

# Projet module 151

Documentation de projet

Paccaud Samuel



|                |
|----------------|
| <b>Contenu</b> |
|----------------|

|     |  |    |
|-----|--|----|
| 1   | Analyse .....                              | 3  |
| 1.1 | Usecase .....                              | 3  |
| 1.2 | Les maquettes.....                         | 4  |
| 1.3 | Diagramme d'activités .....                | 6  |
| 1.4 | Diagramme de séquence système .....        | 8  |
| 1.5 | Base de données – entité-relation .....    | 9  |
| 2   | Conception .....                           | 9  |
| 2.1 | Diagramme de classes partie clientes ..... | 9  |
| 2.2 | Diagramme de classes partie serveur .....  | 10 |
| 2.3 | Diagramme de séquence interactions .....   | 11 |
| 2.4 | Base de données – schéma relationnel.....  | 13 |
| 3   | Implémentation.....                        | 16 |

## 1 Introduction

Le site web de mon projet sera une TODO-List, le principe de base est d'avoir différents projets avec différentes tâches qui ont des informations tels qu'un nom, ou une description. Les tâches ont un état et ils permettent de savoir l'avancer d'une tâche, ainsi que l'avancée du projet dans sa globalité.

Lorsqu'un visiteur arrivera sur l'accueil du site, il sera redirigé vers la page de connexion. Il n'aura pas le choix de se connecter pour accéder au site web. Si le visiteur ne possède pas de compte, il peut cliquer sur un lien en dessous pour se diriger vers la page de création de compte.

Une fois connecter, le client pourra accéder à tous les projets dont il fait partie, il peut aussi crée un nouveau projet dont il sera le seul à y être assigner. Dans chaque projet se trouve différentes tâches, nous pouvons crée une tâche, qui sera assigner par default dans l'état « TODO ». Il est possible de modifier le nom et la description de la tâche.

Les taches pourront changer d'état (TODO, In progress, Done, Validate). Chaque personne du projet peut créer ou, changer l'état des tâches, sauf pour l'état valider qui pourra être affecté par un administrateur.

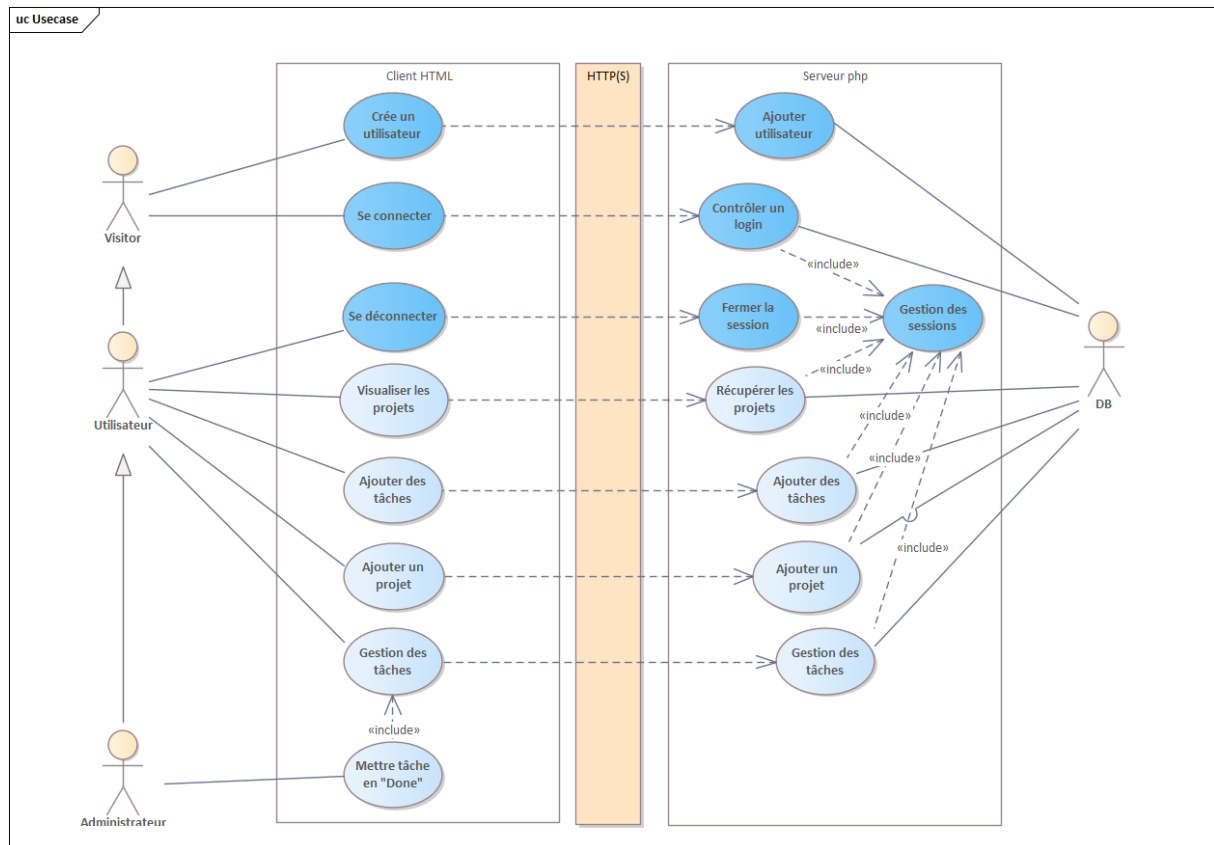
## 2 Analyse

### Usecase

Voici le usecase de mon projet, voici à présent les actions de chaque acteur.

| Acteur         | Activité   | Action  |
|----------------|--|---|
| Visiteur       | Un visiteur est une personne qui n'est pas connecter.                      | Se connecter à un compte utilisateur.   |
|                |  | Crée un compte utilisateur.   |
| Utilisateur    | Un visiteur qui se connecte à un compte deviens un utilisateur.            | Se déconnecter de son compte, et donc revenir à la page de connexion.   |
|                |  | Visualiser les différents projets dont il fait partie.  |
|                |  | Ajouter une tâche au projet.  |
|                |  | Gérer l'état des tâches mais aussi leur description ou leur nom.  |
|                |  | Crée un nouveau projet  |
| Administrateur | Un administrateur est un visiteur qui c'est connecter sur un compte admin. | Il possède les mêmes droits que l'utilisateur, sauf un droit en plus, qui est le pouvoir de mettre une tâche dans l'état valider. |

Et voici le schéma Usecase.

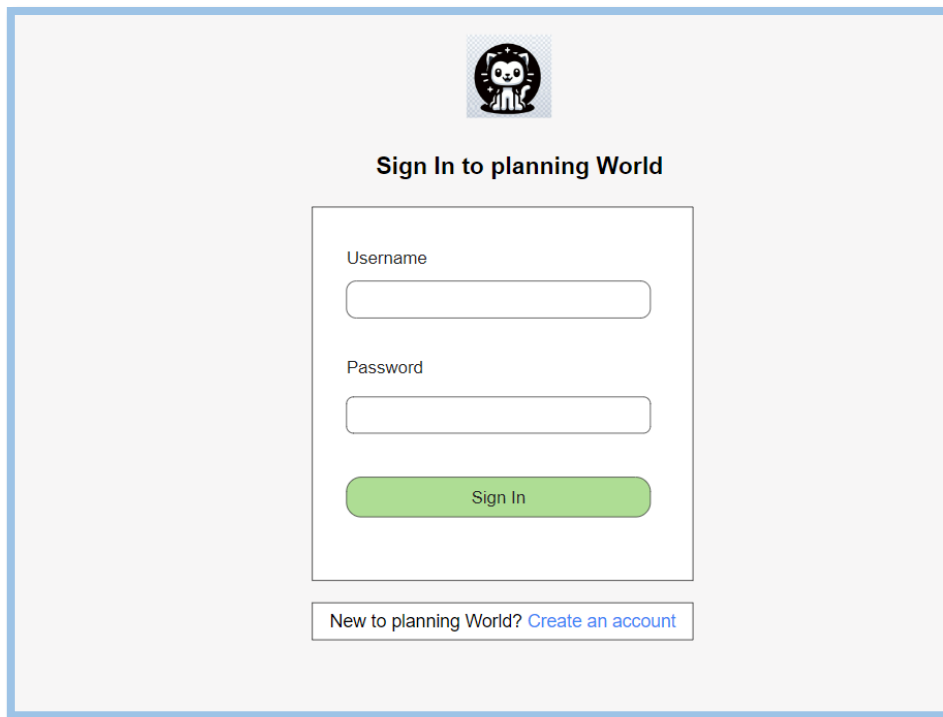


## Les maquettes

### 1. Login

Cette page est une page de connexion, le visiteur doit spécifier son nom d'utilisateur et son mot de passe pour se connecter. Si le visiteur ne possède pas de compte utilisateur, il a la possibilité d'en créer un en cliquant sur « create an account ».

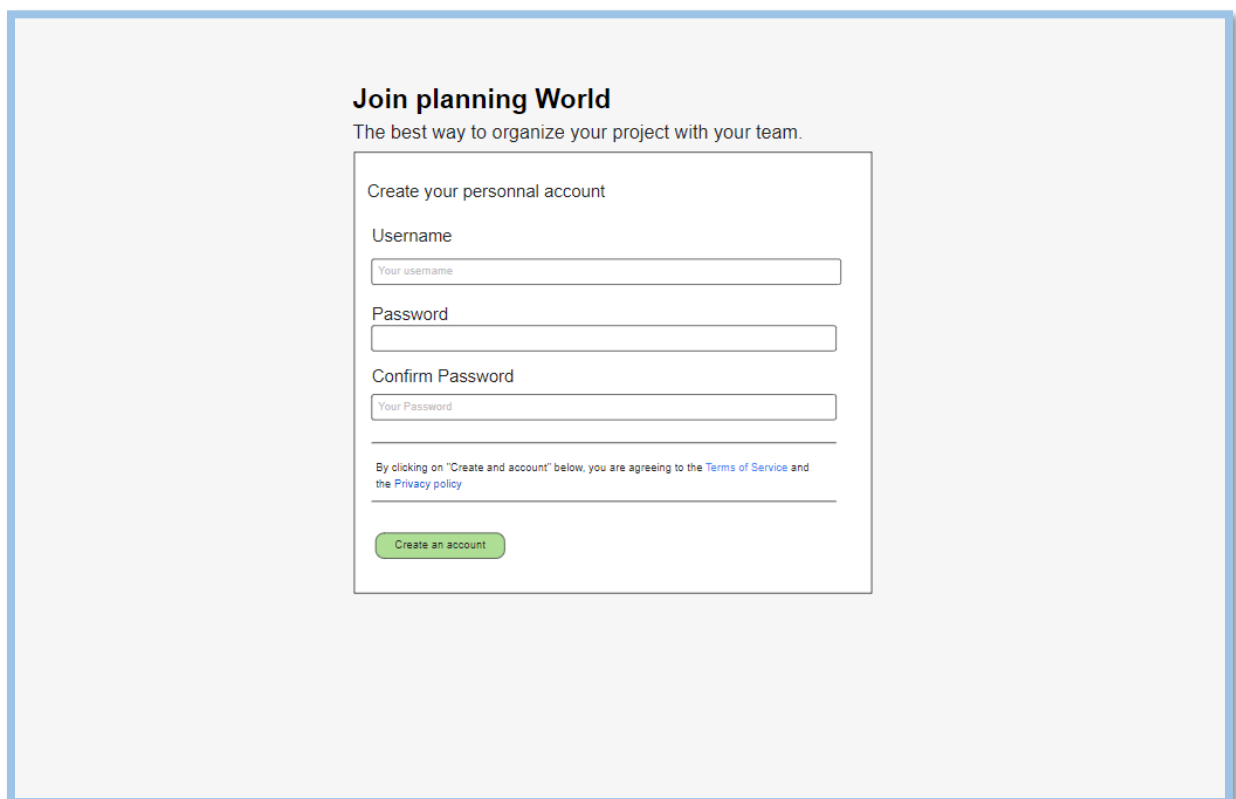
Pour se connecter à un utilisateur admin, il faut se connecter à un utilisateur possédant les permissions administrateur.



The image shows a sign-in form for 'planning World'. At the top center is a circular logo featuring a stylized cat with a star on its head. Below the logo, the text 'Sign In to planning World' is centered. The form itself is a white box with a thin border, containing two input fields: 'Username' and 'Password'. Below these fields is a green button with the text 'Sign In'. At the bottom of the form box, there is a link that says 'New to planning World? [Create an account](#)'.

## 2. Create account

Cette page permet à un visiteur de créer un compte utilisateur, il doit simplement spécifier le nom de l'utilisateur qu'il veut créer, ainsi que son mot de passe. Une fois les informations indiquées, il suffit de cliquer sur le bouton « Create an account ». Il est impossible de créer un utilisateur qui existe déjà.

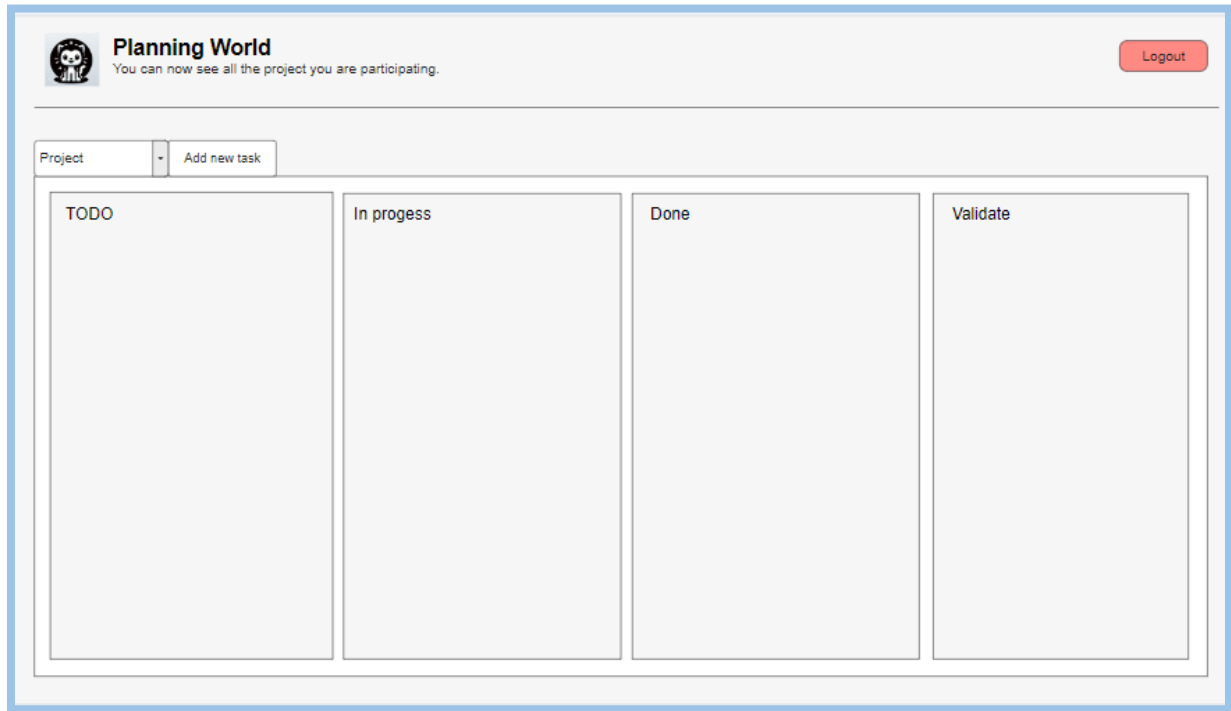


The image shows a 'Join planning World' form. At the top, the text 'Join planning World' is followed by the tagline 'The best way to organize your project with your team.' Below this is a white form box with a thin border. Inside the box, the text 'Create your personal account' is at the top. There are three input fields: 'Username' (with placeholder text 'Your username'), 'Password', and 'Confirm Password' (with placeholder text 'Your Password'). Below the input fields, there is a line of text: 'By clicking on "Create an account" below, you are agreeing to the [Terms of Service](#) and the [Privacy policy](#)'. At the bottom of the form box is a green button with the text 'Create an account'.

### 3. Gestion de projet

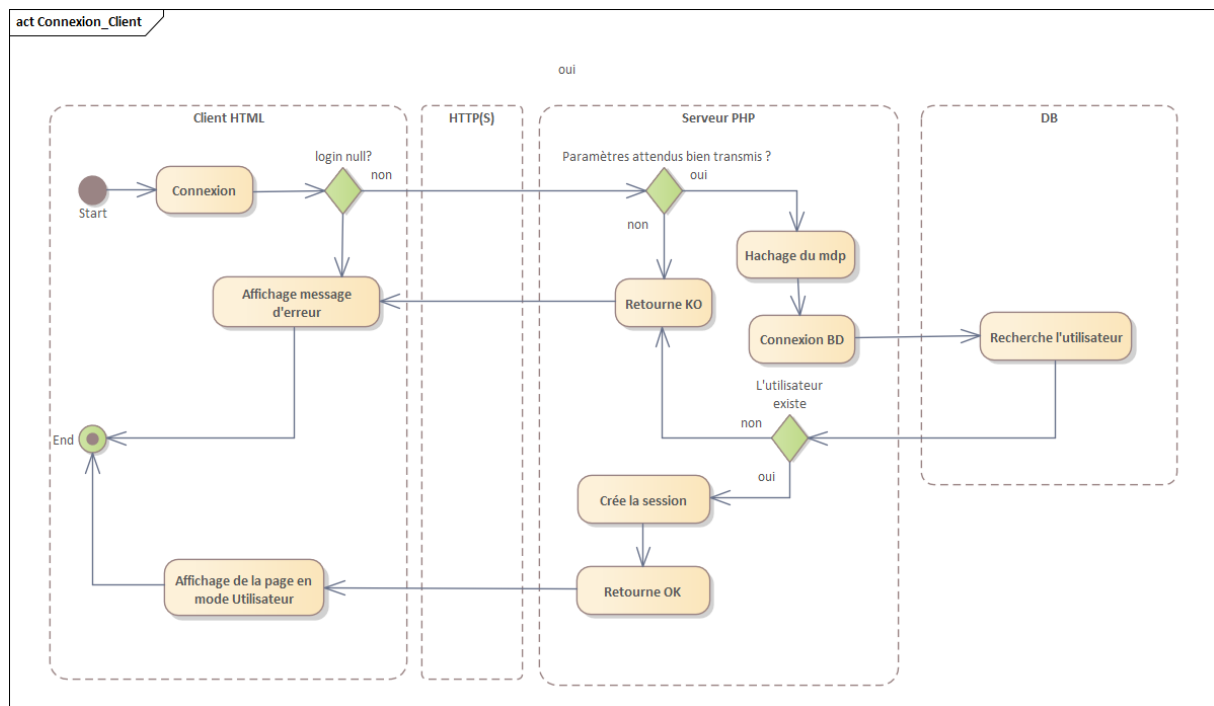
C'est ici que les utilisateurs peuvent accéder au gestionnaire de projets. En cliquant sur le menu déroulant, l'utilisateur peut voir tous les projets sur lesquelles ils travaillent. Il peut changer l'état d'une tâche en « drag and drop » vers un autre état. L'utilisateur peut aussi ajouter de nouvelles tâches en cliquant sur « add new task ». La nouvelle tâche apparaîtra dans l'état « TODO ».

Le nom ou la description des tâches peut aussi être modifié.

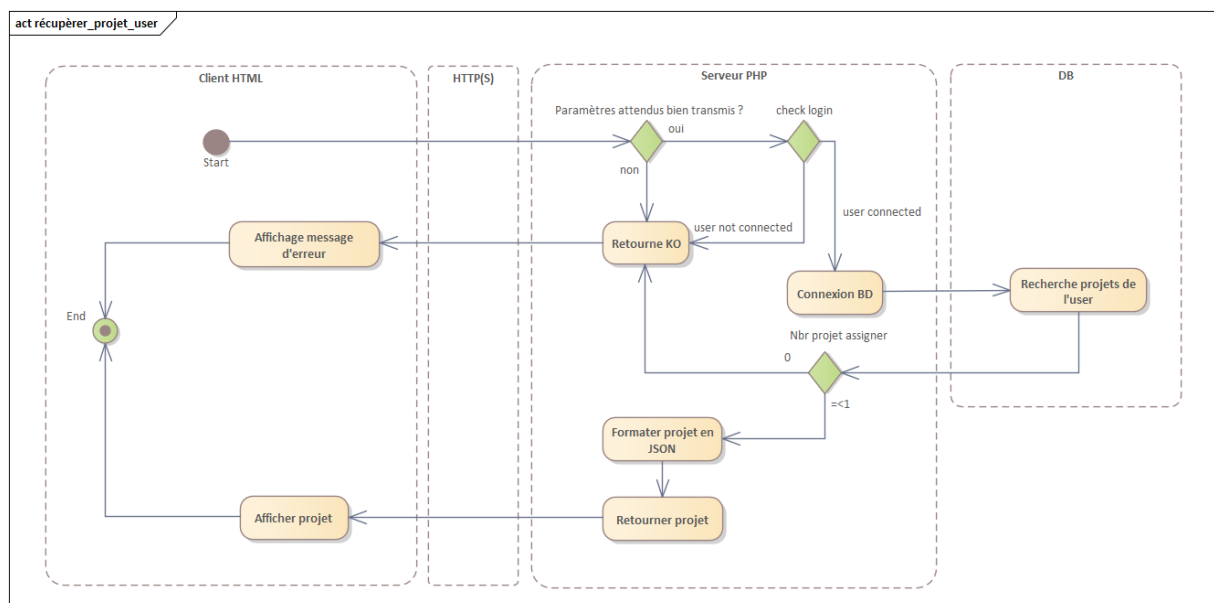


### Diagramme d'activités

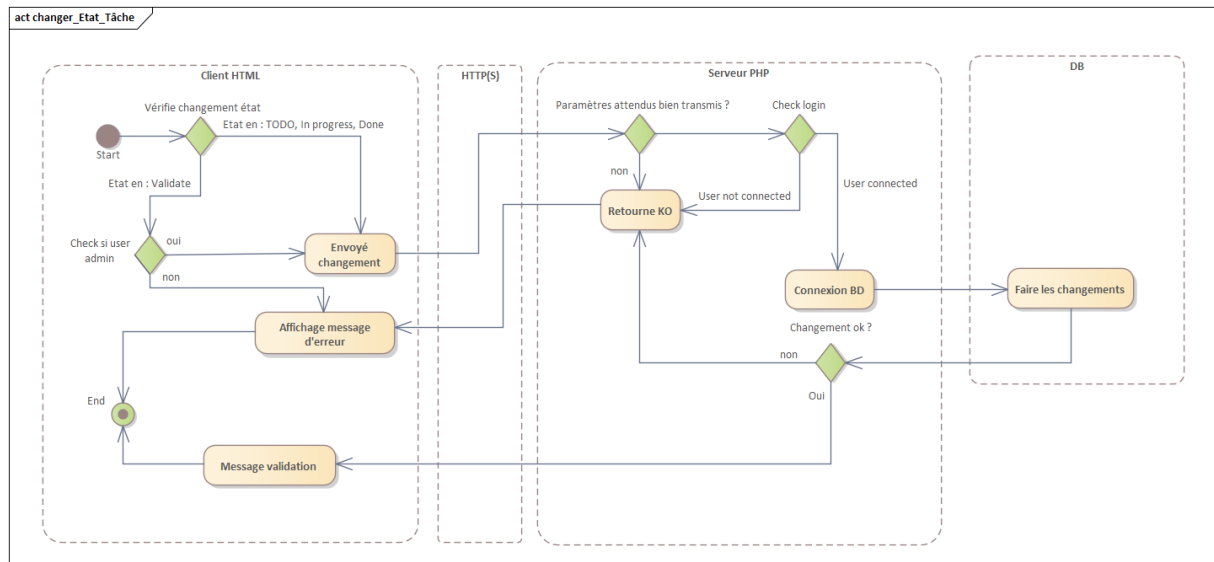
Voici le diagramme d'activité pour se connecter à un compte pour notre application. Nous vérifions d'abord si le login n'est pas vide (Null), puis nous envoyons au serveur PHP le nom d'utilisateur et le mot de passe. Ensuite nous hachons le mot de passe, et envoyons à la DB les informations pour vérifier si nous pouvons nous connecter. Si nous pouvons nous connecter à l'utilisateur, nous créons une session et affichons la page client. Sinon nous envoyons un message d'erreur au visiteur.



Pour récupérer les projets de l'utilisateur, nous allons d'abord envoyer au serveur PHP le nom de l'utilisateur connecter. Ensuite nous allons nous connecter à la DB, puis rechercher tous les projets sur lesquels l'utilisateur participe. La liste va être formater en JSON, envoyé au client et afficher sur l'interface web.

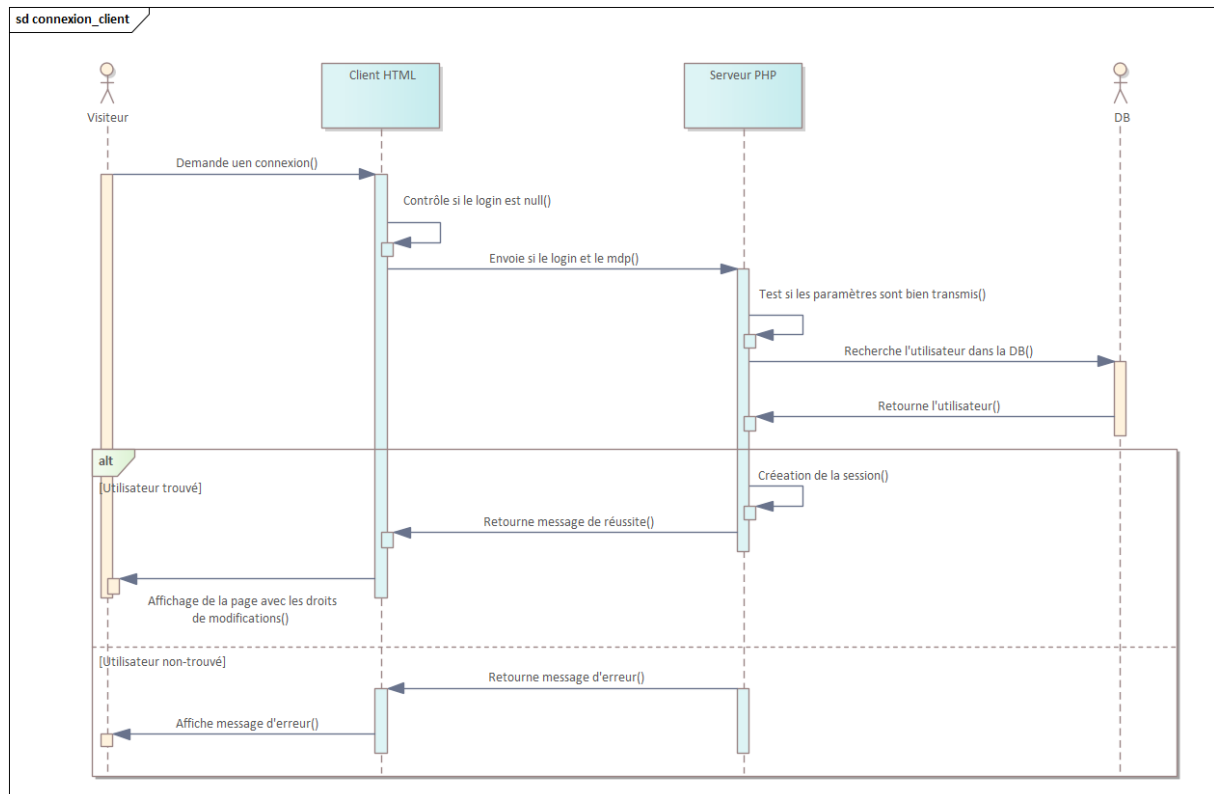


Un autre diagramme pour changer l'état d'une tâche, nous devons d'abord regarder dans qu'elle état nous allons déplacer notre tâche, si la tâche est déplacée dans l'état « DONE », il faut vérifier si notre utilisateur est admin. Ensuite nous allons envoyer au serveur dans qu'elle projet nous faisons les modifications, ainsi que dans qu'elle tâche nous déplaçons. Il faut aussi envoyer dans qu'elle était-elle se trouvait, et dans qu'elle état la tâche va se retrouver.



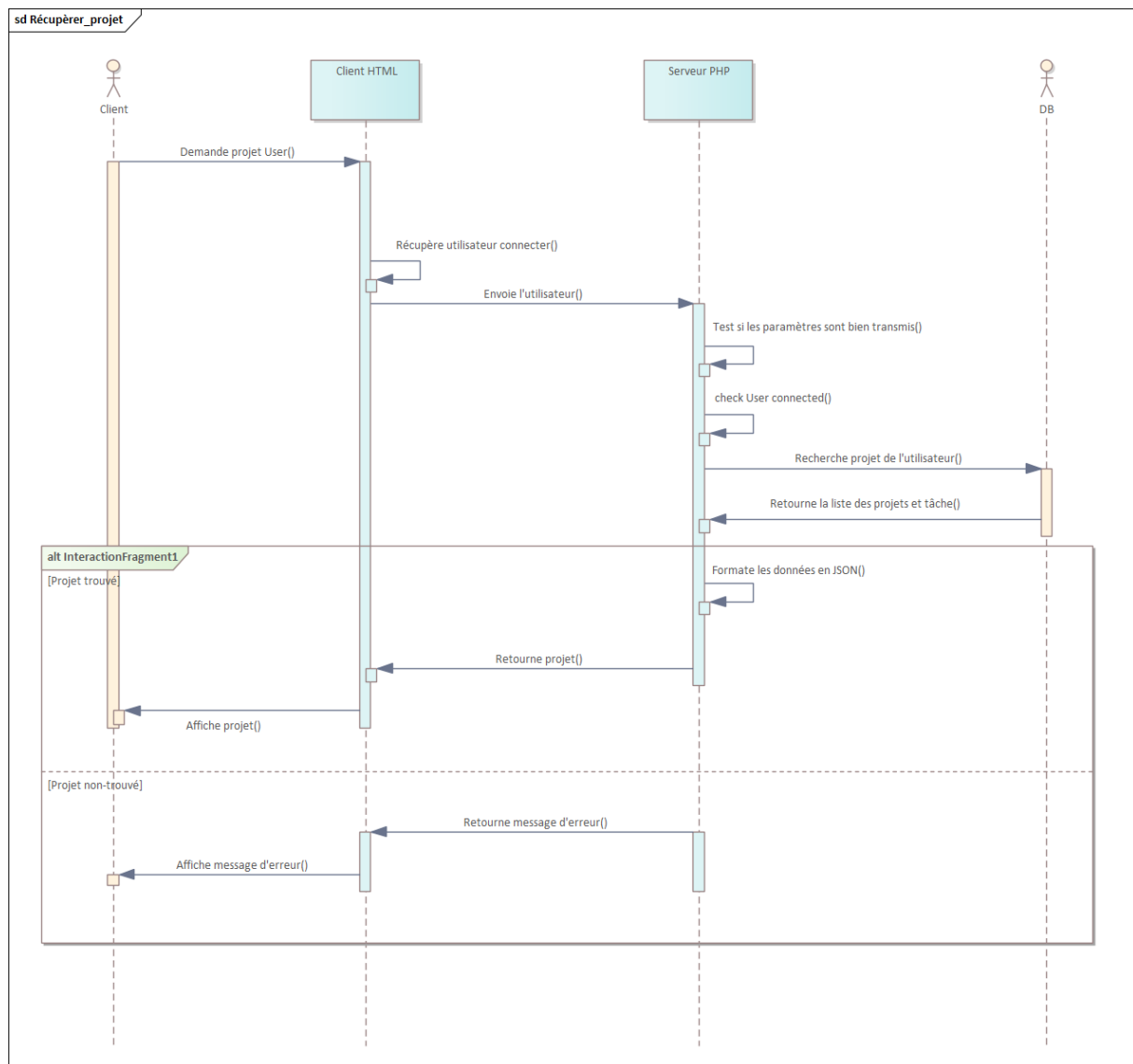
## Diagramme de séquence système

Voici le diagramme séquence système pour la connexion à un utilisateur.



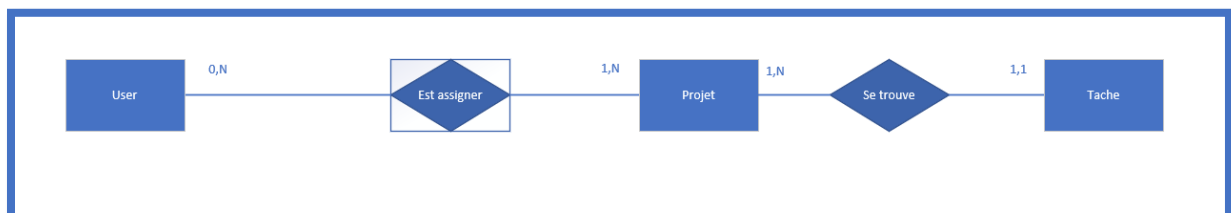
Voici le diagramme pour récupérer les projets d'un utilisateur.





## Base de données – entité-relation

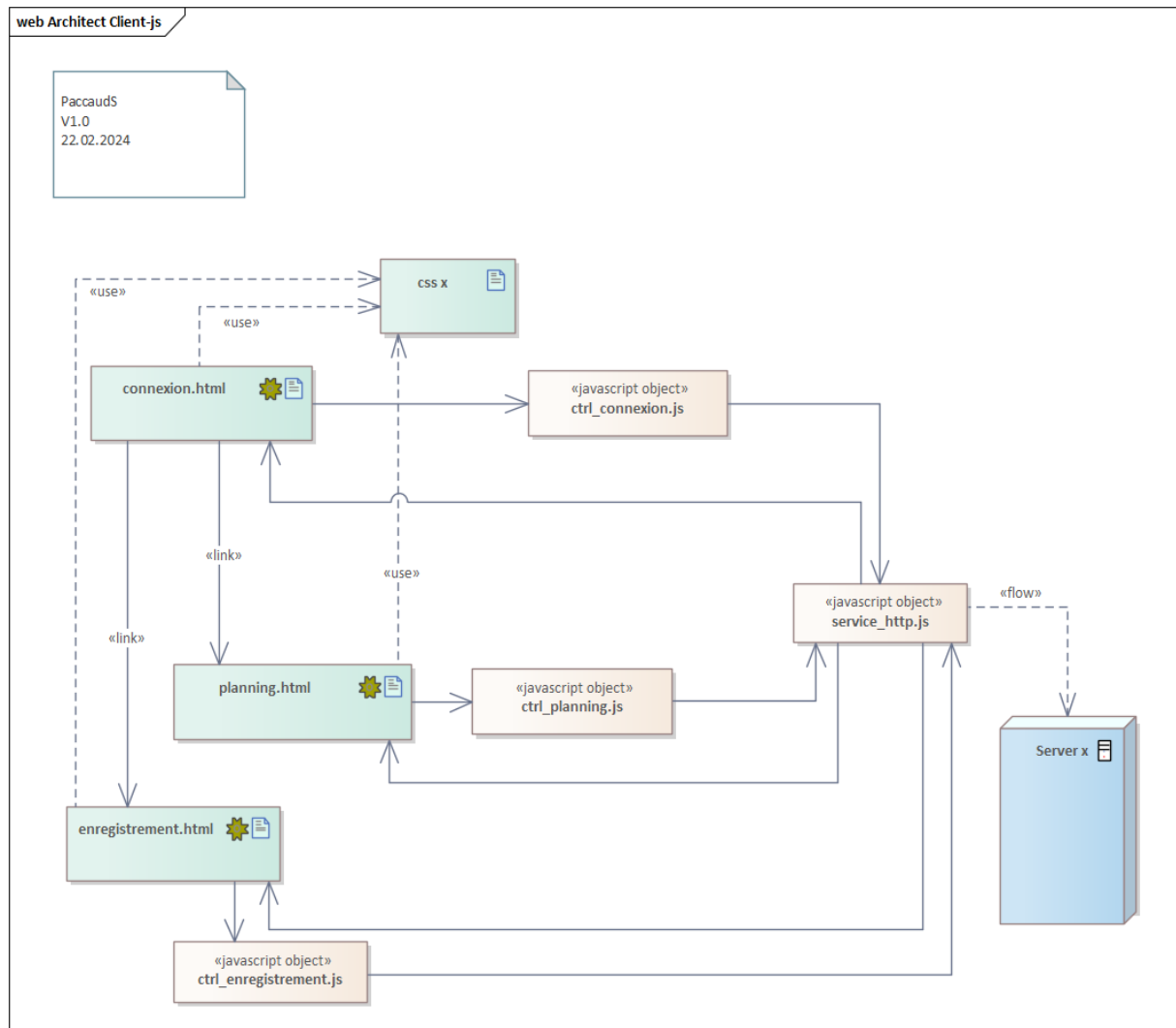
Voici le schéma entité relation de ma base de données. Une table de relation entre la table utilisateurs et la table projet a été ajoutée, car plusieurs utilisateurs peuvent travailler sur un projet. Cependant c'est fonctionnalité n'a pas encore mis en place, mais la table de relation a déjà été mis en place pour ajouter cette fonctionnalité dans un futur proche.



## 3 Conception

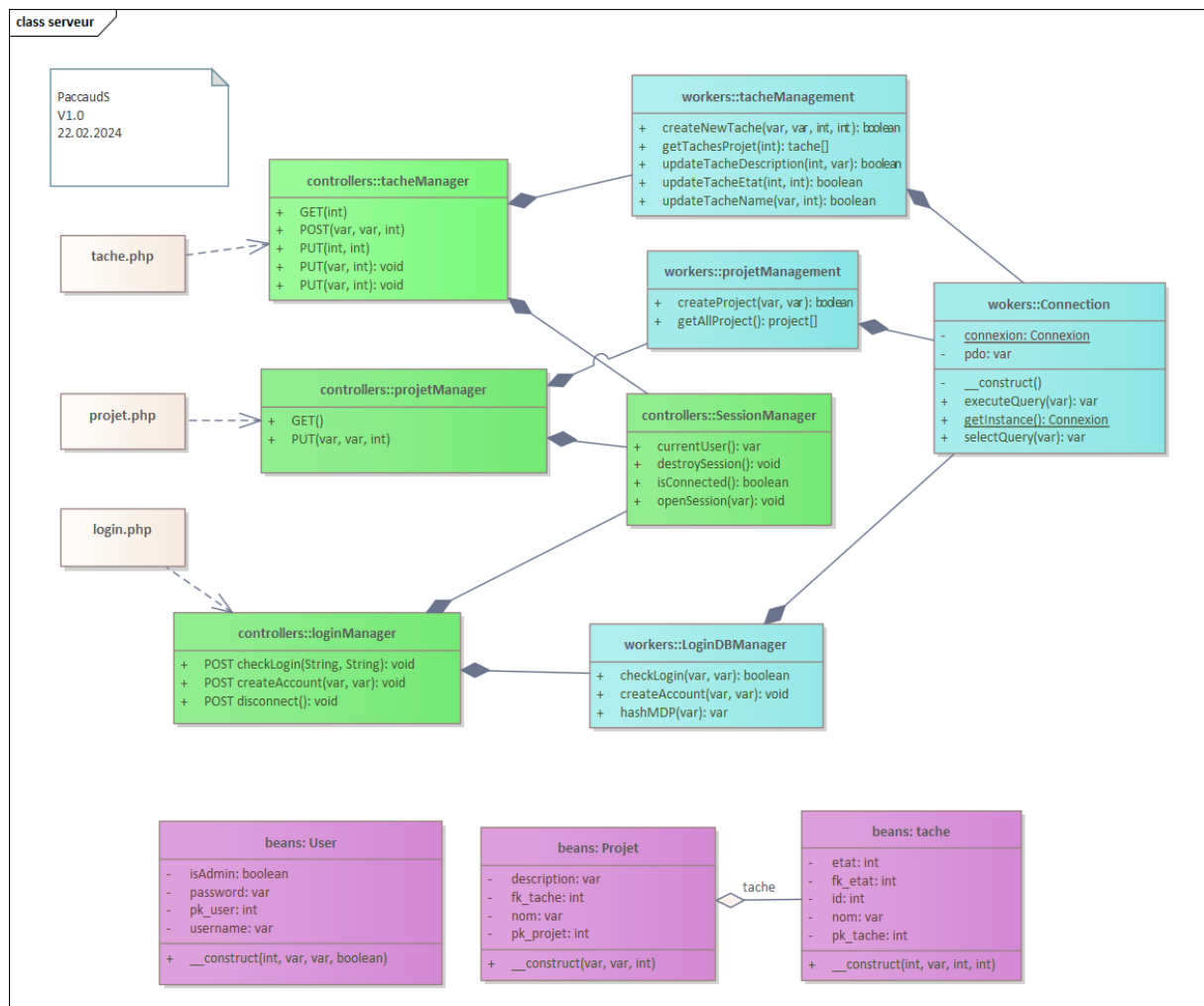
### Diagramme de classes partie clientes

Voici le diagramme de classe pour la partie cliente, nous pouvons voir qu'il n'y a pas d'inex.HTML, car lorsque nous arriverons sur le siteWeb, nous serons dirigés directement sur connexion.html



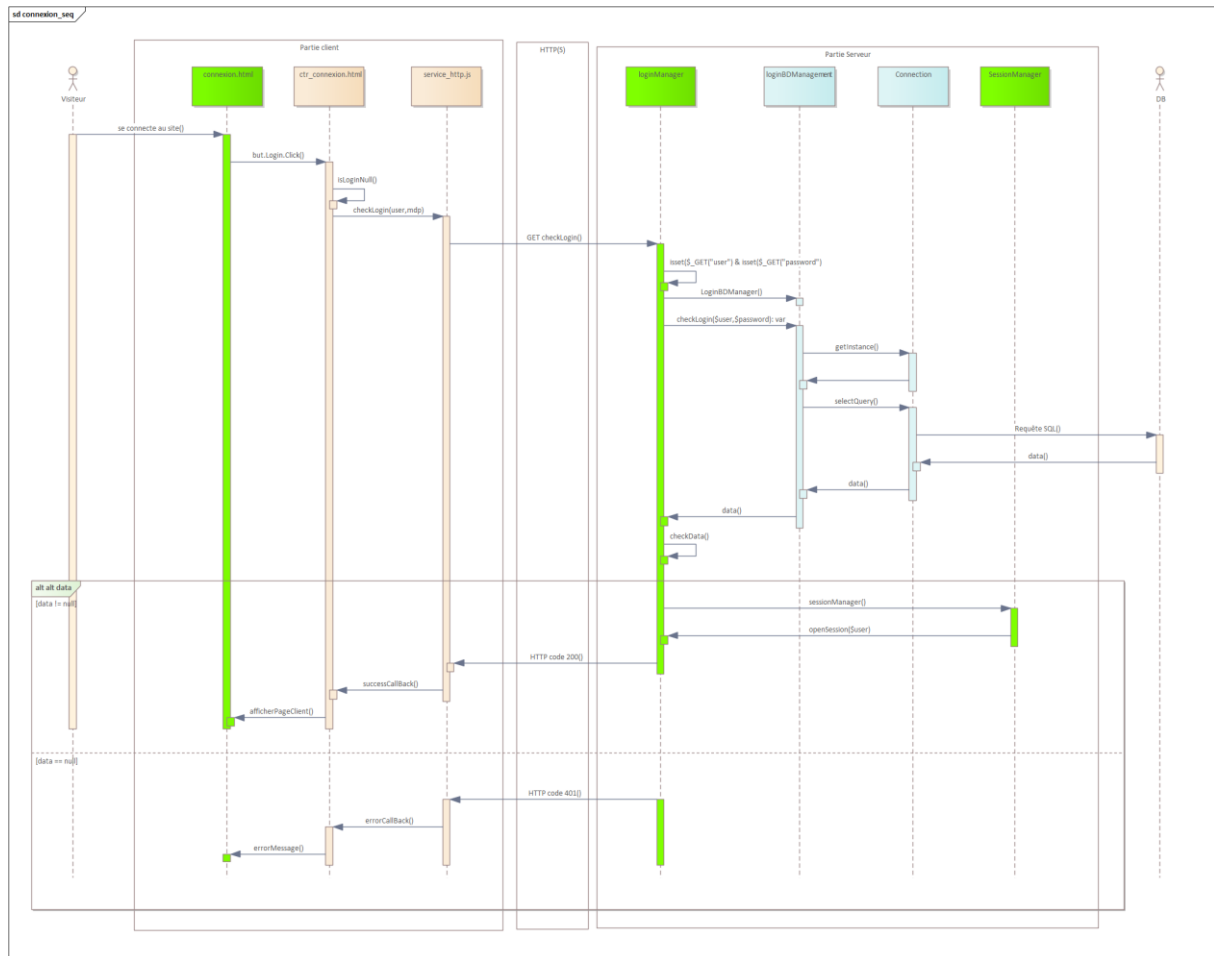
## Diagramme de classes partie serveur

Et voici le diagramme du côté serveur.

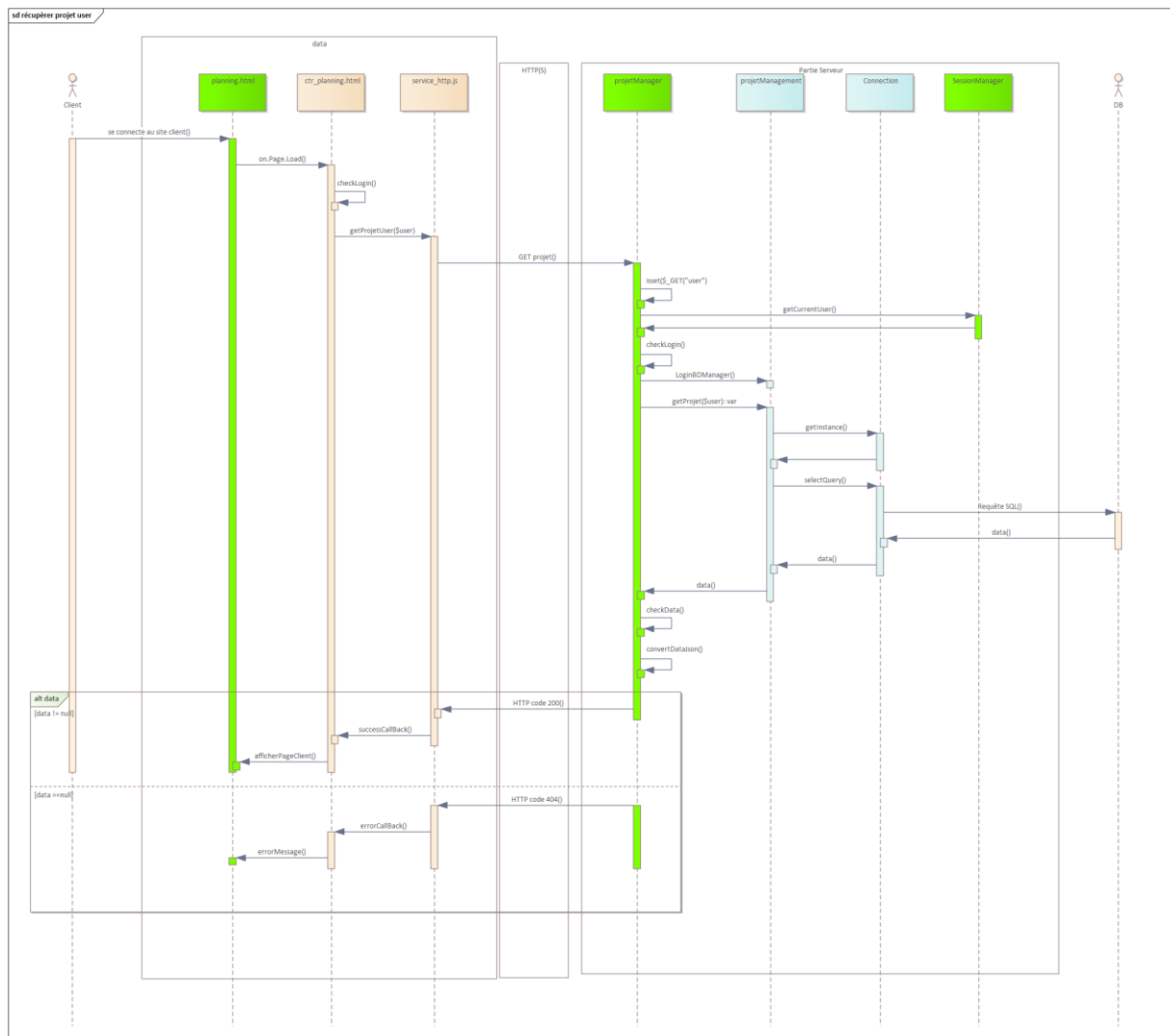


## Diagramme de séquence interactions

Voici le diagramme de séquence d'interaction pour la connexion d'un visiteur.

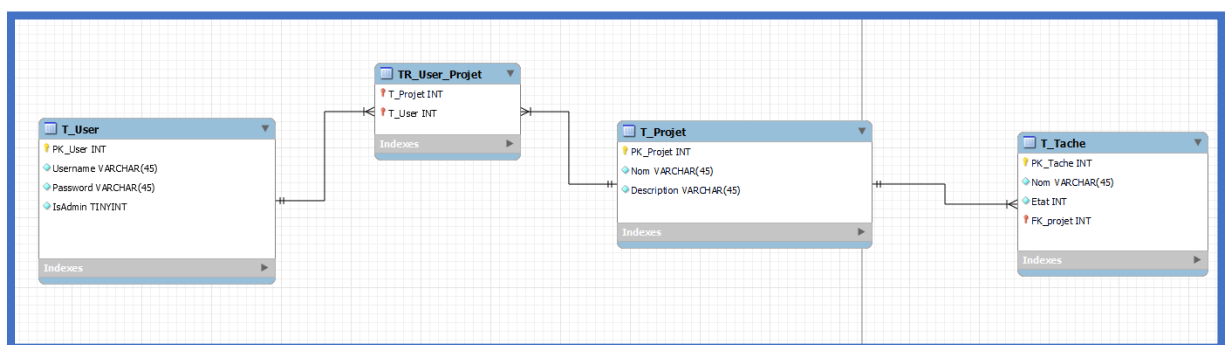


Et voici le diagramme pour la partie où il faut récupérer les projets d'un utilisateur.



## Base de données – schéma relationnel

Voici le schéma relationnel de la DB.



## Conception des tests

Pour ce qui est des tests, voici les pas à effectuer pour vérifier la sécurité ainsi que le fonctionnement du site.

Voici les tests pour vérifier le fonctionnement de mon application :

| Fonctionnement   | Page ?        | Résultat attendu   | Résultat obtenu | Test validé ? |
|--|---------------|--|-----------------|---------------|
| Crée un nouvel utilisateur   | Register.html | Un utilisateur devrait se créer et être stocker sur la DB,                                       |                 |               |
| Crée un utilisateur qui est déjà existant  | Register.html | Un message d'erreur devrait apparaitre.  |                 |               |
| Se connecter à un compte existant avec bon nom d'utilisateur et mot de passe.                | Login.html    | La connexion devrait réussir, et la page « planning.html » devrait s'ouvrir.                     |                 |               |
| Se connecter à un compte existant avec mauvais mot de passe ou nom d'utilisateur inexistant. | Login.html    | Un message d'erreur devrait apparaitre.  |                 |               |
| Recevoir tous les projets d'un utilisateur.  | Planning.html | Tous les projets qui sont assigner à l'utilisateur doivent être afficher dans le menu déroulant. |                 |               |
| Création d'une nouvelle tâche.   | Planning.html | Une tâche doit se crée dans l'état « todo ». Elle doit aussi être stocker dans la DB.            |                 |               |
| Modifié le nom / la description d'une tâche.   | Planning.html | La tâche va changer de nom / description, le changement dois aussi changer dans la DB.           |                 |               |
| Déplacer une tâche dans les états « TODO, In Progress, Done ».                               | Planning.html | La tache doit changer d'état, le changement dois aussi changer dans la DB.                       |                 |               |
| Déplacer une tâche dans l'état « Validate » en tant qu'user non-administrateur.              | Planning.html | Un message d'erreur devrait apparaitre, et la tâche ne dois pas changer d'état.                  |                 |               |
| Déplacer une tâche dans l'état « Validate » en tant qu'user administrateur.                  | Planning.html | La tache doit changer d'état, le changement dois aussi changer dans la DB.                       |                 |               |
| Déconnexion d'un utilisateur.  | Planning.html | L'utilisateur doit se faire déconnecter.   |                 |               |

Et voici les testes pour tester la sécurité de mon site web.

| Fonctionnement   | Page ?  | Résultat attendu   | Résultat obtenu | Test validé ? |
|--|---|--|-----------------|---------------|
| Vérifier si des injections SQL / HTML est possible sur les champs de création de compte. | Register.ht ml                                    | Les injections de devraient pas fonctionner.   |                 |               |
| Vérifier si des injections SQL / HTML est possible sur les champs de connexion.          | Planning.ht ml                                    | Les injections de devraient pas fonctionner.   |                 |               |
| Vérifier qu'avec un outil comme « POSTMAN », les requêtes SQL ne fonctionne pas.         | Register.ht ml / Connexion. html / Planning.ht ml | Le site devrait remarquer qu'aucun user n'est connecter, et les requêtes ne devrait pas être pris en compte. |                 |               |

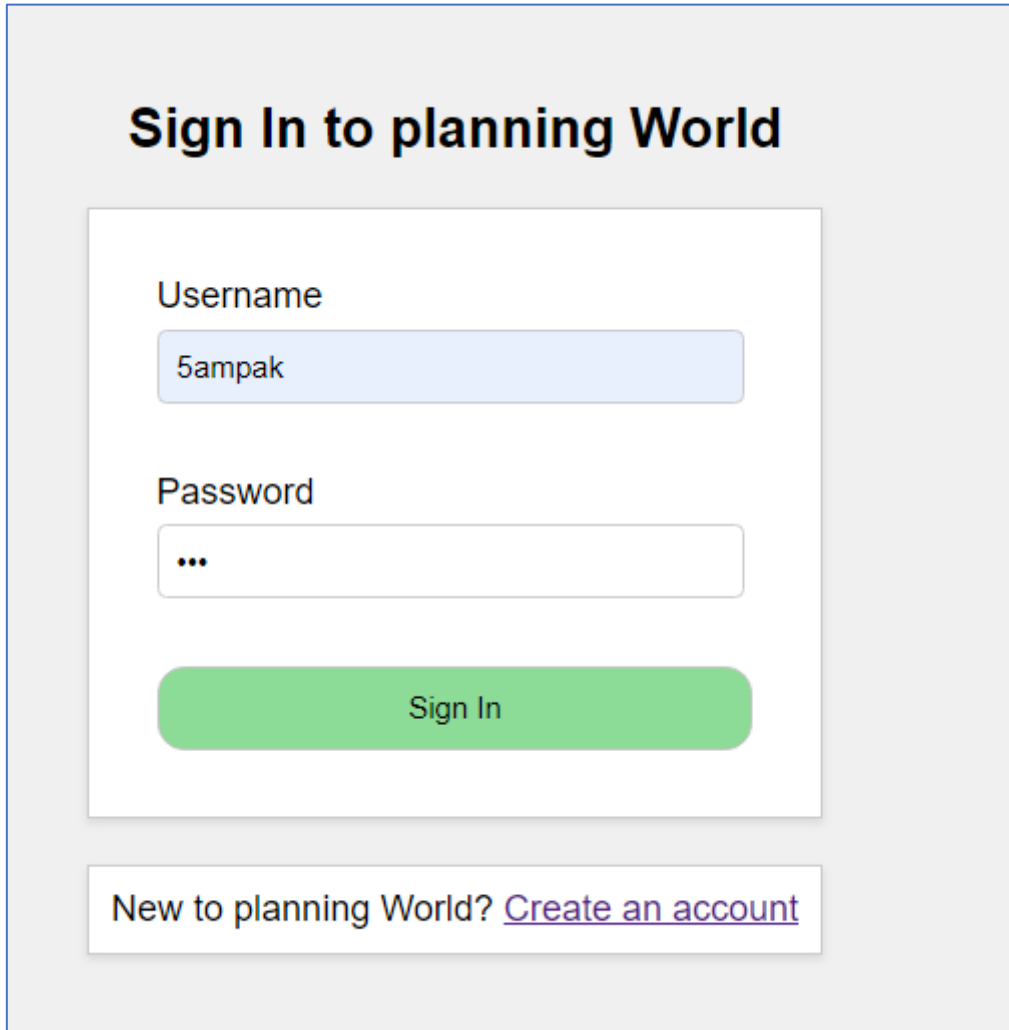
## 4 Implémentation

### Descente de code

Voici à présent la descente du code pour les trois fonctionnalités suivantes.

#### 4. Login

Pour commencer, lorsque nous avons remplie les champs du « username » et du « password », nous allons cliquer sur le bouton « Sign In », qui contient un écouteur dans le javascript.



**Sign In to planning World**

Username  
5ampak

Password  
...

Sign In

New to planning World? [Create an account](#)

Dans l'écouteur du bouton (on click), nous commençons par récupérer les champs des deux valeur, puis nous appelons une méthode login user, avec les paramètres de deux champs.

```
$(document).ready(function () {  
    $("#submitCon").click(function (event) {  
        event.preventDefault();  
  
        // get data  
        console.log("Button login pressed");  
  
        loginValueUsername = $("#username").val();
```



```

        console.log("Username:", loginValueUsername);

        loginValuePassword = $("#password").val();
        console.log("Password:", loginValuePassword);

        //méthode login
        loginUser(loginValueUsername, loginValuePassword);

    });
});

```

Cette méthode appelle une méthode ajax, avec le nom d'utilisateur et le mot de passe. Cette méthode est dans le service http, et va permettre de faire une requête au serveur.

```

function loginUser(username, password) {
    console.log("Connexion à l'utilisateur : " + username + " " + password);

    // Appel de la fonction AJAX pour se connecter
    connexionUserAjax(username, password, function (response) {
        // Callback de succès
    });
}

```

Passons directement dans la méthode ajax. Cette méthode va envoyer au serveur le nom d'utilisateur et le mot de passe, les informations vont en POST et donc passer dans le body et non dans l'url pour une question de sécurité. Le client va questionner le « connexion.php » avec l'action « loginUser » et va attendre un retour en JSON.

```

function connexionUserAjax(username, passwd, successCallback, errorCallback) {
    console.log("-----")
    console.log("Login méthode ajax");
    $.ajax({
        type: "POST",
        dataType: "JSON",
        url: BASE_URL + "connexion.php",
        data: {
            "action": "loginUser",
            "username": username,
            "password": passwd
        },
        xhrFields: {
            withCredentials: true
        },
        success: successCallback,
        error: errorCallback
    });
}

```

Dans le connexion.php, nous avons des tests pour vérifier si nous avons toutes les informations nécessaires, et vu que nous avons une action « loginUser », le serveur récupère les informations du mot de passe et du nom d'utilisateur. Le serveur va ensuite appeler le login Ctrl.

```

if (isset($_POST['action'])) {
    switch ($_POST['action']) {

```

```

    case 'loginUser':
        if (isset($_POST['username'])) {
            if (isset($_POST['password'])) {

                $username = $_POST["username"];
                $password = $_POST["password"];

                //ctrl login
                $login = new LoginManager();
                $login->loginUser($username, $password);
            }
        }
    }
}

```

Login user va demander au wrk si l'utilisateur avec les paramètres peut se connecter, puis il va décoder les informations de retour et vérifier s'il y a une data « connexionPossible », si oui, une session va être créée pour le client. Puis nous faisons un echo pour envoyer toutes les informations au client.

```

public function loginUser($username, $password)
{

    $jsonString = $this->manager->loginUser($username, $password);
    //don't crée session when no logged
    $data = json_decode($jsonString, true);

    if ($data['connexionPossible']) {
        // Crée instance + définit la variable de session (USER UNIQUE !)
        $this->session->set('username', $username);
    }
    echo ($jsonString);
}

```

Pour parler de la méthode du worker, nous avons déjà une requête SQL pour vérifier si l'utilisateur existe, s'il existe nous créons un bean user avec les mêmes paramètres que celui qui existe dans la DB, ensuite nous récupérons son mot de passe haché, et nous utilisons « password\_verify », c'est une fonctionnalité PHP, si un mot de passe haché, et un mot de passe non haché sont les mêmes. Si c'est le cas nous envoyons un « http\_response(200) », ce qui signifie que la connexion est possible, nous retournons aussi un JSON avec quelque info de l'utilisateur, et le « data=true » pour dire que la connexion est possible. Sinon nous retournons des réponses d'erreur.

```

public function loginUser($username, $password)
{

    $json = "";

    // Paramètres pour la requête
    $params = array('username' => $username);
    //requête pour check user existe

```

```

    $row = $this->connexion->selectSingleQuery("SELECT * FROM t_user WHERE
username=:username", $params);

    if ($row) {
        // Création d'un nouvel objet User et initialisation à partir des
données de la base de données
        $user = new User();
        $user->initFromDb($row);

        if ($user->getPassword() !== null) {
            // Récupération du mot de passe haché depuis la base de données
            $db_password_hash = $user->getPassword();

            //vérifier si c'est le même password
            if (password_verify($password, $db_password_hash)) {

                http_response_code(200);
                $json = json_encode(array('connexionPossible'=> true,'message' =>
'Connexion possible ! ', 'Userlogin' => $user->getUsername(), 'UserPK' =>
$user->getPKUser(), "IsAdmin" => $user->isAdmin()));

            } else {
                http_response_code(401);
                $json = json_encode(array('connexionPossible'=> false,'message' =>
'Connexion impossible : mot de passe incorrect'));
            }

        } else {
            http_response_code(404);
            $json = json_encode(array('connexionPossible'=> false,'message' =>
'Connexion impossible : mot de passe non défini dans la base de données'));
        }
    } else {
        http_response_code(404);
        $json = json_encode(array('connexionPossible'=> false,'message' =>
'Connexion impossible : utilisateur non trouvé'));
    }

    return $json;
}

```

Retournons à présent du côté client dans le success callback (http\_response(200)), le client commence par récupérer les informations du JSON, puis le client les stocks dans la session storage. Ensuite nous changer de passe utilisateur pour arriver au site principal. Si la connexion échoue, un message d'erreur apparaît.

```

// Récupération des propriétés de la réponse JSON
var userLogin = response["Userlogin"];
var userPK = response["UserPK"];

```

```

    var userAdmin = response["IsAdmin"];

    alert("Connexion réussis avec succès : " + userLogin);

    sessionStorage.setItem('username', userLogin);
    sessionStorage.setItem('PKuser', userPK);
    sessionStorage.setItem('IsAdmin', userAdmin);
    // Redirigez l'utilisateur vers la page planning.html ou effectuez
d'autres actions nécessaires
    window.location.href = "planning.html";

}, function (error) {
    alert("Login ou mot de passe incorrect");
});
}

```

### 5. Get projet User

La récupération des projets se fait au lancement de la page, dans le « getScript » nous commençons par récupérer les informations utilisateurs dans les « session Storage », ensuite nous vérifions si les variables existe sinon cela c'est signifié que l'utilisateur n'est pas connecté. Nous appelons la méthode « getUserProjet », c'est une méthode dans le « http service ». C'est une méthode Ajax demander au serveur les projets de l'utilisateur.

```

$.getScript("javascript/service/service_http.js", function () {
    console.log("servicesHttp.js chargé !");

    // récupérer les projet
    const pkUserConnected = sessionStorage.getItem('PKuser');
    const UserConnected = sessionStorage.getItem('username');

    console.log(pkUserConnected);
    console.log(UserConnected);

    if (pkUserConnected === null || UserConnected === null) {
        alert("Un problème est survenu, merci de vous reconnecter");
        window.location.href = "connexion.html";
    } else {
        //get liste projet
        getUserProjet(UserConnected, pkUserConnected, function (response) {

```

Cette méthode va envoyer au serveur les informations de utilisateur connecté dans les paramètres, il recevra un JSON.

```

function getUserProjet(userConnected, pkUserConnected, successCallback,
errorCallback) {
    console.log("-----")
    console.log("get User Projet méthode ajax");
    $.ajax({
        type: "GET",
        dataType: "JSON",

```

```

        url: BASE_URL +
`projet.php?action=getProjetUser&username=${userConnected}&pkuser=${pkUserConn
ected}` ,
        xhrFields: {
            withCredentials: true
        },
        success: successCallback,
        error: errorCallback
    });
}

```

Cette méthode dans le serveur va vérifier si tous les paramètres sont bien existant, s oui, il va récupérer les informations envoyées depuis l'URL, et directement les envoyés au ctrl.

```

if (isset($_GET['action'])) {
    switch ($_GET['action']) {
        case 'getProjetUser':
            if (isset($_GET['username'])) {
                if (isset($_GET['pkuser'])) {

                    $username = $_GET["username"];
                    $pkUser = $_GET["pkuser"];

                    //ctrl login 2
                    $login = new ProjetManager();
                    $login->getProjetUser($username, $pkUser);

                }
            }
        }
    }
}

```

Le serveur va d'abord vérifier si l'utilisateur est bien connecté en vérifier s'il fait partie d'une session « 'username' », ensuite nous appelons une méthode dans le worker avec les informations de l'utilisateur pour une requête. Puis le résultat va être echo.

```

public function getProjetUser($username, $pkUser)
{
    //get 'username'
    if ($this->session->get('username')) {
        $result = $this->manager->getProjetUser($pkUser);
        echo ($result);
    }
}

```

La méthode du worker commence par stocker les paramètres avec le « htmlspecialchars », ceci est pour éviter les injections SQL. Puis nous préparons une requête pour récupérer la PK de chaque projet où l'utilisateur en fait partie. Puis nous faisons une deuxième requête pour avoir toutes les informations du projet tels que le nom, description, état. Le JSON cette fois sera en tableau car nous pouvons avoir plusieurs informations.

```

public function getProjetUser($pkUser)
{
    $escapedPKUser = htmlspecialchars($pkUser, ENT_QUOTES, 'UTF-8');
}

```

```

$result = "";
//get pk projet de l'user
$params = array('pkUser' => $escapedPKUser);
$rows = $this->connexion->selectQuery("SELECT FK_Projet FROM
tr_user_projet WHERE FK_User = :pkUser", $params);

if ($rows) {
    // Tableau pour stocker les objets Projet
    $projets = array();

    foreach ($rows as $row) {
        // get clé primaire du projet
        $pkProjet = $row['FK_Projet'];

        // Get projet detail
        $params = array('pkProjet' => $pkProjet);
        $rowProjetData = $this->connexion->selectSingleQuery("SELECT * FROM
t_projet WHERE PK_Projet = :pkProjet", $params);

        if ($rowProjetData) {
            $projet = new Projet();
            $projet->initFromDb($rowProjetData);
            $projets[] = $projet;
        }
    }

    // prepare projet json
    $projetList = array();
    foreach ($projets as $projet) {
        $projetData = array(
            'Nom' => $projet->getNom(),
            'Description' => $projet->getDesccription(),
            'PK_Projet' => $projet->getPKProjet(),
        );
        $projetList[] = $projetData;
    }

    //convert json
    $jsonProjetList = json_encode($projetList);
    http_response_code(200);
    $result = $jsonProjetList;

} else {
    http_response_code(404);
    $result = json_encode(array('message' => 'Aucune ligne trouvée.'));
}
return $result;
}

```

Revenons du côté client, dans le success callback, nous récupérons chaque projet du JSON, et affichons le nom du projet dans le menu déroulant. Nous stockons aussi la liste des projets dans la session storage, car nous en aurons besoins plus tard.

```
// Callback de succès
console.log(response);

if (response.length === 0) {
    alert("Aucun projet trouvé.");
} else {

    alert("Projets récupérés avec succès!");

    // Supprime tous les éléments de la liste déroulante
    var dropdown = document.getElementById("projects-dropdown");
    dropdown.innerHTML = "";

    // Récupère les projets du JSON
    var projects = response;

    // Parcourt chaque projet et ajoute-le à la liste déroulante
    projects.forEach(function (project) {
        var option = document.createElement("option");
        option.value = project.PK_Projet;
        option.text = project.Nom;
        dropdown.appendChild(option);

        //stock session projet
        sessionStorage.setItem('Projet_' + project.PK_Projet,
project.Nom);

    });
}

}, function (error) {
    // Callback d'erreur
    alert("Aucun projet pour cette utilisateur !");
});
}
});
```

### Problème rencontrer

J'ai eu beaucoup de problème durant ce projet, mais voici les plus importants :

Un des problèmes qui m'a fait le plus galérer, c'est pour récupérer les informations du côté serveur des « PUT ». J'avais oublié tellement de paramètres comme le header.

```
header('Access-Control-Allow-Methods: PUT');
```

Mais aussi le fait que le teste ne soit pas un

```
if (isset($_PUT['action'])) {
```

Mais un « request\_method ».

```
} else if ($_SERVER['REQUEST_METHOD'] === 'PUT') {  
  
    parse_str(file_get_contents("php://input"), $vars);
```

J'ai dû perdre au moins une bonne heure à résoudre tous ces problèmes.

Ensuite ce n'est pas vraiment un problème mais plutôt une erreur, c'est pour la méthode ou je crée un projet pour un utilisateur spécifique, je devais garder la PK de l'utilisateur, sauf que j'ai décider d'envoyer le nom d'utilisateur et de faire une requête pour récupérer la clé primaire de l'utilisateur, et d'ensuite faire une requête dans la table de relation avec la clé de l'utilisateur, ce qui est stupide car j'avais en session storage la pk de l'user...

Un autre problème que j'ai eu, c'était pour vérifier du côté serveur si l'utilisateur est bien connecté. Ce que je faisais était ceci :


```
//get 'username'  
if ($_SESSION['username'] == $username) {  
    $result = $this->manager->getProjetUser($pkUser);  
    echo ($result);  
}
```

Je ne comprenais pas correctement le fonctionnement d'une session, il suffisait simplement de vérifier avec la méthode get, si l'utilisateur est dans session « 'username' ».

```
//get 'username'  
if ($this->session->get('username')) {  
    $result = $this->manager->getProjetUser($pkUser);  
    echo ($result);  
}
```



## Test fonctionnelle

Voici le résultat des tests fonctionnels :




| Fonctionnement             | Page ?        | Résultat attendu  | Résultat obtenu | Test validé ?   |
|----------------------------|---------------|---|-----------------|---|
| Crée un nouvel utilisateur | Register.html | Un utilisateur devrait se crée et être stocker sur la DB, | Admin crée      |  |



|   |               |  |   |   |
|---|---------------|--|---|---|
| <b>Crée un utilisateur qui est déjà existant</b>  | Register.html | Un message d'erreur devrait apparaitre.  | Impossible de crée le compte  |    |
| <b>Se connecter à un compte existant avec bon nom d'utilisateur et mot de passe.</b>                | Login.html    | La connexion devrait réussir, et la page « planning.html » devrait s'ouvrir.                     | La connexion est possible   |    |
| <b>Se connecter à un compte existant avec mauvais mot de passe ou nom d'utilisateur inexistant.</b> | Login.html    | Un message d'erreur devrait apparaitre.  | Mot de passe ou nom d'utilisateur incorrecte  |    |
| <b>Recevoir tous les projets d'un utilisateur.</b>  | Planning.html | Tous les projets qui sont assigner à l'utilisateur doivent être afficher dans le menu déroulant. | Reçoit les projets  |    |
| <b>Création d'une nouvelle tâche.</b>   | Planning.html | Une tâche doit se crée dans l'état « todo ». Elle doit aussi être stocker dans la DB.            | Peut créer un projet lorsqu'on change au moins une fois de projet. S'il y a un seul projet, impossible de crée de tâche |   |
| <b>Modifié le nom / la description d'une tâche.</b>   | Planning.html | La tâche va changer de nom / description, le changement dois aussi changer dans la DB.           | Le nom de la tâche !  |  |
| <b>Déplacer une tâche dans les états « TODO, In Progress, Done ».</b>                               | Planning.html | La tache doit changer d'état, le changement dois aussi changer dans la DB.                       | La tache se déplace correctement  |  |
| <b>Déplacer une tâche dans l'état « Validate » en tant qu'user non-administrateur.</b>              | Planning.html | Un message d'erreur devrait apparaitre, et la tâche ne dois pas changer d'état.                  | La tache change d'état dans la DB, la tache change de position en local, mais pas sur le site web.                      |  |

|   |               |  |   |   |
|---|---------------|--|---|---|
| <b>Déplacer une tâche dans l'état « Valide » en tant qu'utilisateur administrateur.</b> | Planning.html | La tâche doit changer d'état, le changement doit aussi changer dans la DB. | Comme pour le test précédent, ce test marche en local mais pas sur le site web. |  |
| <b>Déconnexion d'un utilisateur.</b>  | Planning.html | L'utilisateur doit se faire déconnecter.                                   | Utilisateur correctement déconnecter, session et local Storage.                 |  |

Et voici les tests pour tester la sécurité de mon site web.

| Fonctionnement  | Page ?   | Résultat attendu  | Résultat obtenu   | Test validé ?   |
|---|--|---|---|---|
| <b>Vérifier si des injections SQL / HTML est possible sur les champs de création de compte.</b> | Register.html                                  | Les injections de devraient pas fonctionner.  | Injection SQL ne fonctionne pas                                       |    |
| <b>Vérifier si des injections SQL / HTML est possible sur les champs de connexion.</b>          | Planning.html                                  | Les injections de devraient pas fonctionner.  | Les injections HTML ne sont pas possibles                             |   |
| <b>Vérifier qu'avec un outil comme « POSTMAN », les requêtes SQL ne fonctionnent pas.</b>       | Register.html / Connexion.html / Planning.html | Le site devrait remarquer qu'aucun user n'est connecté, et les requêtes ne devraient pas être prises en compte. | Impossible de faire des actions ou il faut être connecté avec Postman |  |

### Hébergement

Le site web est hébergé sur : <https://paccauds.emf-informatique.ch/151/client/connexion.html>

(Cependant le déplacement des tâches ne fonctionne pas de manière graphique sur le site web, mais on peut voir que les changements se font dans la DB et console.)

## 5 Synthèse

### Présentation réalisation

J'ai fait pas mal de changement du côté serveur, il y avait plein de méthode et même une fonctionnalité que je devais implémenter pour que mon programme fonctionne comme je le voulais.

### Différences entre planning et réalisation

J'ai sous-estimé le temps qu'il faudrait pour mettre en place la différente requête qui communique avec la base de données, il y a énormément de chose à penser, tel que les paramètre à avoir, les différents aspect pour protéger sa base de donnée, la requête et les différents information que nous devons retourner au client.

### Conclusion

J'ai beaucoup appris de chose durant ce module, que se soit en terme de programmation PHP, de javascript ou de sécurité, mes connaissances générales on grandement évoluer. Mise à part une fonctionnalité qui ne fonctionne pas lorsque j'héberge mon site, et quelque fonctionnalité que j'aurais pu améliorer, je trouve que le projet reste très complet.