182 - Réaliser la sécurité informatique d'un système

Rapport personnel

Romain Benedetti

Module du
01.02.2024 au 07.03.2024

Date de création : 01.02.2024

Version 1 du 01.02.2024

Table des matières

1 Introduction

1.1 Objectifs du module

- Principes de conception d'une application cliente Web.
- Principes de conception d'une application serveur Web.
- Utilisation de la norme UML pour un projet internet Client-Serveur
- Connexion entre des applications clientes et serveur Web.
- Connexion d'application serveur sur une base de données Web.
- Mise en service des principes HTML/CSS/JS (Ajax, JQuery) - PHP0.
- Utilisation des sessions dans l'application serveur Web.
- Utilisation des Cookies dans l'application cliente.
- Utilisation du cryptage des données sensibles (mot de passe)
- Protection contre les injections SQL
- Utilisation des transactions SQL
- Consommation de Webservices dans une application Web.
- Découverte et mise en pratique de standards pour les documentations de code (PHPDoc et JSDoc)
- Protocole et requêtes http de base.
- Hébergement et test final en production de l'application cliente-serveur

2 Analyse

2.1 Présentation du projet "TourisTunes"

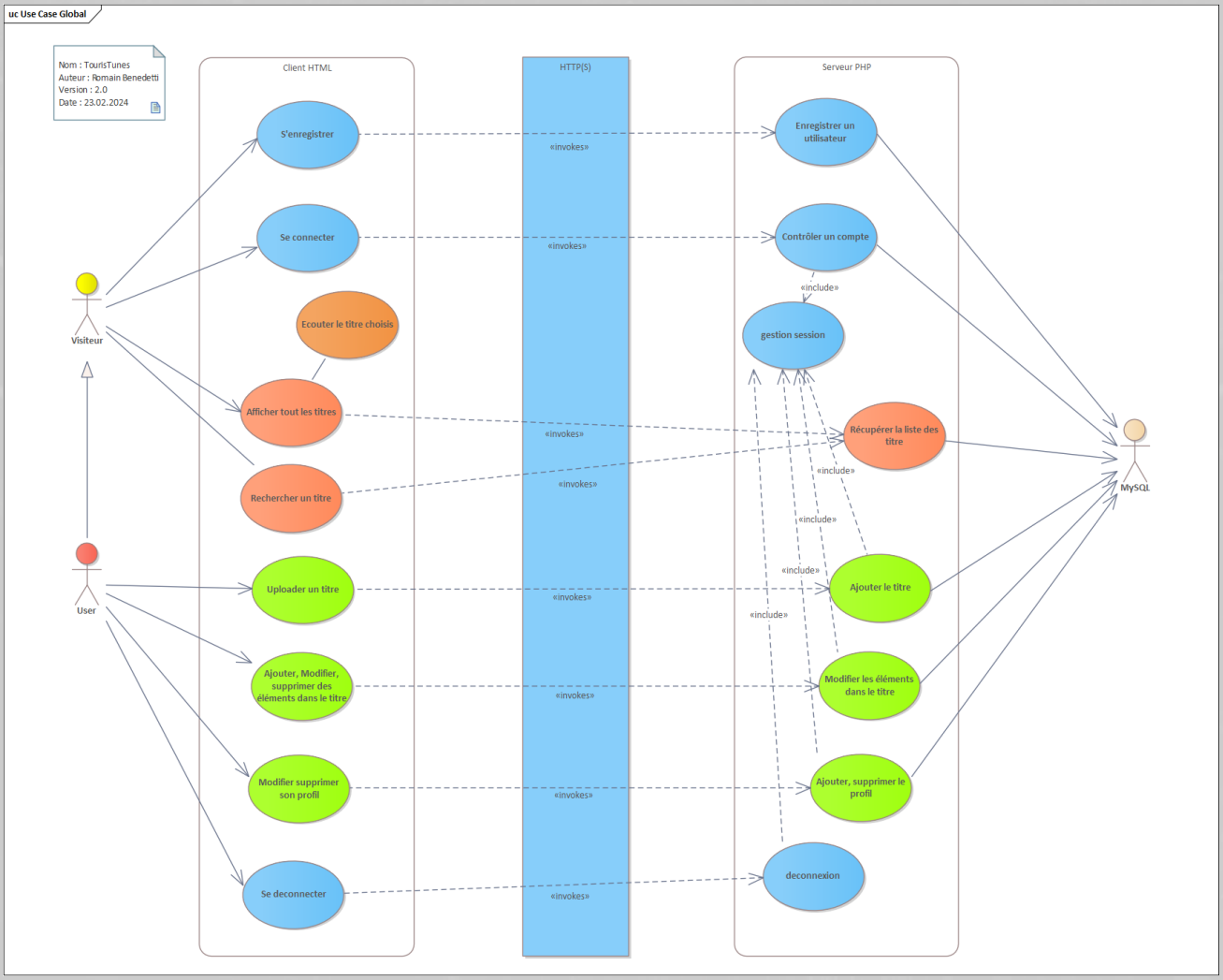
Mon projet de plateforme en ligne tire son inspiration de SoundCloud, mais avec une intention particulière : offrir aux amateurs de musique un espace dédié à la découverte et au partage de créations de sons. Les utilisateurs pourront créer un compte sur cette plateforme afin d'uploader leurs propres chansons et de les partager avec la communauté.

L'objectif principal est de créer une interface facile à utiliser et conviviale où les artistes émergents et les amateurs de musique pourront se connecter, explorer et écouter un large éventail de genres musicaux. Les visiteurs auront la possibilité d'explorer la bibliothèque de titres disponibles et d'apprécier les talents des artistes présents sur la plateforme, même sans être connectés.

Ceux qui s'inscrivent auront accès à des fonctionnalités supplémentaires telles que la possibilité de communiquer avec eux en laissant des commentaires et des likes sur leurs chansons.

En tant que producteur de musique, cette plateforme me permettra de partager mes propres créations et de recevoir des commentaires positifs d'autres artistes et auditeurs.

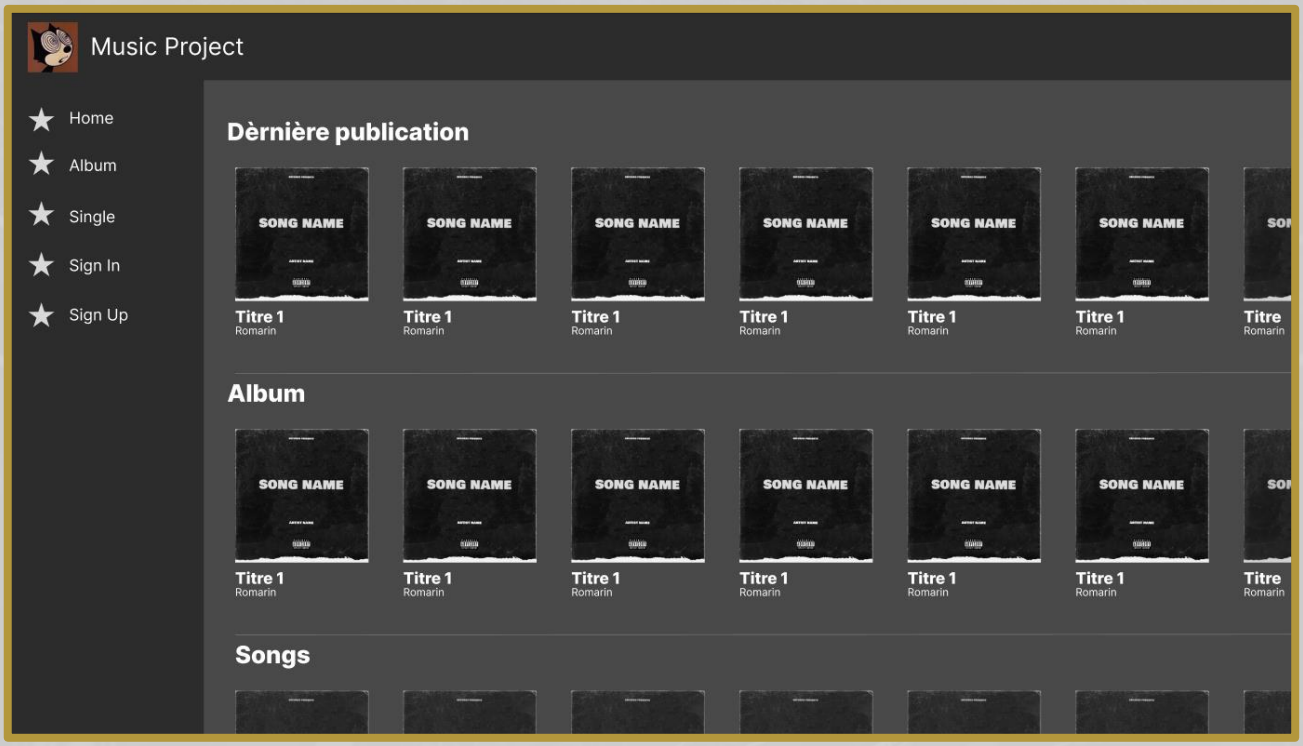
2.2 Uses Case



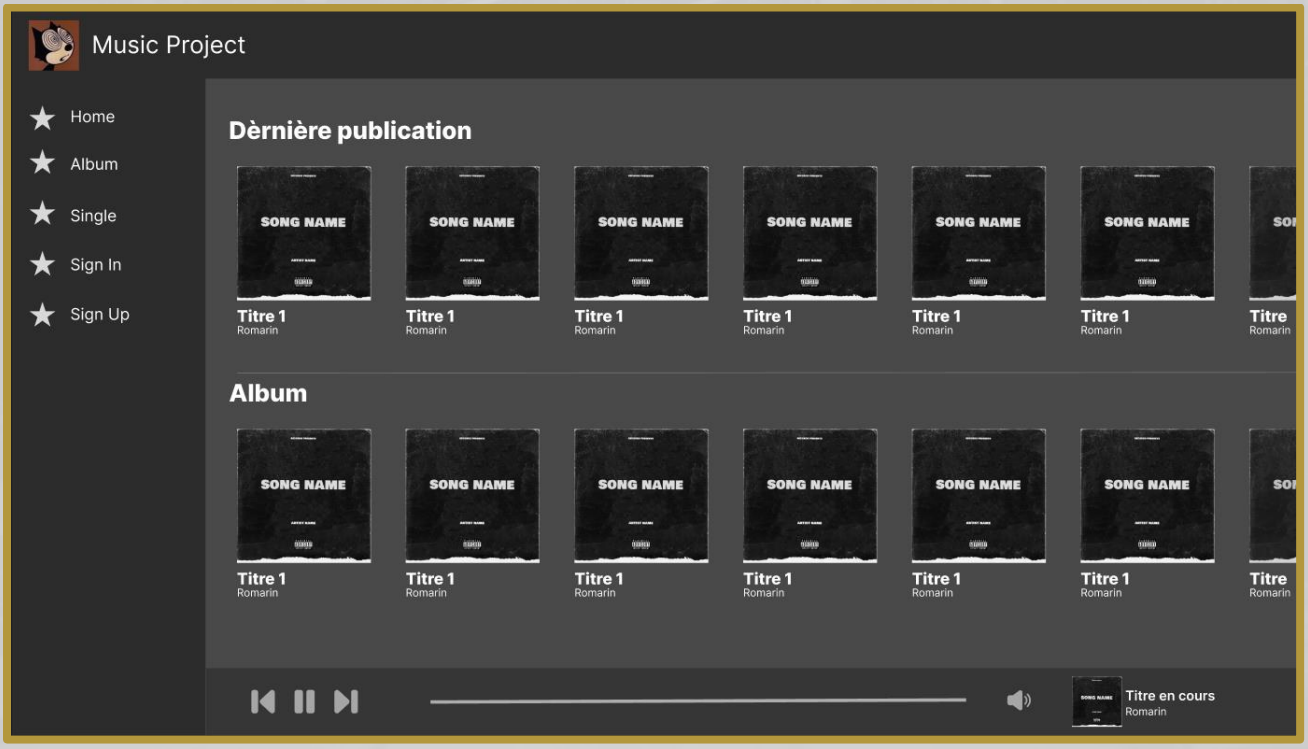
Dans ce use case on voit toutes les actions principales du côté client et côté serveur.

2.3 Maquettes

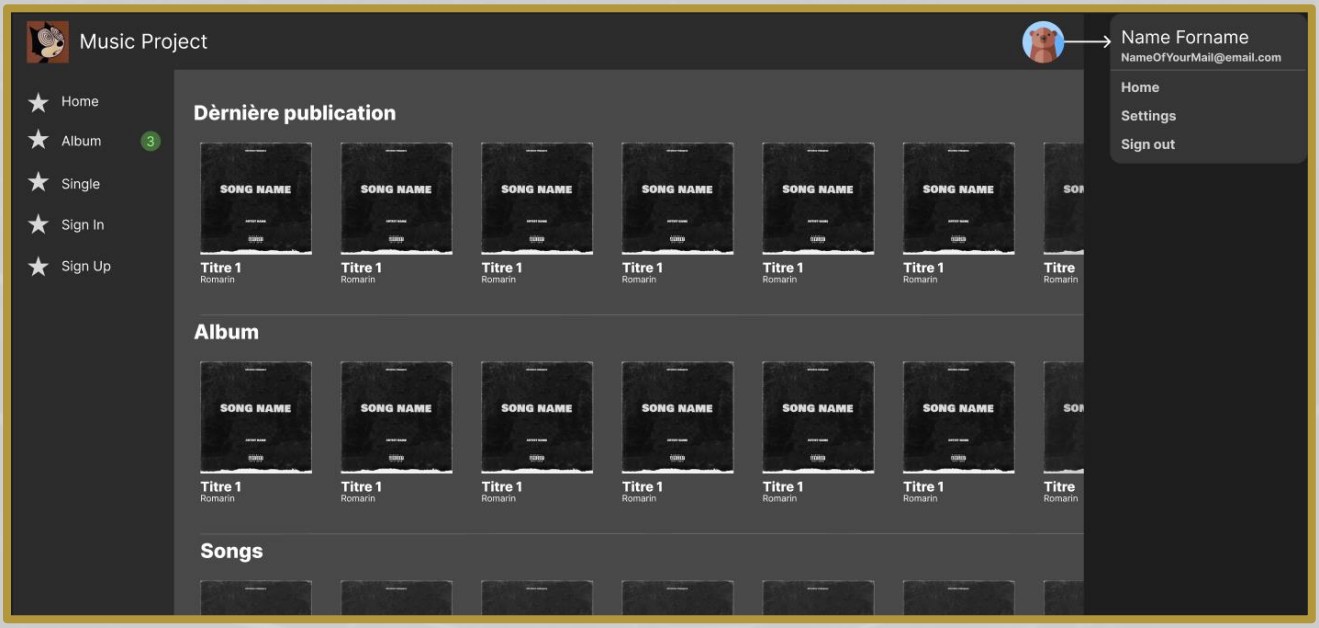
2.3.1 Non connecté



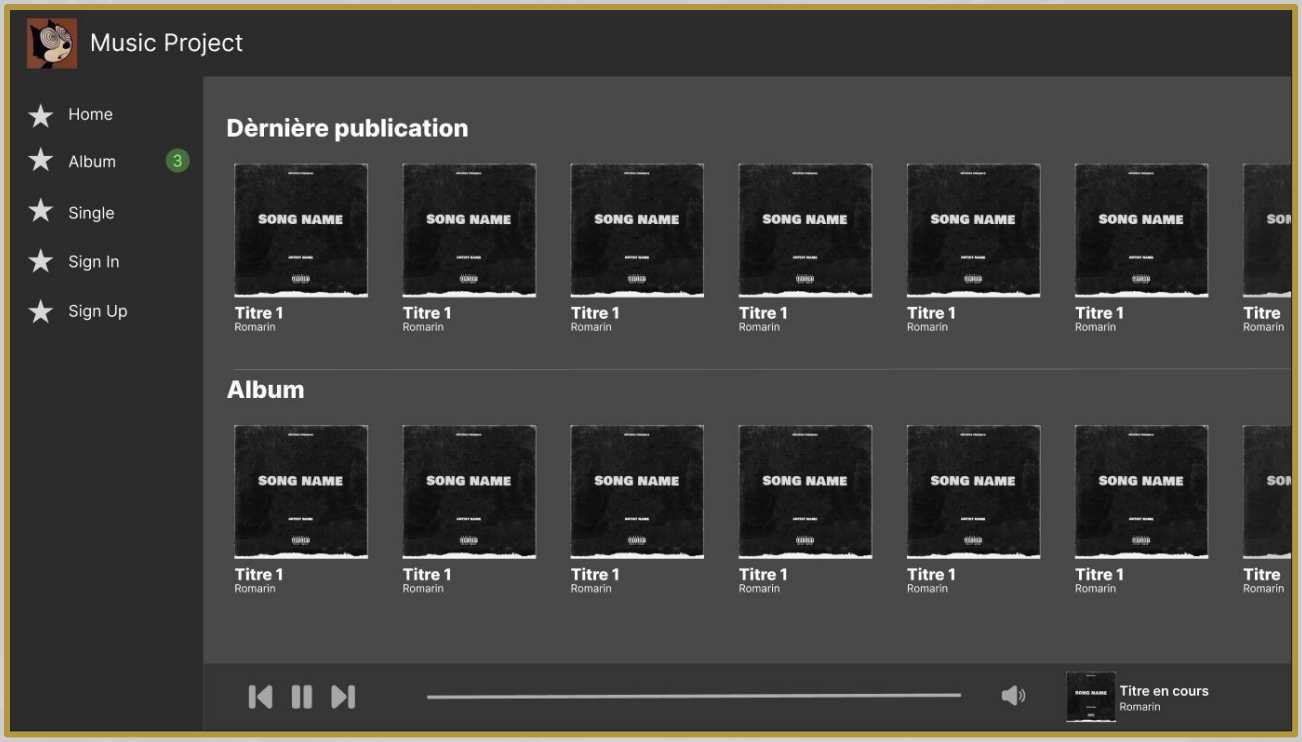
2.3.2 Non connecté – Listening music



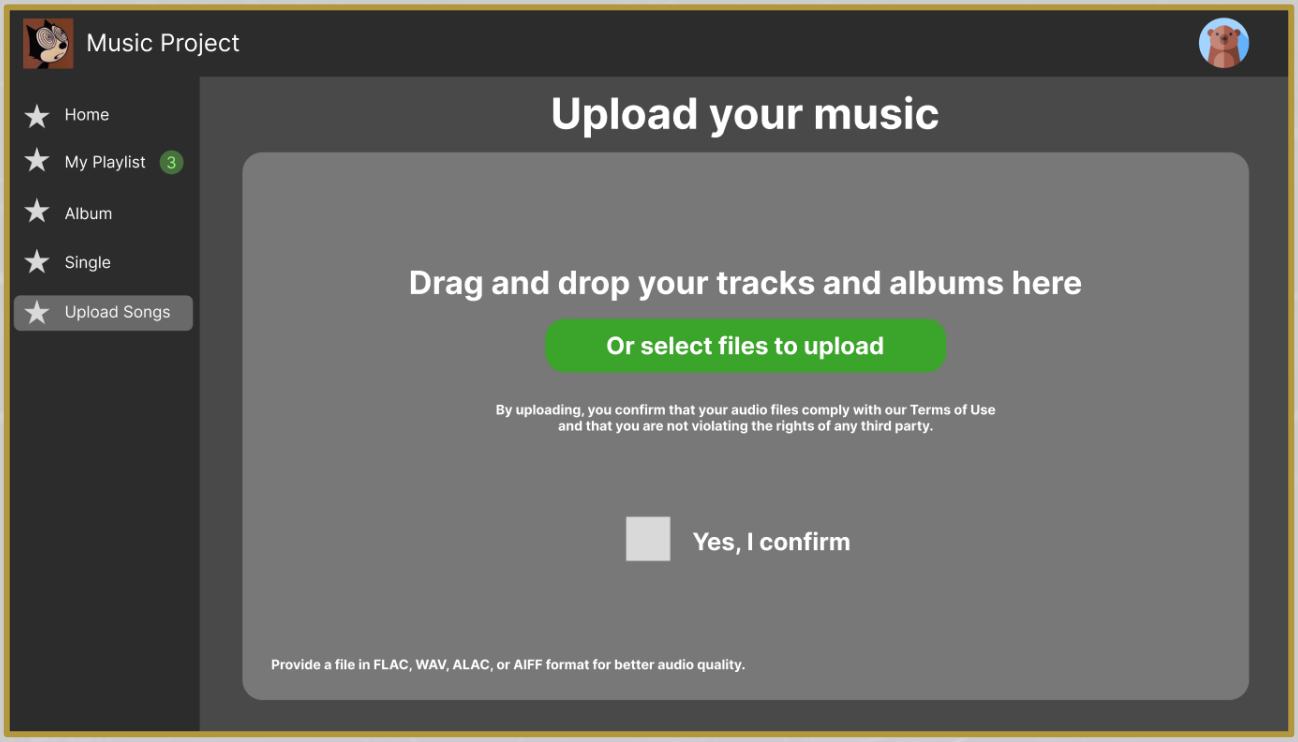
2.3.3 Connecté



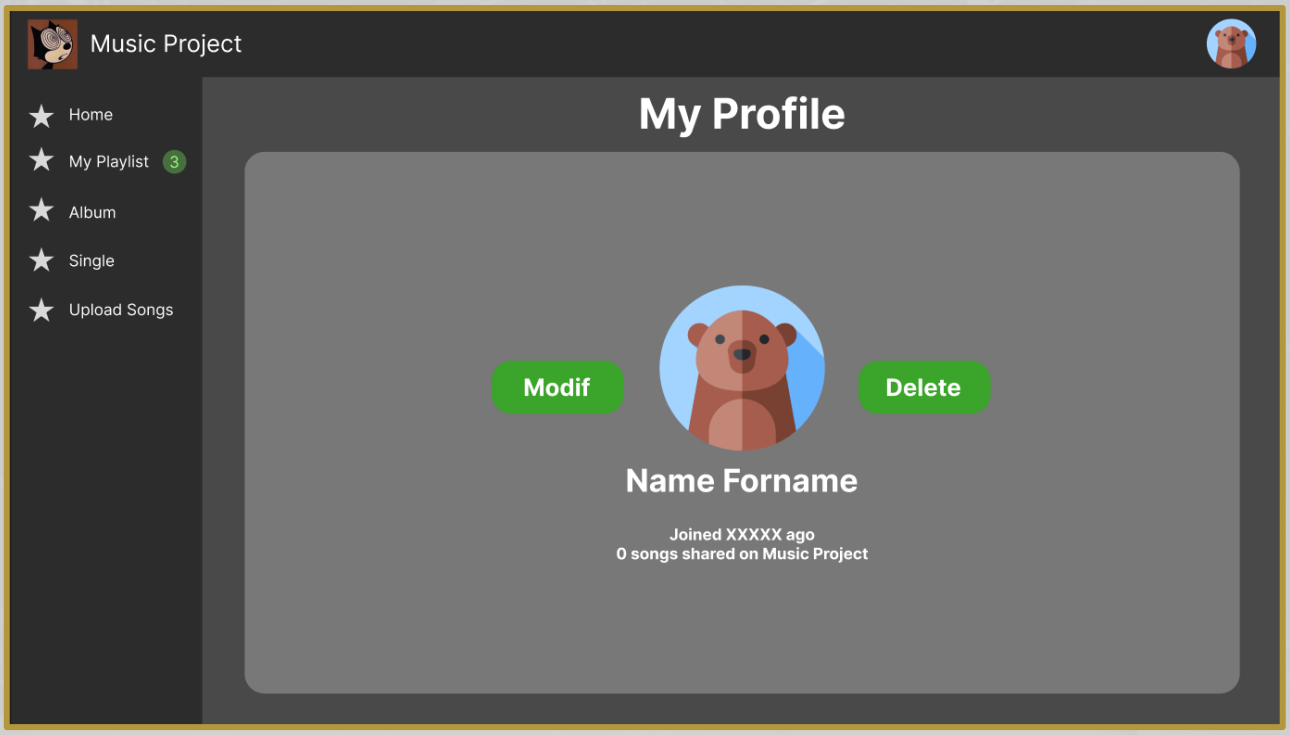
2.3.4 Connecté – Listening music



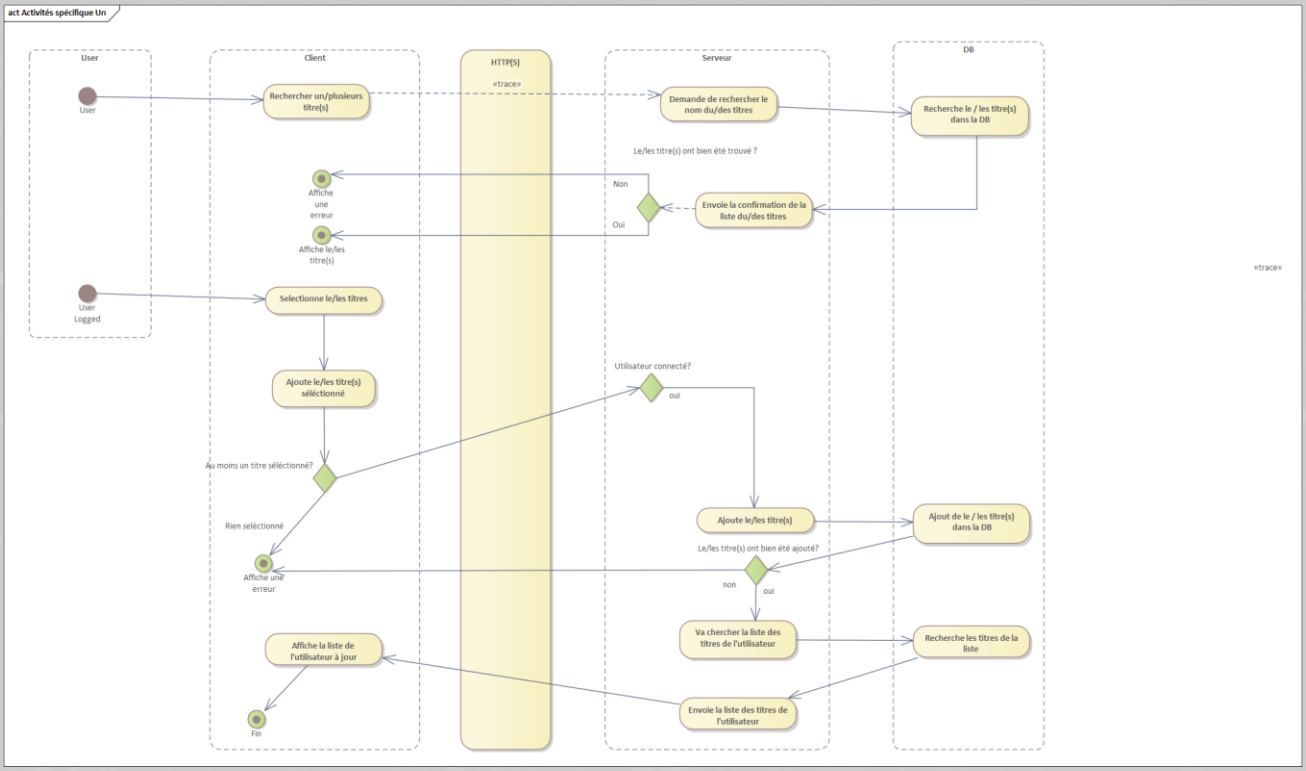
2.3.5 Upload songs



2.3.6 My profile

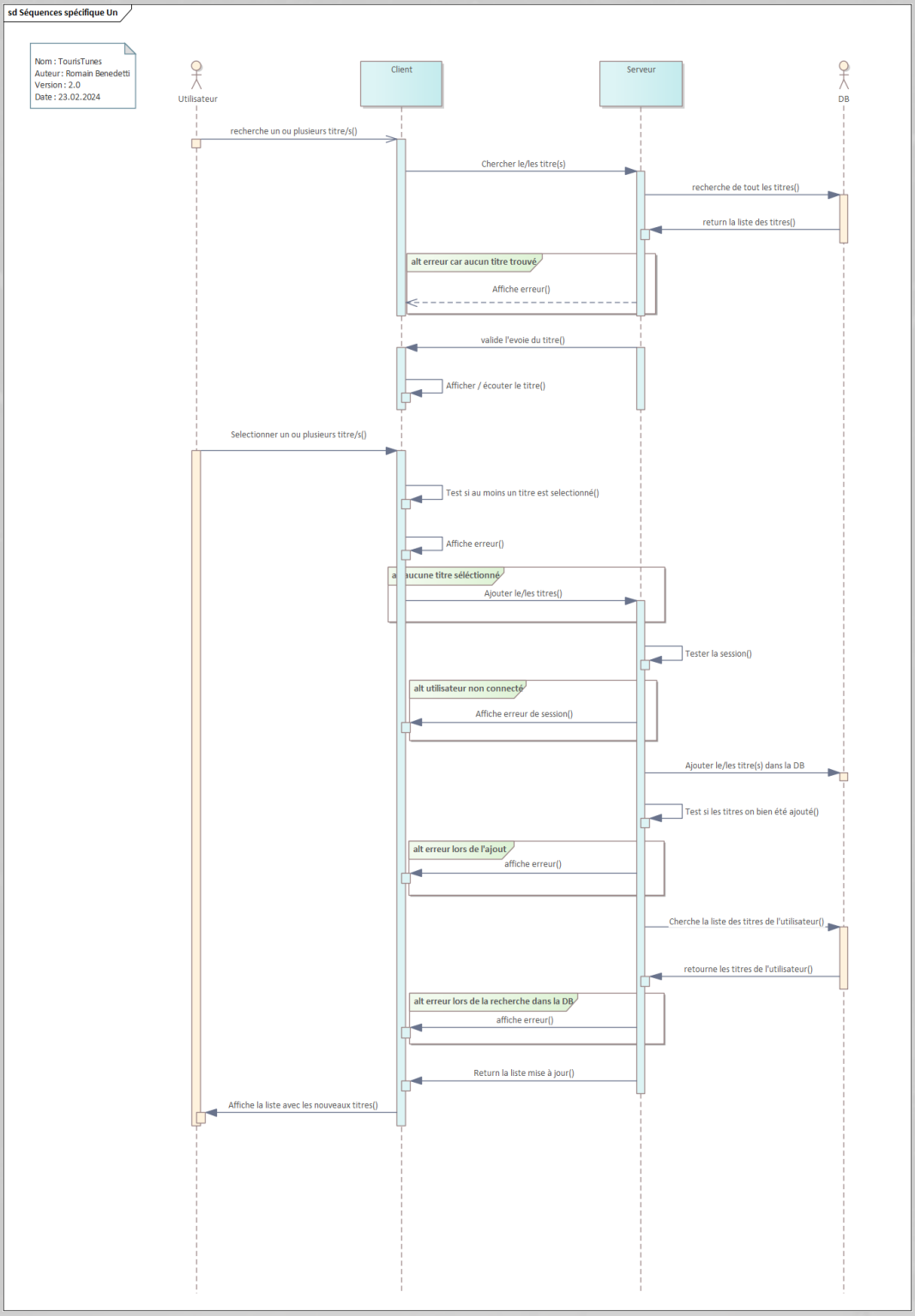


2.4 Diagramme activité



Ce diagramme d'activité montre la même chose que le diagramme de séquence mais d'une autre façon qui nous permet de mieux voir quelle partie (client ou serveur) fait quoi.

2.5 Diagramme de séquences systèmes

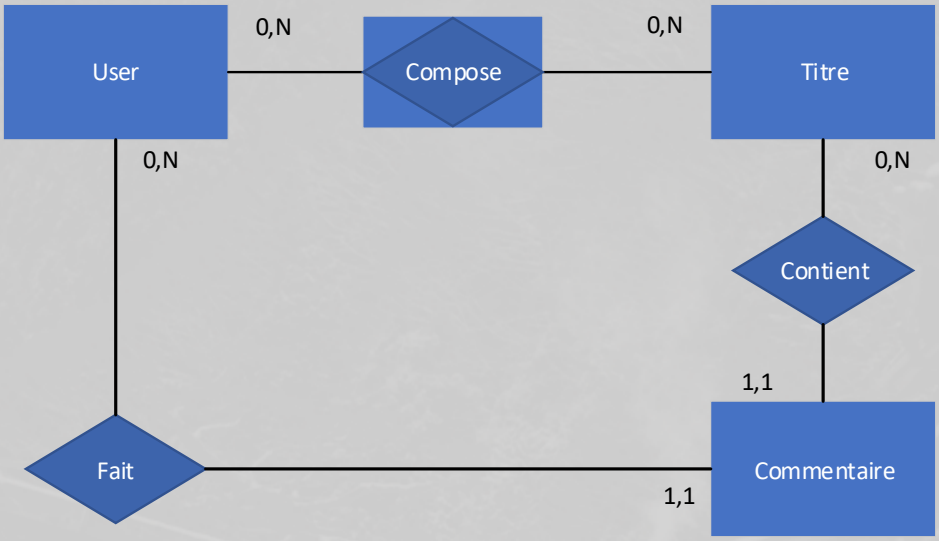


Dans ce diagramme de séquence system on voit deux actions :

Une première action va aller chercher tous les titres qui se trouve dans la db.

La seconde montre le fait d'ajouter un / des titres dans la liste de l'utilisateur. Ce dernier peut sélectionner un ou plusieurs titres puis cliquer sur ajouter pour les ajouter dans sa liste de sons. Pour finir on affiche à nouveau la liste avec les nouveaux titres.

2.6 Schéma ER



2.6.1 Identification des entités principales

- Utilisateur :
  - ID Utilisateur
  - Nom d'utilisateur
  - Mot de passe
  - Adresse e-mail
  - Autres informations utilisateur (facultatives)
- Titre :
  - ID Titre
  - Nom du titre
  - Durée
  - Format
  - Lien vers le fichier audio
  - Lien vers l'image
  - Date de publication
  - Featurings (si un titre a des collaborations)
  - Autres métadonnées du titre
- Fonctionnalités supplémentaires :
  - Commentaires

- Likes
- Données de lecture (combien de fois le titre a été écouté)

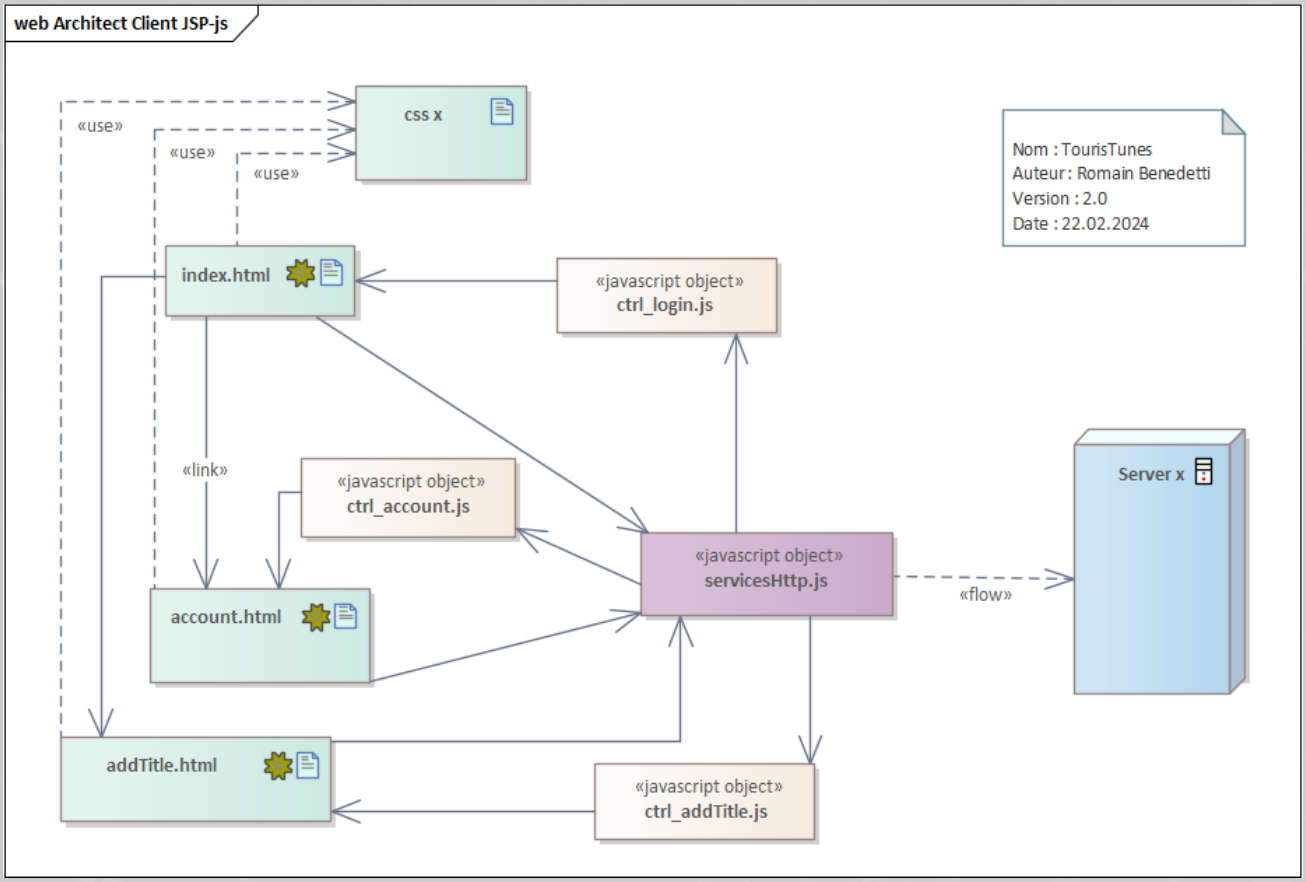
2.6.2 Relations entre les entités

Coté 1	Coté 2
Un utilisateur peut composer plusieurs titres	Un titre est composé par plusieurs utilisateurs
Un titre peut contenir plusieurs commentaires, likes, etc,	Un commentaire, like, etc., est contenu par un seul titre
Un utilisateur peut avoir plusieurs commentaires, likes, etc,	Un commentaire, like, etc., est fait par un seul utilisateur

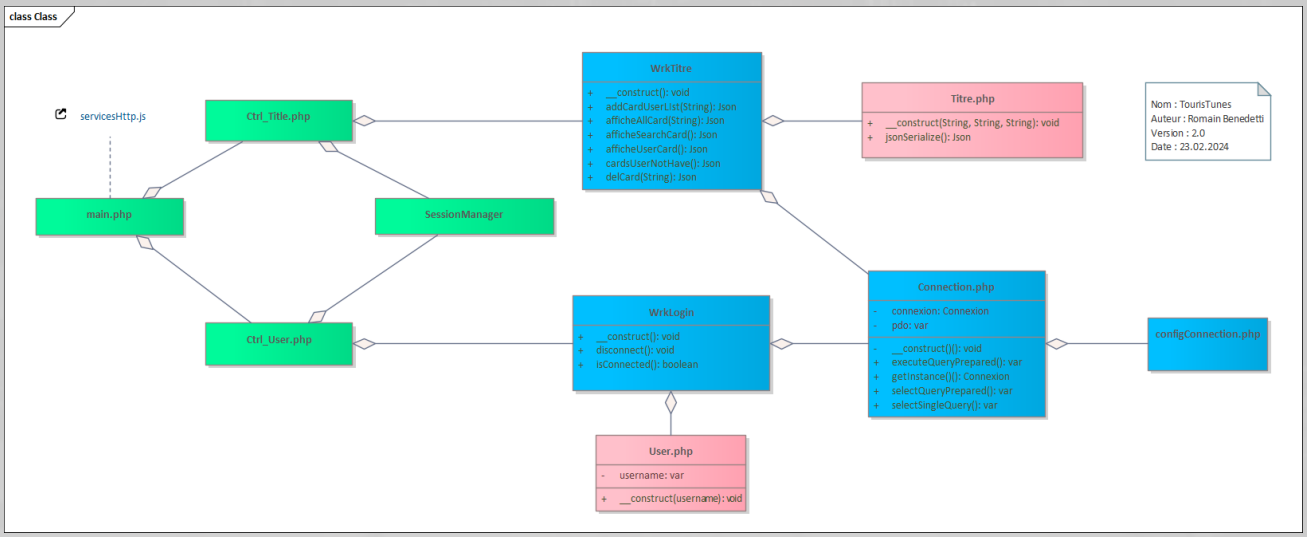
3 Conception

3.1 Diagrammes de classe

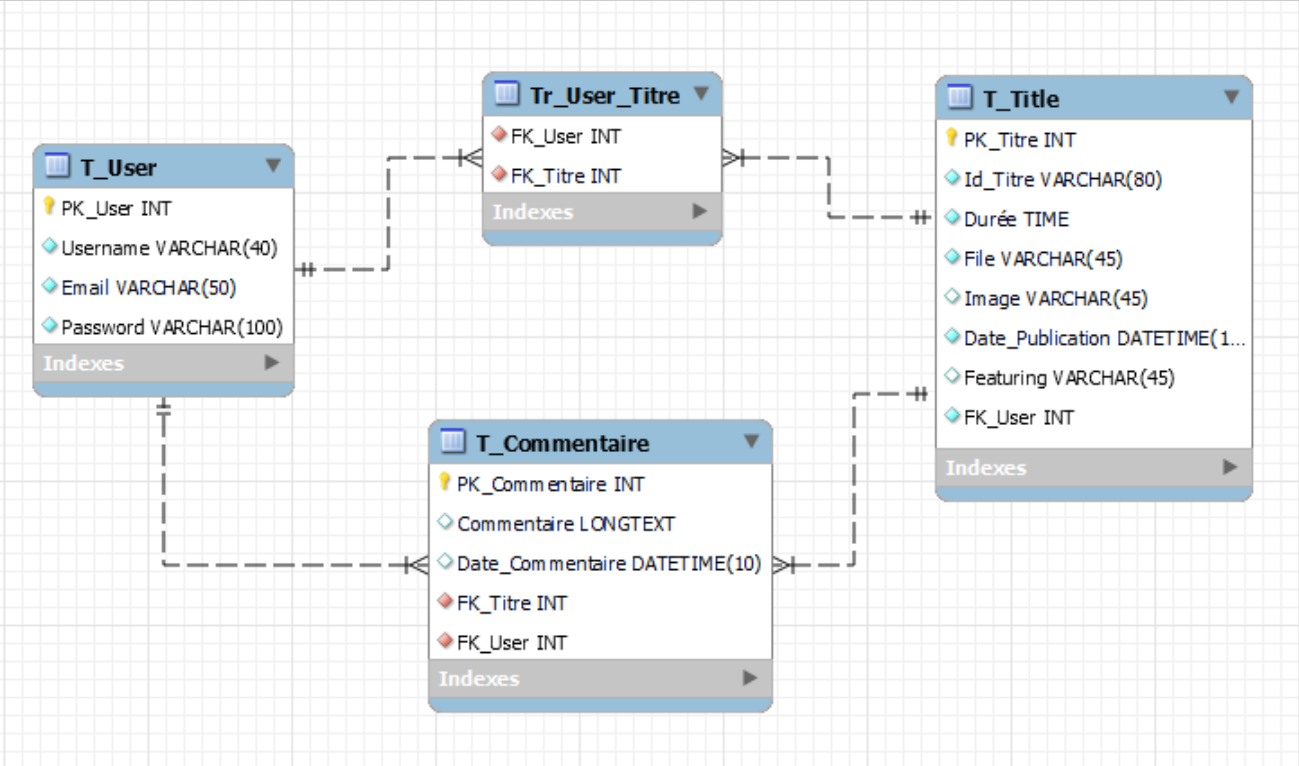
3.1.1 Client



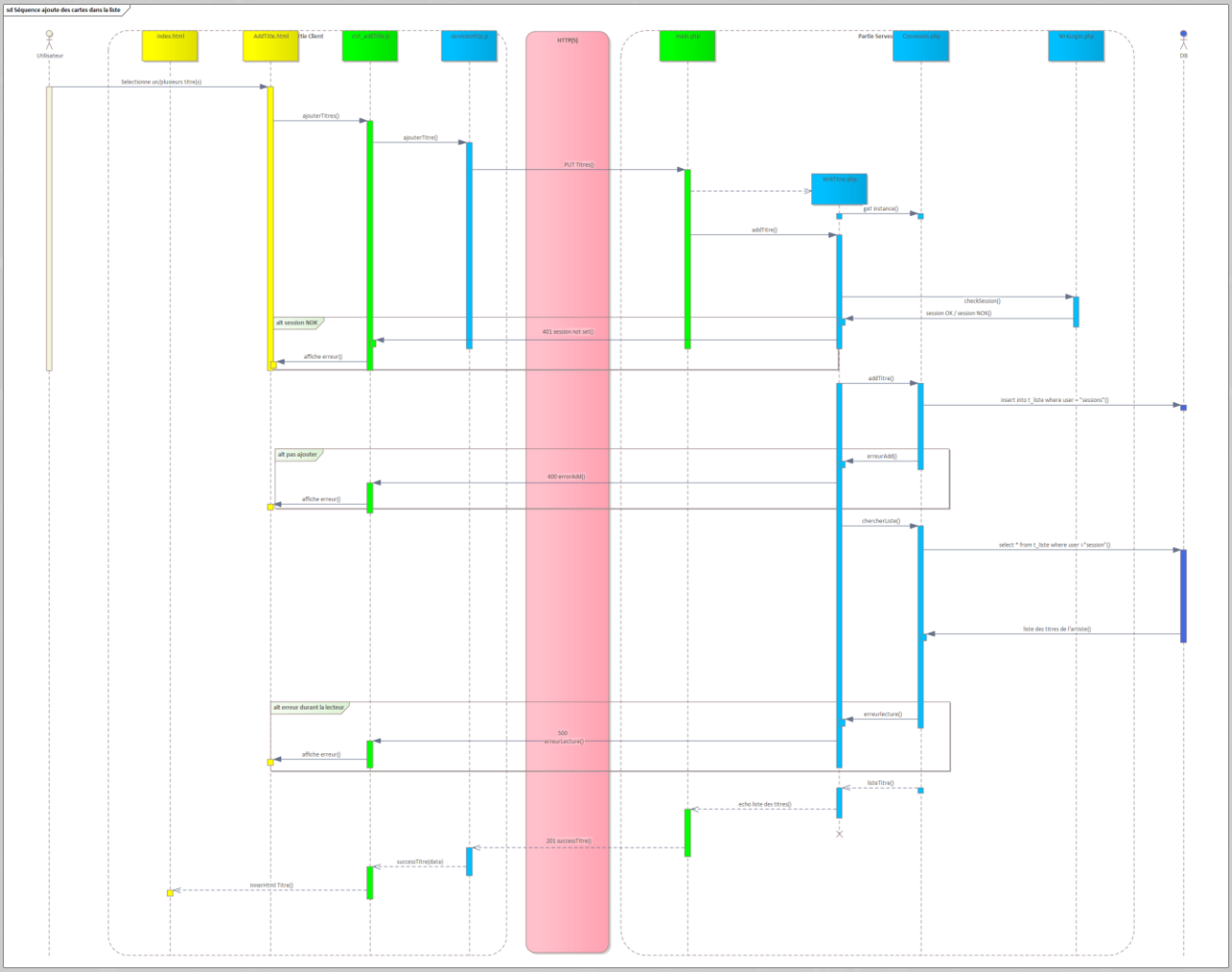
3.1.2 Serveur



3.2 Schéma relationnel



3.3 Diagramme séquence interactions



3.4 Conception des tests

Utilisateur	Cas	Tests	Attendu	Obtenu
Visiteur	Connexion	Avec login existant et mot de passe correct.	On est connecté en tant qu'artiste	
	Connexion	Avec un login qui contient une injection SQL.	L'injection ne fonctionne pas, le login n'est pas possible.	
	Recherche	Rechercher une musique	Recherche possible, le/les titre(s) s'affiche	
	Afficher	Afficher les musiques de l'utilisateur	Affichage possible utilisateur correct	
User (Artiste)	Afficher	Afficher les musiques de l'utilisateur	Affichage possible utilisateur correct	
	Ajouter	Ajouter une musique dans la liste	Ajout possible utilisateur connecté	
	Supprimer	Supprimer une musique de la liste	Suppression possible utilisateur connecté	

4 Implémentation

4.1 Descente de code

4.1.1 Login

Dans un premier temps, je vais aller récupérer grâce au formulaire les infos que l'utilisateur à taper

```
// Appeler la fonction de connexion avec les valeurs des champs
    document.getElementById('submit-login').addEventListener('click', function
(event) {
        event.preventDefault(); // Empêcher le formulaire de se soumettre
normalement

        // Récupérer les valeurs des champs username et password
        const username = document.getElementById('username-field').value;
        const password = document.getElementById('password-field').value;

        // Appeler la fonction de connexion avec les valeurs des champs
        httpServices.login(username, password, loginSuccess, loginError);
});
```

Ensuite ceci va être envoyé dans le servicesHtpp.js avec une requête ajax vers le serveur pour soumettre les infos et demander la connexion.

```
login(username, password, successCallback, errorCallback) {

        let data = {
            "action": "signIn",
            "username": username,
            "password": password
        }

        $.ajax({
            method: "POST",
            url: `${base_url}main.php`,
            dataType: "json",
            data: JSON.stringify(data),
            xhrFields: {
                withCredentials: true
            },
            success: successCallback, // Pour toutes les réponses en code 200
            error: errorCallback // Pour toutes les réponses ayant un code autre que
200, ou mal formées
        });

    },
```

Cette demande va passer dans le main.php qui lui va ensuite envoyer au ctrl_User

```
case 'POST':
            if ($data !== null) {
                switch ($data['action']) {
                    case 'signIn':
                        // Vérifier que les champs nécessaires sont présents dans
les données JSON
                        if (isset($data['username'], $data['password'])) {
                            $username = $data['username'];
                            $password = $data['password'];
                            strip_tags($username);
                            htmlspecialchars($username);
                            $userCtrl = new Ctrl_User();
                            $result = $userCtrl->signIn($username, $password);
                            echo $result;
                        } else {
                            echo 'Certains champs sont manquants dans les données
JSON !';
                        }
                        break;
                    case 'addTitle':
                        if (isset($data['state'], $data['title'])) {
                            $state = $data['state'];
                            $title = $data['title'];
                            $titleCtrl = new Ctrl_Title();
                            $result = $titleCtrl->addTitle($state, $title);
                            echo $result;
                        } else {
                            echo 'Certains champs sont manquants dans les données
JSON !';
                        }
                        break;
            } else {
                echo 'Erreur lors de la lecture des données JSON !';
            }
            break;
```

Le ctrl va regarder tester le login

```
//check si le login est ok echo 200 si oui et 401 si non
    public function TestLogin($username, $password)
    {
        if (!empty($username) && !empty($mail) && !empty($password)) {
            $user = new User(null, $username, null, $password);

            $wrk = new wrkLogin();

            $wrk->TestLogin($user);
```

```
            //Logique de la session en settant les variables de session et écriture
de la réponse en JSON

        } else {
            echo 400;
        }
    }

public function signIn($username, $password)
    {
        $wrk = new wrkLogin();
        $user = $wrk->getUserByUsername($username);

        if ($user) {
            // Vérifier le mot de passe entré par l'utilisateur avec password_verify
            if (password_verify($password, $user->getPassword())) {
                // Mettre la pk du user dans $_SESSION à l'aide du session manager
                $_SESSION['pk_user'] = $user->getPk();
                $_SESSION['username'] = $user->getUsername();
                $_SESSION['email'] = $user->getEmail();
                // Renvoye les variables
                echo json_encode(array('success' => true, 'message' => 'Login
successful', 'pk_user' => $_SESSION['pk_user'], 'username' => $_SESSION['username'],
'email' => $_SESSION['email']));
            } else {
                // Renvoye un message si l'user n'existe pas
                http_response_code(401); //informations d'identification
incorrectes.
                echo json_encode(array('success' => false, 'message' => 'User not
found or Incorrect password'));
            }
        } else {
            // Renvoye un message si l'user n'existe pas
            http_response_code(401); //informations d'identification incorrectes.
            echo json_encode(array('success' => false, 'message' => 'User not found
or Incorrect password'));
        }
    }
```

Ensuite il va se connecter en allant vers le wrkLogin

```
// Vérifie le login et retourne vrai si c'est correct
    public function TestLogin($user)
    {
        $return = false;
        $params = array("username" => $user->getUsername(), "email" => $user-
>getMail());
```

```
        // Vérifier si l'utilisateur existe avec le nom d'utilisateur ou l'adresse
e-mail
        $userData = $this->connection->selectSingleQuery('SELECT * FROM T_user WHERE
username=:username OR email=:email', $params);
        if ($userData) {
            // L'utilisateur existe
            $password = $userData['password'];
            if (password_verify($user->getPassword(), $password)) {
                // Le mot de passe correspond
                $return = true;
            } else {
                // Le mot de passe est incorrect
                $return = false;
            }
        } else {
            // L'utilisateur n'existe pas
            $return = false;
        }
        return $return;
    }

    public function getUserByUsername($username)
    {
        $params = array("username" => $username);

        // Vérifier si le nom d'utilisateur existe
        $existingUser = $this->connection->selectSingleQuery('SELECT * FROM T_User
WHERE username=:username', $params);

        if ($existingUser) {
            $user = new User(
                $existingUser['PK_User'],
                $existingUser['Username'],
                $existingUser['Email'],
                $existingUser['Password']
            );

            return $user;
        } else {
            return false;
        }
    }
```

Et ensuite le worker va faire les demandes à la base de données et regarder le user (le compte artiste)

4.1.2 Afficher les 10 derniers titres

Pour afficher les 10 derniers titres, cela se fera automatiquement lorsqu'un visiteur arrive sur le site. Au début on va chercher tous les titres. Puis on va dans le servicesHttps faire la demande en ajax.

```
$().ready(() => {
    httpServices.loadMusics(successCallback, errorCallback);
});
```

(serviceshttp.js dessous)

```
loadMusics(successCallback, errorCallback) {
        $.ajax({
            type: "GET",
            dataType: "json",
            url: `${base_url}main.php`,
            data: `action=listAllTitles`,
            xhrFields: {
                withCredentials: true
            },
            success: successCallback, // Pour toutes les réponses en code 200
            error: errorCallback // Pour toutes les réponses ayant un code autre que
200, ou mal formées
        });
    },
```

Ensuite dans le main.php on va faire la demande de tous les titres.

```
ase 'GET':
            if ($_GET != null) {
                switch ($_GET['action']) {
                    case 'listAllTitles':
                        $TitleCtrl = new Ctrl_Title();
                        $result = $TitleCtrl->listAllTitles();
                        echo $result;
                        break;
                }
            } else {
                echo 'Erreur lors de la lecture des données JSON !';
            }
            break;
```

Une fois dans le ctrl on va récupérer tous les titres et le retourner en json

```
public function listAllTitles()
  {
    $wrk = new wrkTitre();

    // Appeler la méthode de récupération des titres depuis la base de données
```

```
    $titles = $wrk->getAllTitlesFromDB();

    // Retourner les titres au format JSON ou sous une autre forme selon votre
besoin
    return json_encode($titles);
  }
```

Dans le worker, je vais recup tous ce qui se trouve dans la table T_Title et rechercher aussi dans un tableau le titre lié à l'artiste

```
public function getAllTitlesFromDB()
    {
        $res = [];
        $i = 0;
        $params = array();
        $rows = Connexion::getInstance()->SelectQuery("SELECT * FROM t_title INNER
JOIN tr_user_titre ON PK_Titre = FK_Titre INNER JOIN t_user ON PK_User = FK_User
ORDER BY t_title.PK_Titre DESC LIMIT 10
        ", $params);

        foreach ($rows as $row) {
            $music = new Titre();
            $music->initFromDb($row);

            $res[$i] = [
                "name" => $music->getId_Titre(),
                "Durée" => $music->getDuree(),
                "File" => $music->getFile(),
                "Image" => $music->getImage(),
                "Date_Publication" => $music->getDate_Publication(),
                "username" => $row["Username"]
            ];
            $i++;
        }
        return $res;
    }
```

4.1.3 Se déconnecter

Lorsqu'un user se déconnecte, il va faire la demande et recevoir une réponse pour savoir si tout c'est bien passé. Il va faire le même chemin comme lorsqu'on se connecte sauf cette fois on veut supprimer la session.

```
function logout() {
    httpServices.disconnect(disconnectSuccess, disconnectError);
    loggedIn = false;
}

function disconnectSuccess(response) {
```

```
    // Traiter la réponse du serveur
    if (response.success) {
        // Déconnexion réussie, mettre à jour l'état de connexion
        console.log('Logout successful');
    } else {
        // Afficher un message d'erreur à l'utilisateur
        console.error('Logout failed:', response.message);
    }
}

function disconnectError(error) {
    console.error('Error:', error);
}
```

ServicesHttp.js

```
disconnect(successCallback, errorCallback) {

        let data = {
            "action": "disconnect"
        }

        $.ajax({
            type: "POST",
            dataType: "json",
            url: `${base_url}main.php`,
            data: JSON.stringify(data),
            xhrFields: {
                withCredentials: true
            },
            success: successCallback, // Pour toutes les réponses en code 200
            error: errorCallback // Pour toutes les réponses ayant un code autre que
200, ou mal formées
        });
    },
```

Dans le main.php

```
case 'disconnect':
       $userCtrl = new Ctrl_User();
       $result = $userCtrl->disconnect();
       echo $result;
break;
```

Ctrl_User.php

```
 public function disconnect()
    {
```

```
        $wrk = new wrkLogin();
        return $wrk->disconnect();
    }
```

wrkUser.php

```
function disconnect()
    {
        // Vérifie si l'utilisateur est connecté avant de détruire la session
        if (isset($_SESSION['username'])) {
            // Détruire toutes les données de session
            $_SESSION = array();

            // Supprimer le cookie de session s'il existe
            if (ini_get("session.use_cookies")) {
                $params = session_get_cookie_params();
                setcookie(
                    session_name(),
                    '',
                    time() - 42000,
                    $params["path"],
                    $params["domain"],
                    $params["secure"],
                    $params["httponly"]
                );
            }

            session_unset(); // Détruit toutes les variables de session
            session_destroy(); // Détruit la session
            return json_encode(array('success' => true, 'message' => 'Logout
successful')); // Retourne true si la déconnexion est réussie
        } else {
            http_response_code(401); //informations d'identification incorrectes.
            return json_encode(array('success' => false, 'message' => 'Logout
failed')); // Retourne false si l'user n'était pas connecté
        }
    }
```

4.2 Problèmes rencontrés

J'ai eu plusieurs problèmes lorsque je voulais afficher tous les titres car la commande ne convenait pas, il fallait faire attention dans mon worker lorsque que j'allais sélectionner les titres, il manquait la table de relation me permettant de savoir qui a fait ce titre

```
        $rows = Connexion::getInstance()->SelectQuery("SELECT * FROM t_title INNER
JOIN tr_user_titre ON PK_Titre = FK_Titre INNER JOIN t_user ON PK_User = FK_User
ORDER BY t_title.PK_Titre DESC LIMIT 10
```

Une fois cela trouvé j'ai un autre problème, il manquait le fait d'avoir l'artiste dans un tableau car celui-ci ne se trouvais pas dans la table T_Title.

```
"username" => $row["Username"]
```

4.3 Tests fonctionnels

Cas	Tests	Attendu	Obtenu
Connexion	Avec login existant et mot de passe correct.	On est connecté en tant qu'artiste	Réussi, l'user peut se connecter et déconnecter
Connexion	Avec un login qui contient une injection SQL.	L'injection ne fonctionne pas, le login n'est pas possible.	Impossible
Connexion	Création compte avec un email / utilisateur déjà utilisé	Bloque la création du compte	Manque de temps ceci n'a pas était fait
Recherche	Rechercher une musique	Recherche possible, le/les titre(s) s'affiche	Manque de temps ceci n'a pas était fait
Recherche	Recherche avec une injection SQL.	Recherche et affiche le/les titre(s)	Manque de temps ceci n'a pas était fait
Afficher	Afficher les musiques de l'utilisateur	Affichage possible utilisateur correct	Affiche les 10 derniers titres publié
Afficher	Afficher les musiques de l'utilisateur	Affichage possible utilisateur correct	Affiche le titre avec le nom de l'artiste
Ajouter	Ajouter une musique dans la liste	Ajout possible utilisateur connecté	Manque de temps ceci n'a pas était fait
Supprimer	Supprimer le compte avec les musiques	Suppression possible	Manque de temps ceci n'a pas était fait

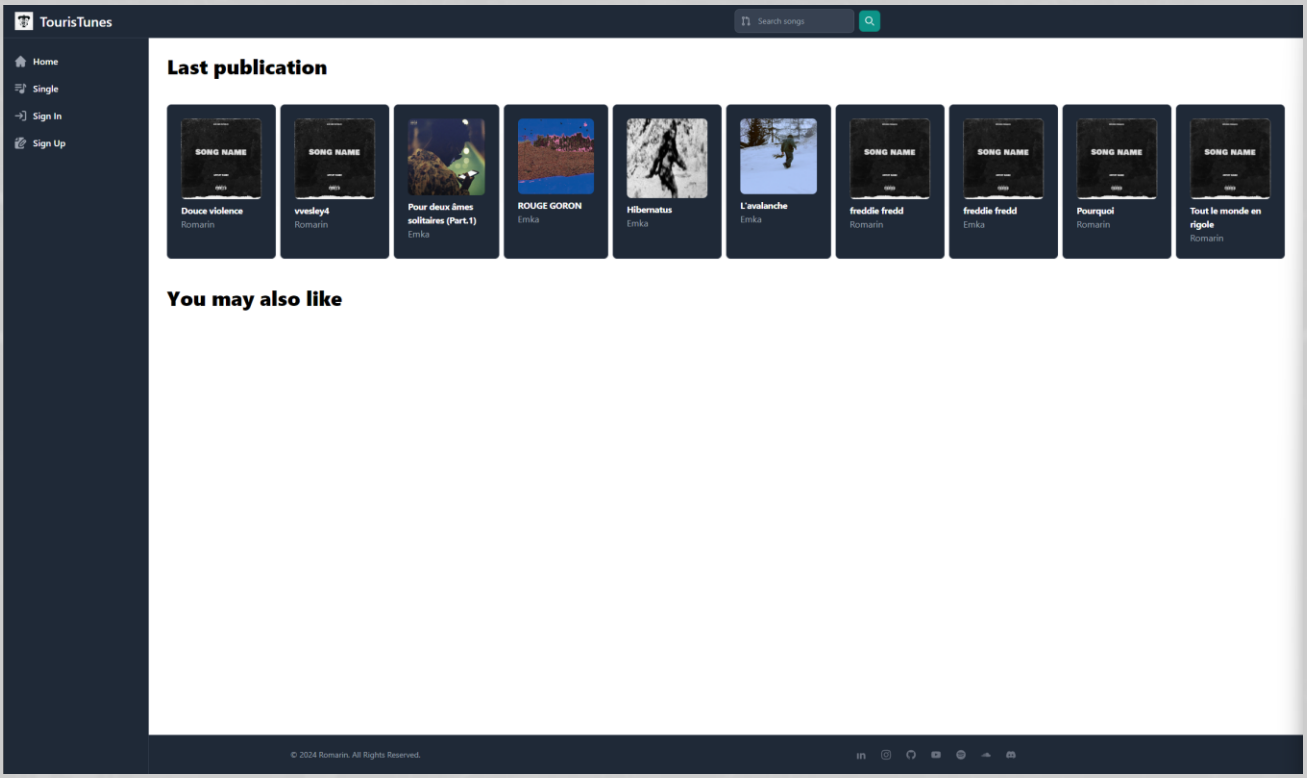
Supprimer	Supprimer une musique de la liste	Suppression possible utilisateur connecté	Manque de temps ceci n'a pas était fait

4.4 Hébergement

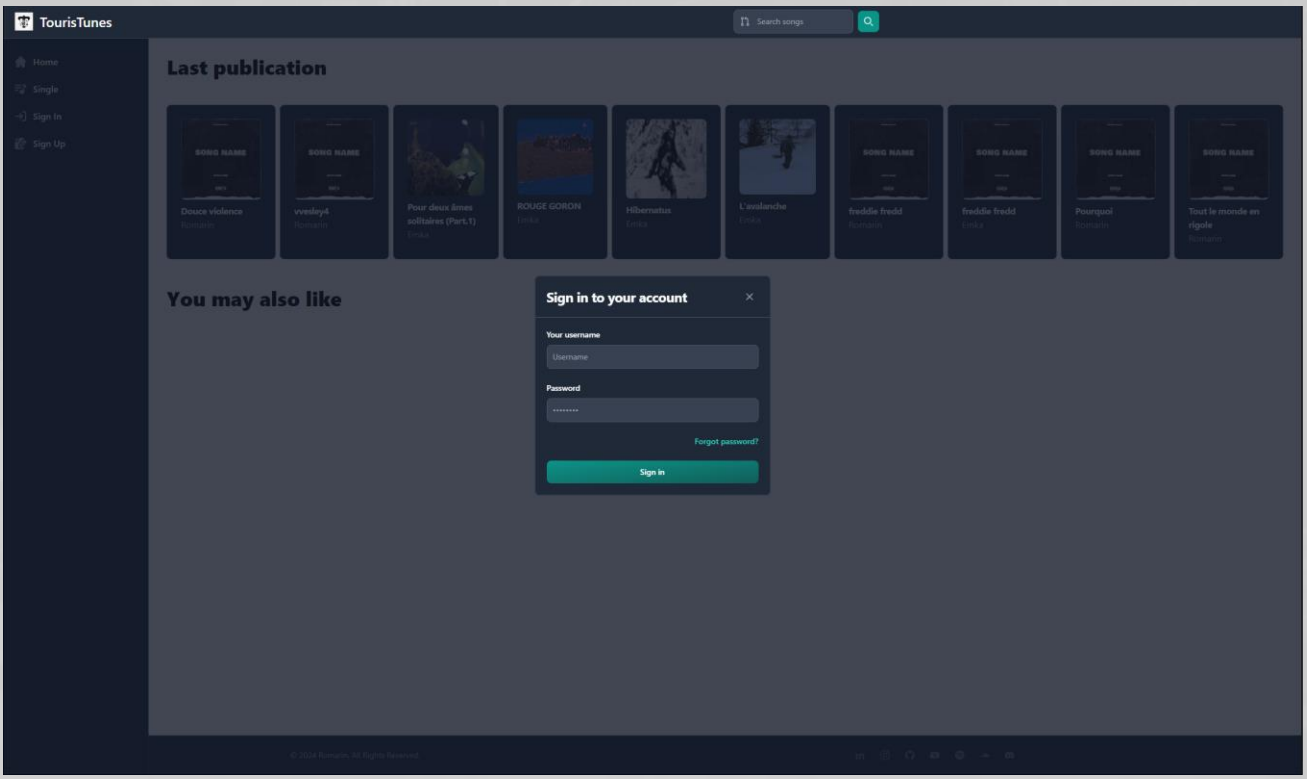
Le site est hébergé sous https://benedettir.emf-informatique.ch/151/client/index.html mais le serveur ne réagit pas comme avec docker.

5 Synthèse

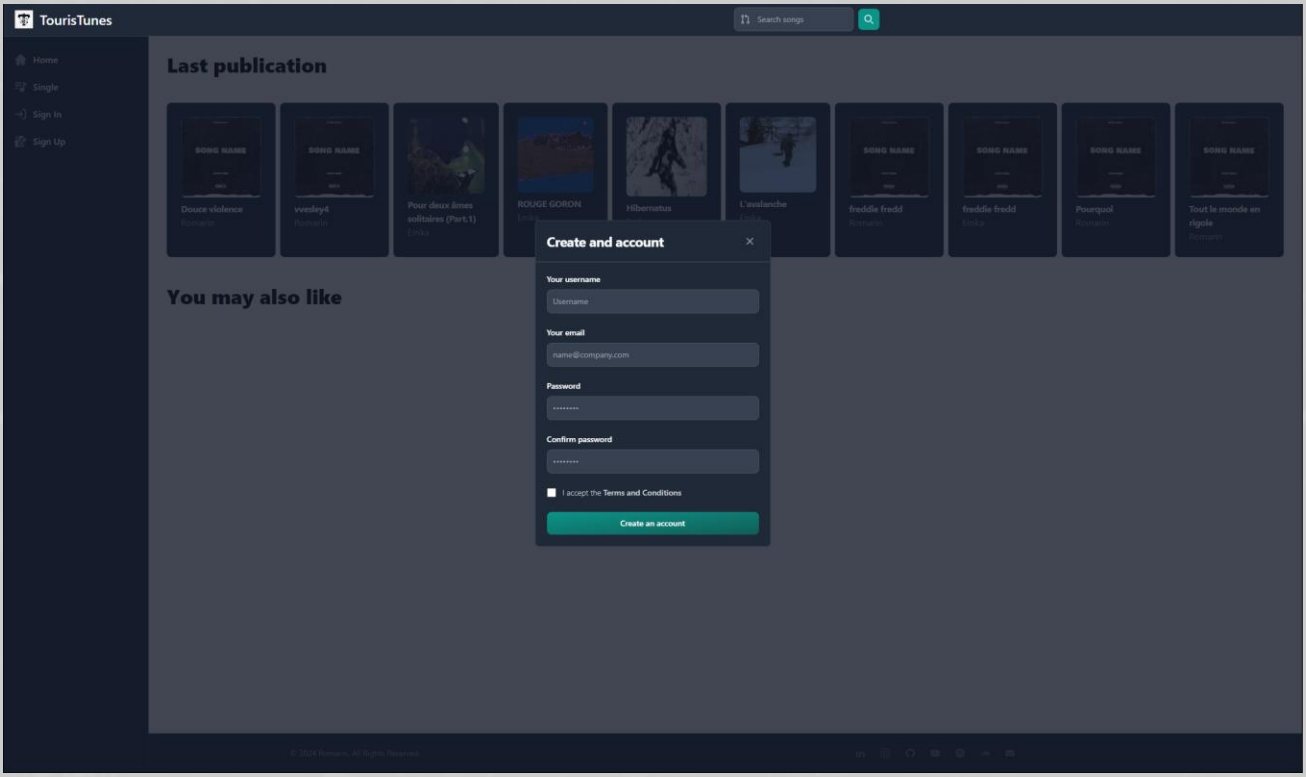
5.1 Accueil du site



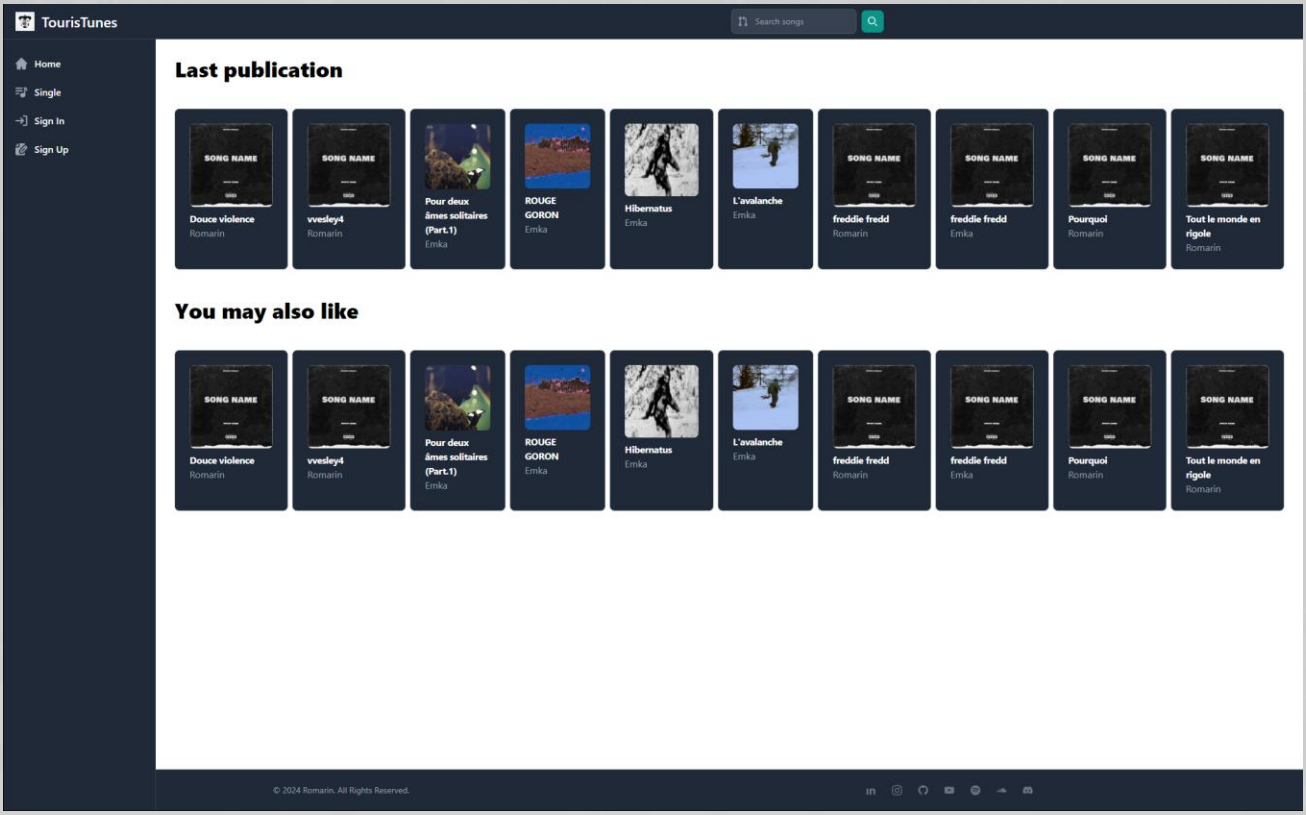
5.2 Sign In



5.3 Sign Up



5.4 Logged



6 Conclusion

Finalement, ce projet m'a pris beaucoup de temps et je me sentais un peu ambitieux. Mais cela m'a donné une bonne base pour le continuer. Ce projet pourrait me servir à l'avenir lorsque je voudrais publier des sons.

Ce module m'a beaucoup appris sur la programmation web et j'ai eu une bonne expérience, malgré l'énorme quantité d'énergie fournie.