Rapport Projet Web : Bakaraoke

Cecconi Quentin Chatelet Leo Goyard Louis
Sadler Alec

May 2020

Table des matières

1 Introduction

Lors de conventions basées sur la culture japonaise(japanExpo, bakanim), il n'est pas rare d'y trouver des séances de Karaoké, ou tout un public se regroupe et chante au rythme des plus grands chefs d'oeuvre nippons.
Cependant, si les visiteurs viennent à ces séances, c'est qu'ils s'attendent à chanter des musiques qu'ils connaissent et aiment. Pourtant les playlists sont souvent préfaites, et les "chanteurs" n'ont pas la possibilité de choisir sur quel hymne ils vont épuiser leurs cordes vocales.
Voici donc le site Bakaraoké : un gestionnaire de playlist où les utilisateurs peuvent rajouter à leur guise les musiques qu'ils veulent chanter.

Nous étions 4 pour ce projet, la répartition des tâches fut :
— Châtelet Leo : implémentation des playlists
— Cecconi Quentin : style css et pages html
— Goyard Louis : implémentation de la queue et interface lektor
— Sadler Alec : système d'utilisateurs, formulaires et sécurité

2 Structure du projet

2.1 Modèle Vue/Controlleur

Pour l'apparence visuel du site nous avons adopté le modèle vue/controlleur. Cela s'avère particulièrement utile pour l'affichage du header qui permet à l'utilisateur d'accéder à toutes les fonctionnalités du site, et ce quel que soit la page sur laquelle il se trouve actuellement.
Chaque page du site inclus donc la vue de ce header en entête, puis une vue de la page en question. Par exemple la page login.php inclus la vue header.php ainsi que la vue de login.php (une autre page qui n'est pas dans public).
L'affichage des images de profil est aussi gérée par des vues et plus précisément par des fonctions php qui permettent d'afficher une image et de décider de ses dimensions ainsi que de sa forme en retournant une balise avec du CSS.
On retrouve dans le dossier `/public/Forms` les fichiers php qui s'occupe de la manipulation de la base de données côté serveur, avec par exemple `login.php` ou `addKara.php`.
Les interfaces User, Lector, Playlist, et Kara se trouvent respectivement dans leur dossier respectifs dans le `/src`.

2.2 Apparence visuelle

Pour assurer qu'aucun style directement implémenté dans le html ne se fasse écraser par la feuille de style, l'utilisation de classe a été privilégiée pour mieux contrôler les balises sur lesquelles le style s'applique. Une feuille de style, style.css, est commune à toutes les pages (car présente dans le `src/View/Layout/head.php` que l'on inclus systématiquement). Il est cependant possible d'intégrer d'autres feuilles de style pour des pages spécifiques.

2.3 la base de donnée

On rajoute plusieurs tables à la base de données pour pouvoir gérer les cosmétiques de chaque utilisateur, leurs playlists, la queue et la liste des lecteurs du côté de lektor. Voici le diagramme UML de la base de données du site :

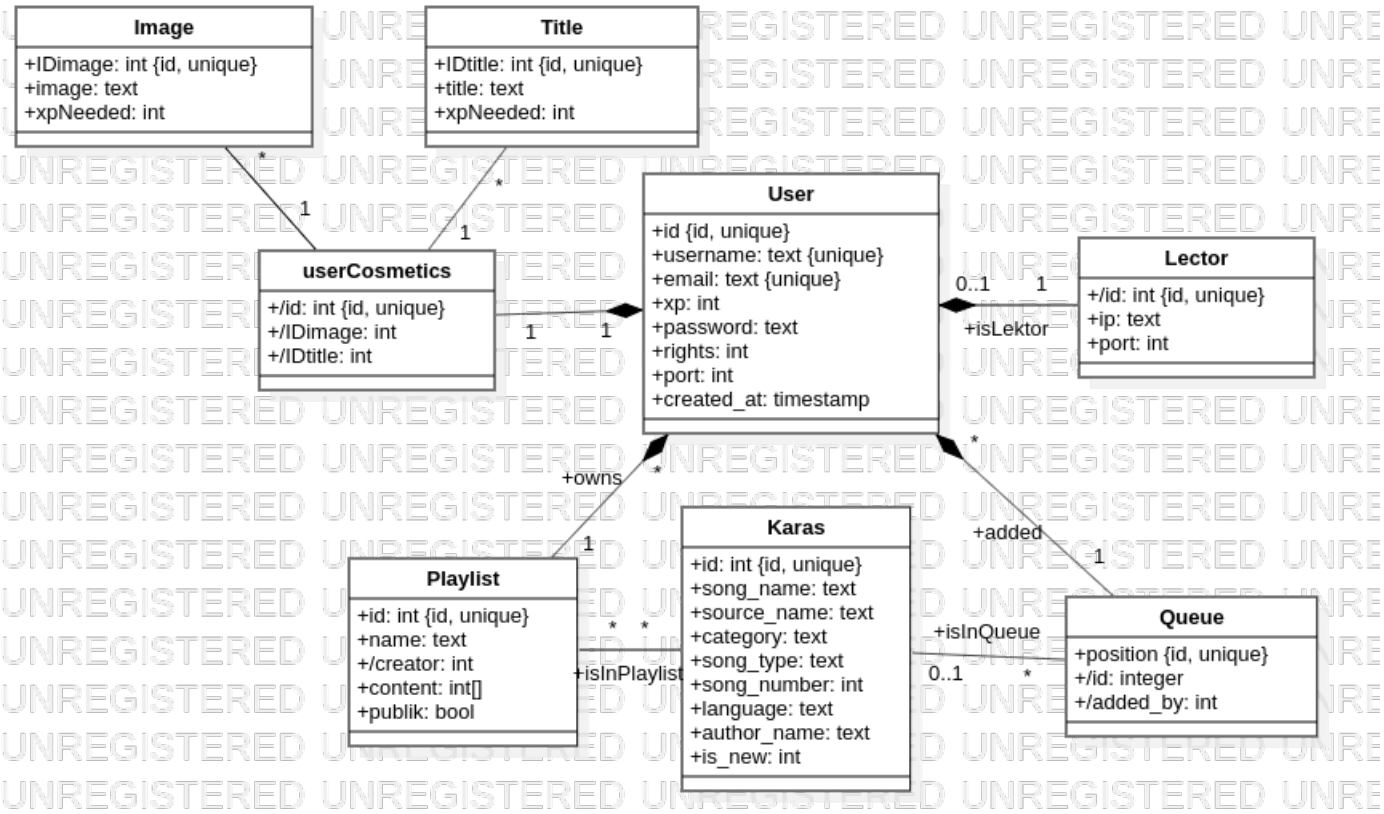

Figure 1 – diagramme UML de la base

3 Sécurité

Un minimum de sécurité a été implanté pour éviter des attaques récurrentes comme un ddos ou des injections SQL :

Chaque variable obtenue d'un formulaire via la méthode POST sera d'abord filtré avec la commande `htmlspecialchars()` pour éviter que des caractères spéciaux apparaissent. Ensuite, ces variables seront rentrées en paramètres dans les commandes SQL via la méthode `bindParam` du PDO, et cela en renseignant à chaque fois le paramètre optionnel `$data_type` qui permet d'éviter un mauvais typage.
Lors d'une création, d'une connexion ou de la modification d'un compte, on vérifie bien que les champs remplis sont bien valides (email au bon format, pas d'espace dans le username), ainsi nous n'aurons pas de risque concernant l'utilisateur qui rentre une commande de type `DROP TABLE`. Chaque formulaire aura avant tout une validation javaScript pour ne pas surcharger le serveur par de mauvaises requêtes.

Pour éviter que l'utilisateur ne "spam" les listes de lectures, on utilise le fichier `/Forms/ddosPrevention.php` qui limite à 6 le nombre de requêtes toutes les 30 secondes (seulement pour les non-administrateurs). On utilise également ce fichier au moment du login ou de l'inscription, pour limiter les tentatives de force brute.

4 Utilisateurs

Lorsque qu'un client arrive pour la première fois sur le site, il n'aura pas accès aux différents services du site. Il devra d'abord se créer un compte sur la page `registration.php`. Ici il entrera ses identifiants comme son adresse email, son nom d'utilisateur, et son mot de passe. Il est à noter qu'un utilisateur ne peut pas avoir d'espace dans son nom, ce qui permet d'avoir une sécurité supplémentaire contre les injections SQL.

Après vérification javascript d'un formulaire, on récupère et envoie ces informations à la page `/Forms/addUser.php` qui va les insérer dans la table `user`, le mot de passe sera hashé pour éviter toute tentative de vol de mot de passe. Tout transfert d'information se fera via la méthode `POST`, afin de transférer les données de manière sécurisée.

Ces informations peuvent être ultérieurement modifiées sur la page `/modifyUser.php`, qui enverra les modifications à `/Forms/modifyUserAccount.php`.

Une fois qu'un utilisateur s'est créé un compte, il peut se connecter via login.php. Si il le fait, on démarre une session et on initialise les attributs de la variable de session, c'est avec ces attributs qu'on déterminera si la session est ouverte, qu'on vérifiera ses droits et qu'on affichera les cosmétiques de l'utilisateur.

4.1 Personnaliser son compte

Aussi, après ajout d'un utilisateur dans `user`, on insère une nouvelle ligne dans la table `UserCosmetics`, cette table permet de savoir quel sont les cosmétiques choisis par l'utilisateur. Chaque utilisateur aura accès à une liste d'images de profil et de titres qu'il pourra sélectionner pour customiser son profil. Ces cosmétiques sont contenus dans les tables `Image` et `Titles`.

Ces images ne sont pas toutes disponibles dès le départ. En effet chaque utilisateur possède des points d'expérience, initialisés à 0 à la création du compte, qui peuvent être gagné lorsqu'il rajoute un karaoke à la queue. Les images se débloquent au fur et à mesure que l'utilisateur passe du temps sur le site. Sur `ChangePP.php` seules les images et titres dont `xpNeeded<$_SESSION['xp']` sont affichés.

Pour pouvoir modifier son compte, l'utilisateur doit se rendre sur `changePP.php`, pour modifier son image de profil et/ou son titre. Un formulaire envoie les informations à `modifyUserCosmetics.php`. L'utilisateur a le choix entre les différentes images contenues dans la table `Images` et les différents titres dans `Titles`.

4.2 Les playlists

Un utilisateur peut créer ses propres playlists de karas disponibles dans la table `karas`. D'ici il pourra ajouter cette playlist dans la queue du lecteur (à condition d'être administrateur), ou simplement retrouver les karas qu'il préfère afin de les ajouter. Toutes ces informations sont contenues dans la table `playlist`, et les utilisateurs peuvent regarder les playlists des autres utilisateurs si elles sont paramétrées comme étant publique. Une playlist ne peut cependant être éditée que par son créateur.

L'implémentation SQL du contenu des playlists est un tableau d'entiers de type `INT[]`. Cela a posé quelques difficultés d'implémentations des requêtes, notamment dans la fonction `\PlaylistRepository->isInPlaylist`, car cela complexifie l'utilisation de la fonction `bindParam`, dans des requêtes comme `SELECT '{:idKara}' && ....`

4.3 Droits des utilisateurs

Lorsqu'un utilisateur s'inscrit dans la base de donnée, on lui attribut un droit 0. Les différents droits sont définis par des entiers :

— 0 pour un utilisateur lambda
— 1 pour un admistrateur
— 2 pour un super-administrateur

Un administrateur a la possibilité de révoquer ou d'augmenter les droits d'un utilisateur ayant des droits inférieurs ou égaux à lui-même. Ceci se fait sur la page `admin.php`, page seulement accessible par les administrateurs. C'est en vérifiant les attributs de la session actuelle `$_SESSION` qu'on détermine si l'utilisateur est bien un admin. Il n'a pas de restriction concernant l'ajout de

karas à la queue et peut la modifier à sa guise. Il peut enfin rajouter des playlists entières à la queue.

5 Queue et Lektor

5.1 Queue

La queue, qui est une "playlist courante", est une table de karaokes. Ses deux principaux attributs sont la position et l'id du karaoke. Elle représente donc une liste ordonnée de karaokes qui vont être lus.

Elle possède aussi un attribut `added_by`, qui permet de savoir quel utilisateur a ajouté chaque karaoke de la queue.

La queue est donc unique et n'importe quel utilisateur peut ajouter des karaokes à celle-ci. L'ajout se fait avec une requête `POST` à la page `/Forms/addKara.php`, en fournissant l'id du karaoke que l'on veut rajouter. Cette page va effectuer la requête, en s'assurant tout d'abord que l'utilisateur est connecté, et, si il n'est pas un administrateur, qu'il n'ajoute pas trop de karaokes en un temps restreint. En pratique, cette page est seulement appelée par les fonctions JavaScript `addKara` et `deleteKara`, qui permettent d'ajouter et de retirer des karaokes tout en restant sur la page `/index.php`. Cela nécessite donc de pouvoir rafraîchir la queue depuis `/index.php`, ce qui est fait grâce à la fonction JavaScript `loadQueue`, qui va faire une requête `GET` à la page `/Forms/getQueue.php` à l'aide de `XMLHttpRequest`, et ensuite actualiser le contenu de l'élement `div` de `index.php`. Seul les administrateurs peuvent supprimer des karaokes de la queue à travers une requête `POST` vérifiant les droits nécessaires à travers la variable `$_SESSION['rights']`

Afin de pouvoir trouver facilement les karaokes voulus dans la queue, la liste complète des karaokes est chargée sur la page html et affichée sous forme de liste de formulaires. Une barre de recherche permet de rentrer des mot-clés, ce qui va exécuter une fonction javascript qui va cacher les karaokes qui ne correspondent pas. Cela permet aux utilisateurs de faire des recherches rapides et de ne pas surcharger le serveur de requêtes (la table `kara` n'ayant pas vocation à changer durant une session).

5.2 Lektor

Lektor est un logiciel de lecture de karaoke. Il fonctionne avec une architecture client-serveur permettant de faire tourner un daemon (lektord) sur son ordinateur, auquel on peut envoyer des commandes à l'aide d'un client (lkt) ou bien tout simplement à l'aide d'une socket ou de netcat. Lektord va ensuite, en fonction des commandes qu'il reçoit, lire les karaokes voulus sur le système où il tourne. Il respecte le protocole MPC, bien qu'étant encore en développement (par des élèves de l'ENSIIE).
Pour pouvoir faire fonctionner lektor sur son pc, une explication se trouve à la

racine du projet dans le readMe.

C'est ce logiciel qu'utilise notre site pour lire des karaokes : chaque utilisateur peut, si il le souhaite et qu'il possède sur son ordinateur personnel la base de données de karaokes ainsi que lektord, s'ajouter sur le site en tant que lecteur. Il faut pour celà avoir l'adresse IP du client, qui est récupérée à l'aide de `$_SERVER['HTTP_CLIENT_IP']`, `$_SERVER['HTTP_X_FORWARDED_FOR']` ou bien `$_SERVER['REMOTE_ADDR']`. L'utilisateur doit aussi signaler le port sur lequel son lektord écoute (si non renseigné, la valeur par défaut est le port 6600, qui est la valeur par défaut de lektord). Ces informations sont stockés dans la base de données. Une information plus "volatile" est stockée dans la variable `$_USER['is_lector']`, qui permet de savoir si un utilisateur est un lecteur sans faire appel à la base de données.

Ensuite, à chaque changement impactant la queue (ajout ou retrait de karaoke), le serveur va récuperer la liste des lecteurs dans la base de donnée, puis envoyer les commandes correspondantes aux lecteurs à l'aide de sockets php. L'implémentation actuelle du site envoie juste les sockets une par une aux différents lektor, avec un timeout de 4 secondes afin d'éviter qu'un lecteur injoignable ne bloque totalement le site. L'implémentation actuelle n'est donc pas vraiment optimale car l'information pourrait prendre un certain temps à atteindre les derniers lecteurs si les lecteurs précedents prennent du temps à accepter la connection. Pendant le développement, une version avec des forks php générés à l'aide de `pcntl_fork()` a été développé. Cependant, les processus fils ne s'arrêtait pas quand il le fallait, ce qui posait des problèmes de processus ne s'arrêtant jamais, et ce même en ayant un `exit` dans chaque processus enfants et en implémentant une boucle de `pcntl_waitpid` dans le processus parent afin d'attendre que tout les processus enfants soient terminés pour quitter. Cette solution est donc momentanément abandonnée.

6 Conclusion

Notre site Bakaraoké a donc rempli les objectifs fixés par le projet, que ce soit la gestion d'utilisateurs ou bien l'ajout de personnalisations, l'utilisation de scripts pour protéger le site ou encore la gestion de tables permettant de gérer les karaokés de la base de donnée et la création de playlists.

Nous pensons que l'amélioration du site se fera à partir de maintenant sur l'approfondissement de la personnalisation des comptes et des playlists, en permettant de créer des playlists collaboratives par la modification de la table pour rajouter un champ collaborateur, ou l'ajout d'une fonctionnalité de lecture directe sur le site en ajoutant un lecteur de vidéos sur le site.