2 = $req2->fetch();  
$plateforme = $donnees2['nom_plateforme'];
```

readStudioByGame du modèle jeu :

```
//read studio name by game name  
public function readStudioByGame() {  
    $myQuery = 'SELECT  
        nom_studio  
    FROM  
        studios  
    WHERE id_studio = (SELECT  
        id_studio  
    FROM  
        '.$this->table.'  
    WHERE  
        nom_jeu = :nom_jeu)';  
    $stmt = $this->connect->prepare($myQuery);  
    $stmt->bindParam(':nom_jeu', $this->nom_jeu);  
    $stmt->execute();  
    return $stmt;  
}
```

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

```
//read platform by game name
public function readPlateformeByNomJeu() {
    $myQuery = 'SELECT
        nom_plateforme
    FROM
        plateformes
    INNER JOIN
        distribuer
    ON
        distribuer.id_plateforme = plateformes.id_plateforme
    WHERE
        distribuer.id_jeu = (SELECT
            id_jeu
        FROM
            '.$this->table.'
        WHERE
            nom_jeu = :nom_jeu)';

    $stmt = $this->connect->prepare($myQuery);
    $stmt->bindParam(':nom_jeu', $this->nom_jeu);
    $stmt->execute();
    return $stmt;
}
```

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







