

Rapport technique

Projet Soundhub

Auteurs

Leroy, Maxime – 111 244 596

Longle, Henri – 111 244 597

Équipe

13

Réalisé dans le cadre du cours

GLO-2005 – Modèles et langages des bases de données pour ingénieurs

Rapport présenté à

L'Enseignant, Richard Khoury

Remis le

15 avril 2019

Enonciation du problème et de ses exigences (2 points)

Aujourd'hui, YouTube est la plateforme dominante sur le marché du partage de contenus audiovisuel, et malgré sa croissance incessante, elle s'éloigne de plus en plus de sa communauté en axant davantage ses plans marketing sur des méthodes qui ont pour but de faire fructifier économiquement leur entreprise.

La communauté est donc mise de côté et c'est dans ce cadre qu'intervient notre application Soundhub, qui se veut être une plateforme novatrice en utilisant les dernières technologies du web afin de combler les nouveaux besoins des utilisateurs de YouTube : un endroit où l'information mise en avant est de qualité.

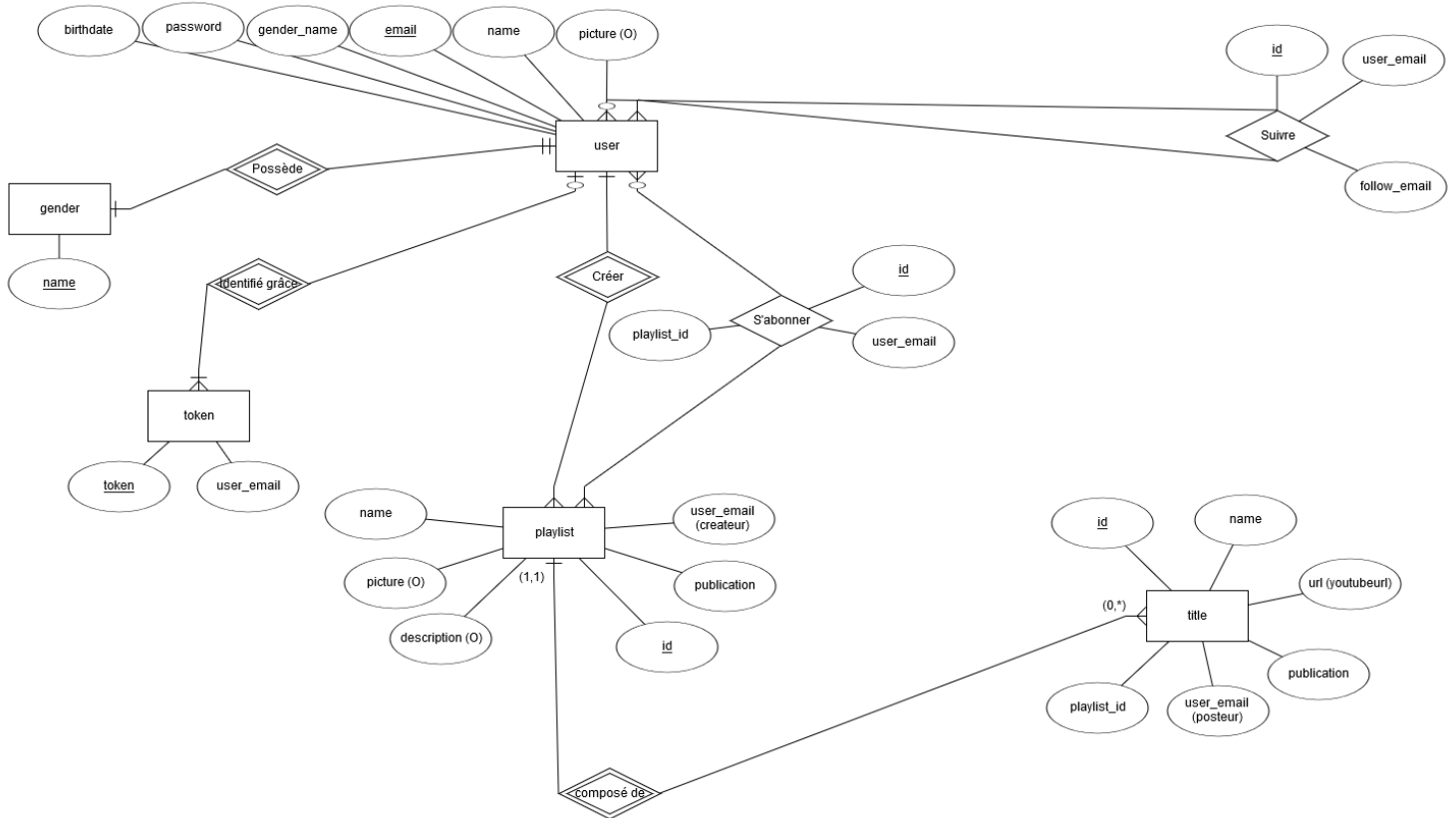
Soundhub est donc une application de partage de contenu musical de qualité, destiné à un cercle privé d'utilisateurs : il est impossible de consulter le contenu de la plateforme sans y être membre.

Le projet répond aux exigences suivantes :

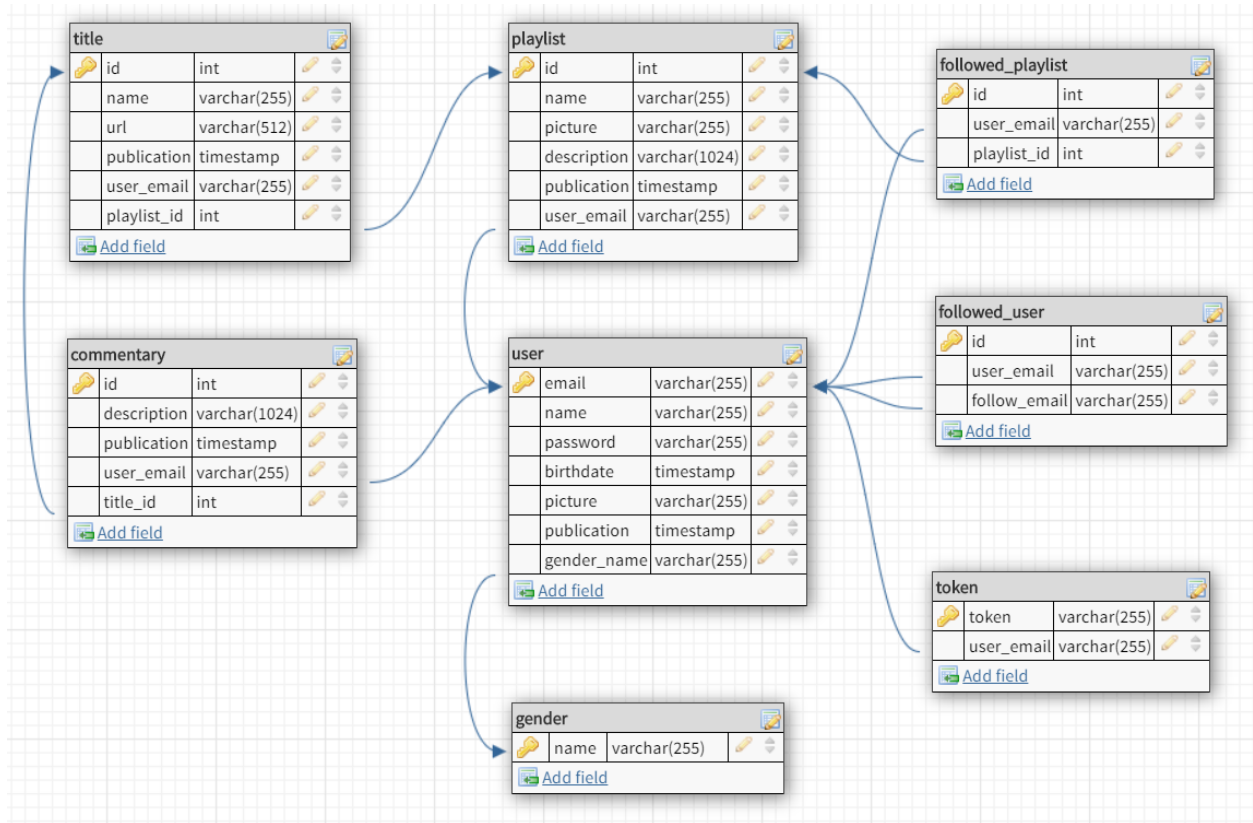
- Créer un compte
- Se connecter
- Se déconnecter
- Afficher une liste de playlists
- Afficher les informations d'une playlist
- Afficher les titres associés à une playlist
- Afficher les informations d'un titre
- Afficher les commentaires associés à un titre
- Afficher les utilisateurs suivis d'un utilisateur
- Afficher ses utilisateurs suivis
- Afficher les playlists suivis d'un utilisateur
- Afficher ses playlists suivis
- Ajouter une playlist
- Ajouter un titre à une playlist
- Ajouter un commentaire à un titre
- Lire un titre via YouTube
- Editer une playlist
- Editer un titre

- Editer un profil
- Supprimer une playlist
- Supprimer un titre
- Supprimer un commentaire
- S'abonner à une playlist
- S'abonner à un utilisateur
- Se désabonner d'une playlist
- Se désabonner d'un utilisateur



Modèle entité-relation du système (3 points)



Modèle relationnel du système (3 points)



Légende :

-  : clé primaire
-  : clé étrangère

Implémentation et fonctionnalités du niveau serveur de BD (4 points)

L'application Soundhub est centrée autour de l'utilisateur ; celui-ci dispose de nombreuses possibilités sur le site, il fallait donc implémenter une entité permettant d'être facilement reliée aux différentes relations mises en place. C'est pourquoi les relations et les actions de l'utilisateur lui sont affectées au moyen de sa clef primaire : son courriel. Le courriel étant unique (chaque utilisateur se connecte sur le site via une combinaison courriel/mot de passe), il a été désigné en tant que clé primaire de la table « user » pour éviter de dupliquer des données via une clé primaire classique (un nombre entier en auto incrémentation).

Le principe de Soundhub est de permettre aux utilisateurs de créer des playlists, c'est pourquoi il a fallu créer une table « playlist », l'utilisateur y est identifié grâce à son courriel via la clé étrangère « user_email ». Chaque utilisateur peut participer à l'élaboration d'une playlist, en y ajoutant des titres.

Comme dit précédemment, les relations liées à l'utilisateur (« followed_playlist », « followed_user » ...) utilisent l'adresse courriel de l'utilisateur en tant que référence à une clé étrangère (« user_email ») afin de lier la relation à l'utilisateur. Dans le cas des playlists suivies, la table comporte une clé étrangère faisant référence à l'id de playlist (« playlist_id »), qui est la clé primaire de la table « playlist ». Pour les « followed_user », l'entrée « follow_email » désigne le courriel de l'utilisateur suivi par le « user_email ». Ces relations n'ont pas besoin d'être plus compliquées, elles permettent de retrouver facilement les entités en relation et n'ont besoin d'aucune donnée supplémentaire.

De ce fait, la table « title » doit posséder 2 clés étrangères : « playlist_id » correspondant à la clef primaire de la playlist qui permettra de lier le titre à la playlist et « user_email » référençant le courriel de l'utilisateur qui ajoute ce titre.

Les utilisateurs peuvent réagir aux différents titres d'une playlist en y postant des commentaires, il faut donc pouvoir, pour chaque commentaire, identifier l'utilisateur et le titre qui sont liés. C'est pourquoi la table « commentary » est composée de 2 clés étrangères elle aussi : « title_id » référençant le titre, et « user_email » qui désigne l'utilisateur l'ayant posté.

Des gâchettes SQL ont été mises en place lors de la création et la modification des entités « user », « playlist » et « title », elles vérifient certaines données (assez de caractères dans les champs par exemple), et renvoient une erreur appropriée. Afin d'éviter la redondance de code (une gâchette à la création et une gâchette à la modification), des procédures et des fonctions ont été mises en place. La majorité des procédures vérifient les champs d'une entité donnée, mais il y a aussi une procédure spécifique qui vérifie grâce à une expression régulière (REGEX) la validité d'une URL. Cette vérification d'URL est utilisée lors de la création et la modification des entités « user », « playlist » et « title » ; « user » et « playlist » possèdent toutes les deux un attribut « picture » qui est une URL vers une image, le « title » quant à lui possède une URL vers une vidéo YouTube, l'URL est vérifiée par une procédure spécifique puis transformée en lien pour « vidéo intégrée ».

Indexation des données, normalisation des relations, et optimisation des requêtes (4 points)

Une attention particulière a été portée à la mise en place de la base de données, notamment sur la duplication des données ou encore l'indexation de celles-ci. Aucune donnée n'est redondante dans les différentes tables, chaque entité et chaque relation à des attributs qui lui sont propres, les dépendances sont découpées en plusieurs relations. De plus les entités comme les relations ont toutes une clé primaire qui permet au système de gestion de la base de données (ici MySQL) de retrouver rapidement les données car les opérations de recherche et de tri sont accélérées.

Comme expliqué précédemment un exemple pertinent est l'utilisation de l'adresse courriel de l'utilisateur en tant que clé primaire, cette solution évite la duplication d'index et simplifie grandement la recherche d'utilisateur spécifique : on ne doit plus rechercher l'entité qui possède

l'attribut « email » mais on fait une recherche par clé primaire ce qui augmente grandement la vitesse. Ce choix est d'autant plus justifié que dans la logique d'interface, comme dans la logique d'affaire, il est simple d'avoir accès à l'adresse courriel de l'utilisateur.

Les relations et les entités ont aussi été optimisées en ajoutant des références, grâce au système de clés étrangères, qui font référence aux clés primaires des entités dépendantes. Tout comme les clés primaires, les clés externes augmentent grandement la vitesse d'exécution des requêtes, mais elles permettent aussi de mieux gérer les dépendances en y ajoutant des contraintes : toutes les tables faisant référence à des clés étrangères ont été protégées grâce aux contraintes ON DELETE et ON UPDATE, ainsi si une entité est modifiée ou supprimée, l'ensemble de ses relations s'adaptera en conséquence.

Implémentation et fonctionnalités de la logique d'affaire (3 points)

Les sites web les plus récents utilisent tous une API afin de centraliser les appels à la base de données et de ne pas dupliquer ceux-ci dans leurs différentes applications, c'est pourquoi une API pour Soundhub a été mise en place et implémentée grâce à Python Flask. Une API est un regroupement de classes, de méthodes et de fonctions qui sert de façade par laquelle une application offre des services. Dans le cas d'une API Web, c'est à travers des requêtes HTTP sur des URL précises que l'on accède à ces fonctionnalités, par exemple, dans le cas de l'API Soundhub, il est possible d'appeler la route « /users » en « GET » pour récupérer une liste d'utilisateurs.

L'API Soundhub se veut le plus RESTful possible, c'est-à-dire qu'elle respecte l'architecture logicielle REST qui définit un ensemble de contraintes qui permettent de manipuler les données sans aucun état. L'avantage d'une application RESTful réside dans le fait que les appels qui lui sont faits sont uniformes et prédéfinis.

Il existe de nombreux formats de sérialisation des données, tous plus ou moins complexes, dans le cas de Soundhub, l'API utilise le format JSON pour communiquer. Le JavaScript Notation

Object (JSON), est un format de données textuelles basées sur les objets du langage JavaScript, cette notation a l'avantage d'être rapidement interprétées (par les humains comme les machines), d'être simple à construire par le client (contrairement à d'autres formats comme le XML).

Python propose de nombreuses bibliothèques très facilement intégrables ; notamment SQLAlchemy, un ORM (Object-Relationnal Mapping) qui permet d'interfacer des classes pour pouvoir les utiliser en tant que modèle de base de données, ce qui simule une base de données « orientée objet ». Les ORM ont deux avantages majeurs : la manipulation des données, et la construction des requêtes de façon procédurale et non à la volée. Les classes étant reliées aux entités, il est très simple de créer/modifier/supprimer des tuples, de plus cela permet d'interagir directement avec la classe et de ce fait, de ne pas construire de requête à la volée.

Implémentation et fonctionnalités de l'interface utilisateur (3 points)

Le site web de Soundhub utilise 3 technologies principales :

- Le langage de balisage HTML qui permet de structurer les pages web de façon sémantique et logique.
- Le langage CSS qui permet de décrire les différents éléments HTML en y appliquant des styles et des classes. Soundhub utilise le framework CSS Bootstrap qui regroupe différents styles permettant de simplifier le design, le responsive ou encore le placement des éléments HTML. Bootstrap est l'un des frameworks les plus populaires, il a l'avantage d'être extrêmement intuitif d'utilisation et d'offrir de nombreuses possibilités en plus de fournir d'innombrables styles prédéfinis ainsi qu'un système de grille pour placer les éléments.
- Le langage de programmation JavaScript est utilisé de façon « native » (sans aucun framework). JavaScript est à la base un langage qui permet d'écrire des scripts permettant de réagir aux actions d'un utilisateur ou au comportement d'un navigateur, le langage a maintenant évolué et possède un champ étendu de fonctionnalité et de paradigme. Dans le cas de Soundhub, le JavaScript est à la fois utilisé pour fournir à l'utilisateur une

expérience dynamique et interactive (chargement de page dynamique, chargement des données à la volée), mais aussi pour communiquer avec l'API via un SDK.

Bien qu'aucun framework ne soit utilisé, certaines librairies JavaScript sont utilisées, notamment la librairie de chargement polyfill de JavaScript « es6-module-loader », qui permet d'importer des modules ES6 (ensemble de normes concernant la programmation de script JavaScript) dans les différentes vues HTML. De plus afin de pouvoir charger certains éléments HTML de façon dynamique et sans avoir à répéter de code, Soundhub utilise la fonction « w3.includeHTML() » qui permet d'importer des fragments d'HTML de façon native.

Toutes les pages HTML fonctionnent de la même façon, elles sont chargées de façon statique (via l'URL), une fois la page prête (document.ready()), des scripts s'occupent de charger les ressources nécessaires et de générer à la volée des éléments HTML afin d'afficher les données récupérées. Les différents scripts JavaScript des pages utilisent le SDK Soundhub mis en place qui centralise tous les appels à l'API (par exemple récupérer un utilisateur qui a un courriel spécifique, récupérer les dernières playlists etc...), le SDK (Software Development Kit) est totalement indépendant de la logique d'interface et peut être utilisé par d'autres applications qui souhaitent intégrer Soundhub. Certaines pages (par exemple le Profil) utilisent des contrôleurs qui s'occupent d'appeler le SDK et générer les éléments HTML.

Sécurité du système (2 points)

Afin d'éviter que certains utilisateurs n'accèdent à des ressources auquel ils ne sont pas autorisés, Soundhub authentifie ses utilisateurs à chaque appel à la base de données via le système d'authentification OAuth. Quand un utilisateur se connecte, il envoie son email et son mot de passe au serveur qui vérifie ses données et lui donne en échange un token d'authentification permettant d'accéder aux ressources du site. Ce token est unique et lié à l'utilisateur. Aujourd'hui OAuth est le protocole d'authentification le plus utilisé par les services web, de nombreuses entreprises telles que Facebook, Twitter ou encore Google utilisent ce système.

Le mot de passe est haché grâce au paquet python : Bcrypt, qui est une librairie de cryptage basé sur l'algorithme de chiffrement par blocs « Blowfish ».

La librairie utilise un système de salage de mot de passe. En effet, pour éviter que deux données identiques possèdent le même hachage, des données supplémentaires aléatoires sont volontairement rajoutées dans le hachage. De plus, afin de s'adapter au fil des années aux nouvelles puissances de calcul des machines qui pourraient attaquer par force brute les informations, Bcrypt propose d'effectuer plusieurs passes sur le hachage. Dans Soundhub, Bcrypt fait 10 passes et le sel est généré par la librairie qui ne le sauvegarde jamais.

La base de données est protégée des injections car toutes les recherches de données se font sur les clés primaires via l'ORM « SQLAlchemy » : aucune requête n'est donc construite à la volée.

Pour éviter l'insertion de mauvaises données dans la base de données, plusieurs systèmes de sécurité ont été mis en place. Dans un premier temps il y a une vérification des données par le JavaScript qui vérifie les données du formulaire entré. Par la suite l'api Python vérifie que les données ne sont pas vides et répondent bien aux différents critères. Finalement, c'est la base de données SQL qui, à travers différents triggers (avant insertion et modification), valide les différents champs, en plus des contraintes qui sont présentes sur les tables (clefs externes, élément unique, élément non null).

Organisation, gestion de l'équipe, et division des tâches (1 points)

L'organisation a été simple puisque tout le travail réalisé a été fait en la présence de la totalité des membres sur toute la durée du projet : du premier au dernier commit inclus.

La gestion de l'équipe s'est faite entièrement via un système de branches et de pull requests sur un GitHub public créé avec le compte GitHub de Maxime.

Tout le monde a travaillé sur l'ensemble des parties, cependant certains aspects ont été réalisés principalement par une personne. Vous trouverez ci-dessous une liste des participants et les parties qui ont été principalement faites par chacun des membres :

- Maxime :
 - Configuration et installation de l'API (Python Flask)
 - Gestion des modèles et des entités dans l'API
 - Gestion des routes de l'API

- Henri :
 - Gestion et création du serveur MySQL
 - Gestion et création du « docker-compose » ainsi que des images Docker
 - Gestion et population des entités dans la base de données

- Maxime et Henri :
 - Logique frontend (HTML, CSS, JavaScript)
 - Configuration et gestion du SDK (JavaScript)
 - Logique backend SQL (trigger, contraintes, indexisation)
 - Rédaction du rapport

Revue du code (5 points)

Lien du code source :

<https://github.com/maximeleroylaval/bdd/tree/master/projet>

Instructions :

<https://github.com/maximeleroylaval/bdd/blob/master/README.md>