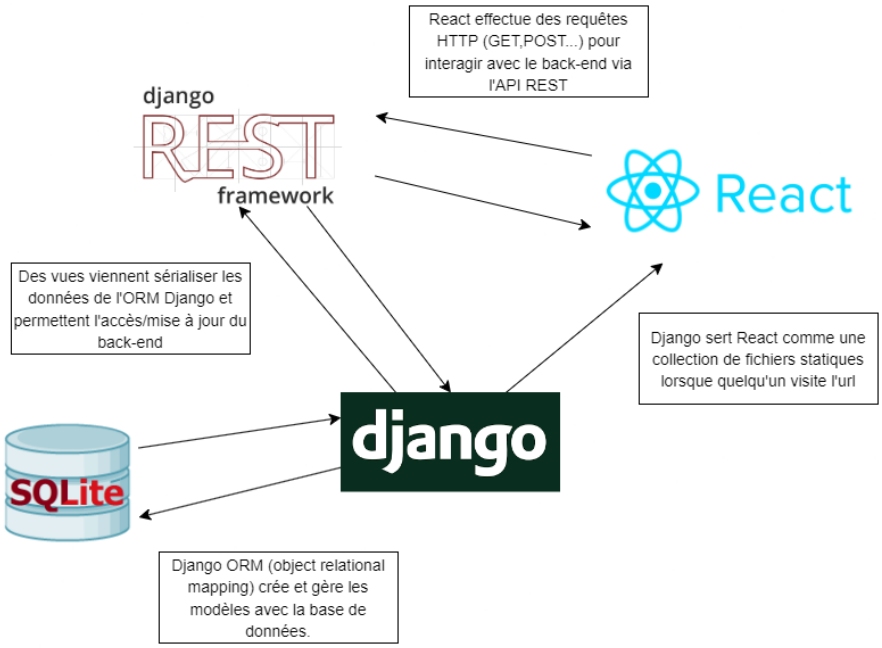
Documentation de La Boite à Son

Technologies utilisées et guide d'installation

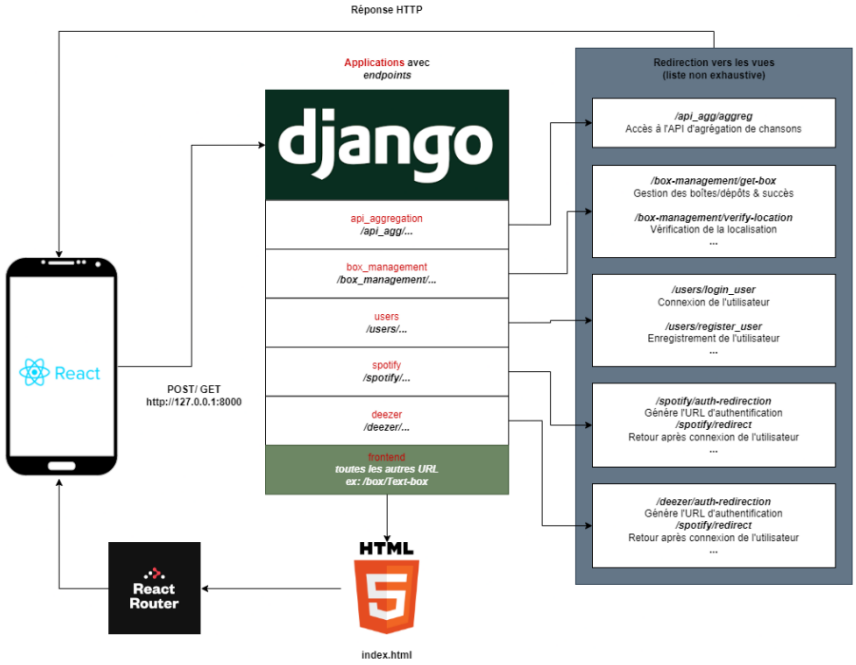
Les informations relatives aux technologies utilisées et le guide d'installation se trouvent dans le fichier 'Technologies & installation.pdf'

Architecture du projet

Architecture globale de l'application



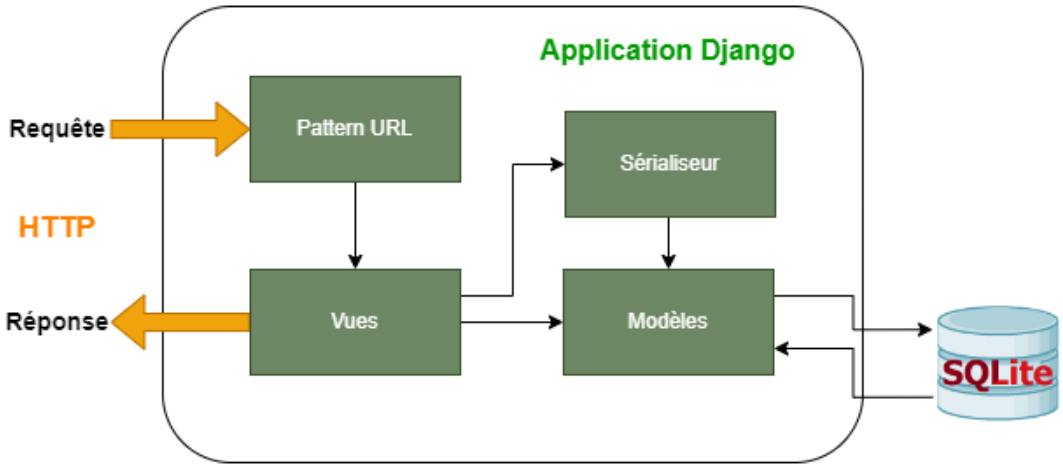
Mécanisme de distribution d'URLs



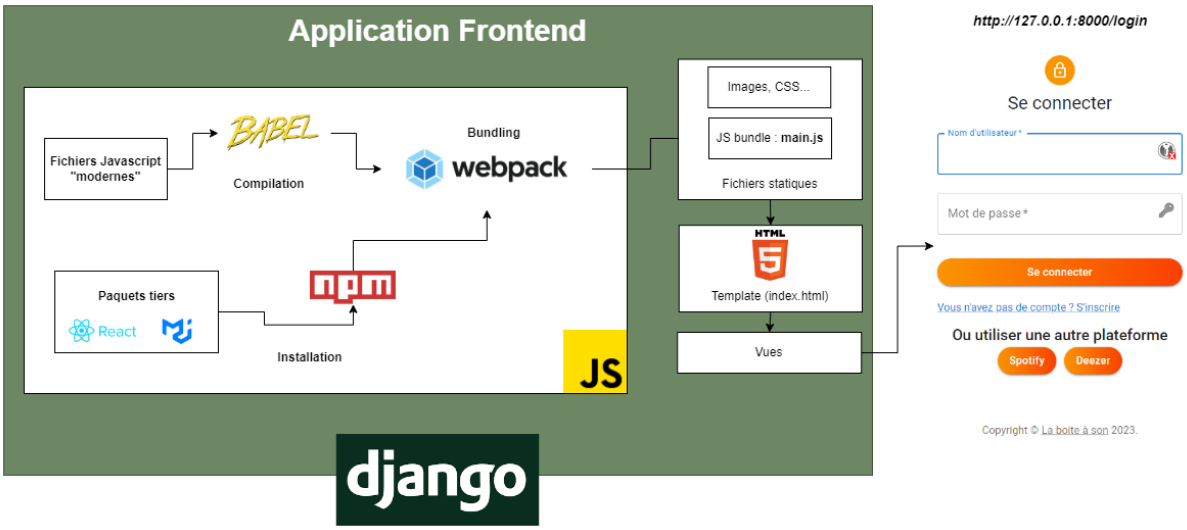
La figure précédente représente en rouge les différentes applications backend.

La figure suivante présente le fonctionnement de chaque application.

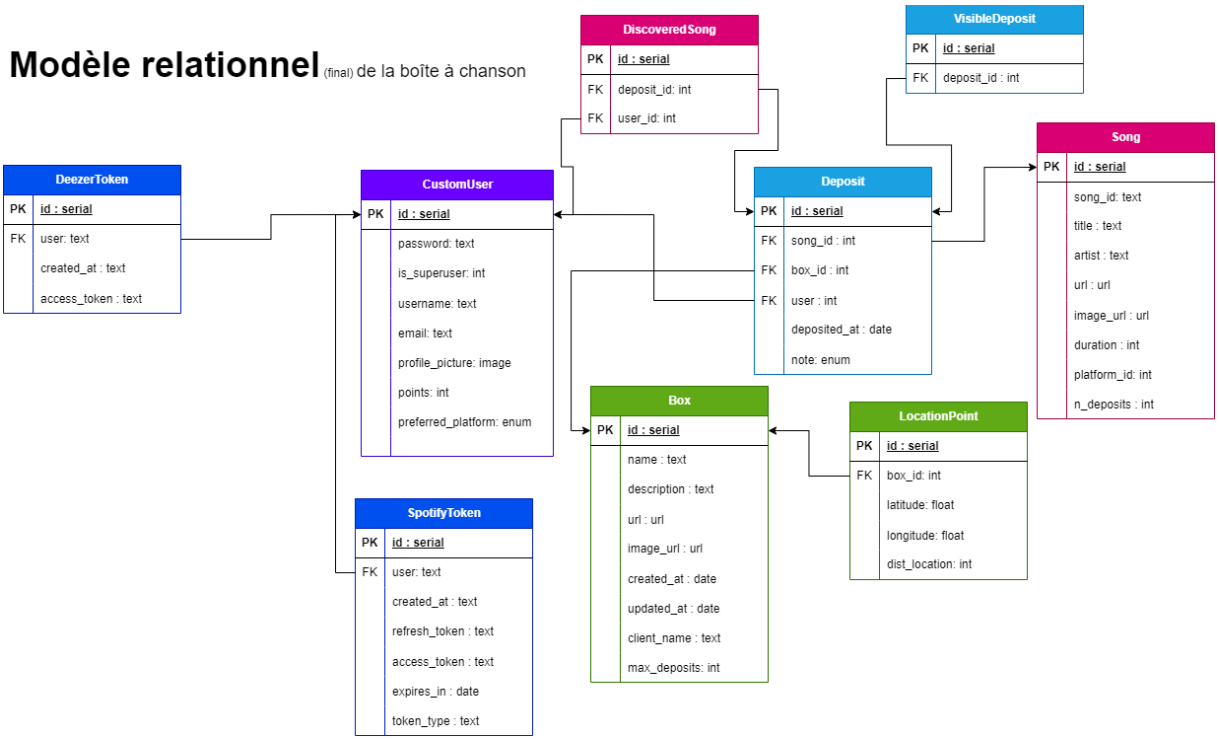
Fonctionnement des applications backend



Lien entre le frontend et le backend



Modèle relationnel de la base de données



Utilisateurs

La classe `CustomUser`

Le modèle `CustomUser` étend le modèle `AbstractUser` fourni par Django pour fournir une fonctionnalité de gestion personnalisée des utilisateurs dans votre application.

Champs du modèle

- username : nom d'utilisateur
- password : mot de passe
- email : adresse électronique
- profile_picture : champ ImageField permettant à l'utilisateur de télécharger et de stocker une image de profil, les images sont enregistrées dans un répertoire spécifique nommé « media »
- points : champ IntegerField qui enregistre le nombre de points de l'utilisateur
- preferred_platform : champ CharField qui permet à l'utilisateur de sélectionner sa plateforme préférée parmi les choix disponibles (Spotify ou Deezer)

Méthodes du modèle

- `save(self, *args, **kwargs)` : Cette méthode est une substitution de la méthode save() héritée de la classe parent AbstractUser. Elle est utilisée pour supprimer l'ancienne photo de profil lorsque celle-ci est modifiée. Lorsqu'un utilisateur existant est enregistré à nouveau, cette méthode compare la photo de profil actuelle avec celle enregistrée précédemment. Si elles sont différentes, l'ancienne photo de profil est supprimée de la base de données.
- `profile_picture_path(instance, filename)` : Cette méthode génère le chemin de destination pour enregistrer l'image de profil de l'utilisateur. Elle utilise une fonction generate_unique_filename provenant du module utils pour garantir l'unicité du nom de fichier
- `delete_profile_picture(sender, instance, **kwargs)` : Cette fonction est un récepteur d'un signal pre_delete qui est déclenché avant la suppression d'un utilisateur. Lorsque cet événement se produit, la photo de profil de l'utilisateur est supprimée de la base de données.
- `delete_old_profile_picture(sender, instance, **kwargs)` : Cette fonction est un récepteur d'un signal pre_save qui est déclenché avant l'enregistrement d'un utilisateur. Elle vérifie si l'utilisateur existe déjà dans la base de données, récupère l'utilisateur existant et compare la photo de profil actuelle avec celle enregistrée précédemment. Si elles sont différentes, l'ancienne photo de profil est supprimée de la base de données.

Gestion des utilisateurs par des appels API

Les méthodes s'appliquent à l'utilisateur connecté.

Connexion d'un utilisateur :

- URL : */users/login_user*
- Méthode : POST
- Payload attendu :

```
{
   "username": "joli_nom_utilisateur",
   "password": "mon_beau_mot_de_passe"
}
```

- Retour : *True* si l'utilisateur a pu se connecter, sinon *False*

Déconnexion d'un utilisateur :

- URL : */users/logout_user*
- Méthode : GET
- Retour : *True* si l'utilisateur a été déconnecté, *False* s'il était déjà anonyme

Enregistrement (register) d'un utilisateur :

- URL : */users/register_user*
- Méthode : POST
- Payload attendu :

```
{
   "username": "nouvel_utilisateur",
   "email": "email@mail.fr",
   "profile_picture": passer la profile picture ici (optionnel),
   "password1": "le_beau_mdp",
   "password2": "le_beau_mdp"
}
```

- Retour : *True* si l'utilisateur a pu être enregistré, messages d'erreurs si nécessaire (ex : MDP ne correspondant pas, username déjà existant …)

Vérification de l'état de connexion d'un utilisateur :

- URL : */users/check-authentication*
- Méthode : GET
- Retour : JSON vide {} si non connecté, payload si connecté :

```
{
   "username": "blabla",
   "email": "blabla@blabla.fr",
```

```
    "profile_picture_url": "/media/url.jpg",
    "preferred_platform": "platforme_pref",
    "points": NB_POINTS,
    "is_social_auth": True si utilisateur enregistré via Spotify ou Deezer sinon False
}
```

Changer de mot de passe :

- URL : */users/change-password*
- Méthode : POST
- Payload attendu :

```
{
    "old_password": "mon_ancien_mdp",
    "new_password1": "mon_nouveau_mdp",
    "new_password2": "mon_nouveau_mdp"
}
```

- Retour : *True* si le mot de passe a pu être changé, messages d'erreurs si nécessaire

Changer de photo de profil :

- URL : */users/change-profile-pic*
- Méthode : POST
- Payload attendu : FICHIER => nouvelle photo de profil
- Retour : *True* si la photo de profil a pu être changée, messages d'erreurs si nécessaire

Ajout de points :

- URL : */users/add-points*
- Méthode : POST
- Payload attendu :

```
{
    "points": NB_POINTS_A_AJOUTER
}
```

- Retour : *True* si les points ont pu être ajoutés, messages d'erreurs si nécessaire

Remarque : pour supprimer des points, on ajoute un nombre négatif de points.

Récupérer le nombre de points de l'utilisateur :

- URL : */users/get-points*
- Méthode : GET
- Retour : erreur si déconnecté, payload avec nombre de points si connecté :

```
{
   "points": NB_POINTS_UTILISATEUR
}
```

Gestion des boîtes

La classe `Box`

Le modèle `Box` contient les informations des boîtes virtuelles.

Champs du modèle

- name : nom de la boîte
- description : description de la boîte
- url : url d'accès à la boîte
- image_url : url de l'image de la boîte
- created_at : date de création
- updated_at : date de la dernière mise à jour de la boîte
- client_name : nom du client
- max_deposits : nombre maximum de dépôts visibles sur la boîte

Méthodes du modèle

- `__str__(self)` : Cette méthode est une substitution de la méthode __str__(self) héritée de la classe parent Model. Elle permet d'afficher le nom de la boîte dans l'interface administrateur de Django.

La classe `LocationPoint`

Le modèle `LocationPoint` contient les informations de tous les points de localisation de toutes les boîtes. Il est indispensable que chaque boîte soit associée au minimum à un point de localisation pour que la localisation puisse fonctionner.

Champs du modèle

- box_id : identifiant de la boîte
- latitude : latitude du point de localisation
- longitude : longitude du point de localisation
- dist_location: distance maximale entre l'utilisateur et le point de localisation

Méthodes du modèle

- `__str__(self)` : Cette méthode est une substitution de la méthode __str__(self) héritée de la classe parent Model. Elle permet d'afficher le nom de la boîte correspondante, la latitude et la longitude dans l'interface administrateur de Django.

La classe `Song`

Le modèle `Song` contient les informations des chansons qui ont été déposées au moins une fois.

Champs du modèle

- song_id : identifiant de la chanson
- title : titre
- artist : artiste
- url : url de la chanson (Spotify ou Deezer)
- image_url : url de l'image de l'album
- duration : durée de la chanson
- platform_id : numéro de la plateforme (1 = Spotify, 2 = Deezer) à laquelle correspondent les urls

- n_deposits : nombre de dépôts de la chanson

Méthodes du modèle

- `__str__(self)` : Cette méthode est une substitution de la méthode __str__(self) héritée de la classe parent Model. Elle permet d'afficher le titre et artiste de la chanson dans l'interface administrateur de Django.

La classe `Deposit`

Le modèle `Deposit` contient les informations de tous les dépôts qui ont été faits.

Champs du modèle

- song_id : identifiant de la chanson
- box_id : identifiant de la boîte
- user : utilisateur ayant déposé la chanson
- note : note choisie par l'utilisateur dans la liste NOTE_CHOICES au dépôt de la chanson
- deposited_at : date du dépôt

Méthodes du modèle

- `__str__(self)` : Cette méthode est une substitution de la méthode __str__(self) héritée de la classe parent Model. Elle permet d'afficher l'identifiant de la chanson et de la boîte pour le dépôt dans l'interface administrateur de Django.
- `save(self, *args, **kwargs)` : Cette méthode est une substitution de la méthode save() héritée de la classe parent Model. Elle est utilisée pour rendre modifiable le champ deposited_at.

La classe `VisibleDeposit`

Le modèle `VisibleDeposit` contient les dépôts qui sont affichés dans chaque boîte.

Champs du modèle

- deposit_id : identifiant du dépôt

Méthodes du modèle

- `__str__(self)` : Cette méthode est une substitution de la méthode __str__(self) héritée de la classe parent Model. Elle permet d'afficher l'identifiant du dépôt visible et l'identifiant du dépôt dans l'interface administrateur de Django.

La classe DiscoveredSong

Le modèle `DiscoveredSong` contient les chansons découvertes par chaque utilisateur.

Champs du modèle

- song_id : identifiant de la chanson
- user_id : identifiant de l'utilisateur

Méthodes du modèle

- `__str__(self)` : Cette méthode est une substitution de la méthode __str__(self) héritée de la classe parent Model. Elle permet d'afficher l'identifiant de l'utilisateur et celui du dépôt dans l'interface administrateur de Django.

Actions sur les boîtes

Accès à la boîte

- URL : */box-management/get-box?name=ma_boîte*
- Méthode : GET
- Retour : payload :


```
{
    "last_deposits": [
        {
            "id": 115,
            "note": "",
            "deposited_at": "2023-06-20T13:59:38.661706+02:00",
            "song_id": 100,
            "box_id": 1,
            "user": null
        }
    ],
    "last_deposits_songs": [
        {
            "id": 100,
            "song_id": "2bSk87AVkCIIC3Bcligq1z",
            "title": "Life Goes On",
            "artist": "Lil Baby",
            "url": "https://open.spotify.com/track/2bSk87AVkCIIC3Bcligq1z",
            "image_url":
"https://i.scdn.co/image/ab67616d0000b2736cab41f8c84d6164976400d4",
            "duration": 247,
            "platform_id": 1,
            "n_deposits": 1
        }
    ],
    "deposit_count": 2,
    "box": {
        "id": 1,
        "name": "test",
        "description": "",
        "url": "",
        "image_url": "",
        "created_at": "2023-06-13T10:23:08.686251+02:00",
        "updated_at": "2023-06-13T10:23:08.686251+02:00",
        "client_name": "TSE",
        "max_deposits": 5
    }
}
```


Dépôt de chanson

- URL : */box-management/get-box?name=ma_boîte*
- Méthode : POST
- Payload attendu :

```
{
  "option": "choix_de_chanson_utilisateur",
  "boxName": "nom_boite"
}
```

- Retour : si l'action s'est bien effectuée renvoie :

```
{
```

```
    'new_deposit': new_deposit,
     'achievements': successes
}
```

Récupération de la boîte actuelle (boîte sur laquelle l'utilisateur est)

- URL : /box-management/current-box-management
- Méthode : GET
- Retour : erreur 400 si pas de boîte ouverte ou payload :

```
{
    "current_box_name": "tse"
}
```

Mise à jour de la boîte actuelle

- URL : */box-management/get-box?name=ma_boîte*
- Méthode : POST
- Payload attendu :

```
{
   "current_box_name": "choix_de_chanson_utilisateur",
   "boxName": "nom_boite"
}
```

- Retour : 200 OK si succès de mise à jour, 401 UNAUTHORIZED si le champ est absent du payload, 500 INTERNAL SERVER ERROR si une exception a lieu lors de la mise à jour

Vérification de la localisation

- URL : */box-management/verify-location*
- Méthode : POST
- Payload attendu :

```
{
   "latitude": "valeur_latitude",
   "longitude": "valeur_longitude",
   "box ": "BoxObject"
}
```

- Retour : coordonnées et 200 si les coordonnées sont valides, erreur 404 si pas de points de localisation renseignés pour la boîte, erreur 403 si l'utilisateur n'est pas à la bonne distance de la boîte

Mise à jour des dépôts visibles

- URL : */box-management/update-visible-deposits*
- Méthode : POST
- Payload attendu :

```
{
   "box_name": "nom_boite"
}
```

- Retour : 200 success à la mise à jour des dépôts visibles pour la boîte

Remplacement du dépôt visible

- URL : */box-management/replace-visible-deposits*
- Méthode : POST
- Payload attendu :

```
{
   "box_id": "identifiant_boite",
   "visible_deposit_id": "identifiant_boite",
   "search_deposit_id": "depot_apres_recherche"
}
```

- Retour : 200 success si le dépôt visible a bien été remplacé, erreur 400 si le nouveau dépôt est déjà dans les dépôts visibles

Liste des chansons découvertes par l'utilisateur

- URL : */box-management/discovered-songs*
- Méthode : GET
- Retour : payload :


```
[
    {
        "id": 114,
        "song_id": "2PpruBYCo4H7WOBJ7Q2EwM",
        "title": "Hey Ya!",
        "artist": "Outkast",
        "url": "https://open.spotify.com/track/2PpruBYCo4H7WOBJ7Q2EwM",
        "image_url": "https://i.scdn.co/image/ab67616d0000b2736a6387ab37f64034cdc7b367",
        "duration": 235,
        "platform_id": 1,
        "n_deposits": 1
    },
    {
        "id": 96,
        "song_id": "",
        "title": "Daylight",
        "artist": "David Kushner",
        "url": "https://open.spotify.com/track/1odExI7RdWc4BT515LTAwj",
        "image_url": "https://i.scdn.co/image/ab67616d0000b27395ca6a9b4083a86c149934ae",
        "duration": 212,
        "platform_id": 1,
        "n_deposits": 1
    }
]
```


Ajout de la chanson du dépôt à la liste des chansons découvertes

- URL : */box-management/discovered-songs*
- Méthode : POST
- Payload attendu :

```
{
   "user": "CustomUserOject"
}
```

- Retour : 200 success si l'ajout aux chansons découvertes a bien été effectué, erreur 401 si l'utilisateur n'est pas connecté, erreur 400 si la chanson est déjà liée à un autre dépôt de l'utilisateur

Ajout d'une note au dépôt

- URL : */box-management/add-note*
- Méthode : POST
- Payload attendu :

```
{
   "note": "note_choisie"
}
```

- Retour : 200 success si la note a bien été ajoutée au dépôt, erreur 401 si aucune note n'a été sélectionnée, erreur 400 si le dépôt n'existe pas

Connexion Spotify/Deezer

La Classe `SpotifyToken`

Le modèle `SpotifyToken` contient les informations nécessaires pour communiquer avec l'API Spotify. Il permet aussi la liaison entre le compte utilisateur et le compte Spotify de l'utilisateur.

Champs du modèle

- user : utilisateur qui sera lié au token Spotify
- created_at : date de création du token
- refresh_token : token lié au compte Spotify de l'utilisateur qui permet de rafraichir la connexion à l'API Spotify sans avoir besoin de se reconnecter
- access_token : token lié au compte Spotify de l'utilisateur qui permet de faire des requêtes sur l'API Spotify (notamment obtenir les titres écoutés récemment par l'utilisateur)
- expires_in : date à laquelle l'access_token sera périmé (utile pour savoir quand utiliser le refresh_token)
- token_type : donne le type du token

La Classe `DeezerToken`

Le modèle `DeezerToken` contient les informations nécessaires pour communiquer avec l'API Deezer. Il permet aussi la liaison entre le compte utilisateur et le compte Deezer de l'utilisateur.

Champs du modèle

- user : utilisateur qui sera lié au token Deezer
- created_at : date de création du token
- access_token : token lié au compte Deezer de l'utilisateur qui permet de faire des requêtes sur l'API Deezer (notamment obtenir les titres écoutés récemment par l'utilisateur). Il ne possède pas de date d'expiration donc pas besoin de le rafraichir.

Communication entre l'API Spotify/Deezer et l'application

Les méthodes s'appliquent pour un utilisateur connecté ou non. Certaines méthodes sont communes pour Spotify et Deezer et ne seront donc pas détaillés 2 fois (pour Deezer remplacer spotify par deezer au niveau des url)

Authentification Spotify/Deezer :

- URL : */spotify/auth-redirection*
- Méthode : GET
- Retour : URL de redirection vers la page de connexion Spotify/Deezer

Déconnexion Spotify/Deezer :

- URL : */spotify/disconnect*
- Méthode : GET
- Retour : *True* si la déconnexion s'est bien effectuée

Redirection de l'authentification Spotify/Deezer :

- URL : */spotify/redirect*
- Méthode : GET
- Retour : Page de profil de l'utilisateur s'il s'est bien connecté à Spotify/Deezer

Vérification de l'authentification au compte Spotify/Deezer :

- URL : */spotify/is-authenticated*
- Méthode : GET
- Retour : True si l'utilisateur est bien connecté à son compte Deezer/Spotify et False s'il ne l'est pas.

Obtention des titres récemment écoutés par l'utilisateur sur Spotify/Deezer :

- URL : */spotify/recent-tracks*
- Méthode : GET
- Retour : JSON vide {} si aucun son écouté ou une liste des sons récemment écoutés par l'utilisateur :


```
[
  {
    "id": "2gZUPNdnz5Y45eiGxpHGSc",
    "name": "POWER",
    "artist": "Kanye West",
    "album": "My Beautiful Dark Twisted Fantasy",
    "image_url" : "https://i.scdn.co/image/ab67616d0000b273d9194aa18fa4c9362b47464f",
    "duration" : 292,
    "platform_id" : 1,
    "url": "https://open.spotify.com/track/2gZUPNdnz5Y45eiGxpHGSc"
  },
  {
    "id": "3FjYAj4hg5KiXpVAMjpLKq",
    "name": "This Is the Day",
    "artist": "The The",
```

```
        "album": "Soul Mining",
        "image_url": "https://i.scdn.co/image/ab67616d0000b2731bbff7a308778b39fc8c08ba",
        "duration": 297,
        "platform_id": 1,
        "url": "https://open.spotify.com/track/3FjYAj4hg5KiXpVAMjpLKq"
    }
]
```


Recherche avec l'API Spotify/Deezer :

- URL : */spotify/search*
- Méthode : POST
- Payload attendu :

```
{
   "search-query": "titre de chanson à rechercher"
}
```

- Retour : Liste de chansons renvoyées par l'API Spotify :


```
[
    {
        "id" : "2gZUPNdnz5Y45eiGxpHGSc",
        "name" : "POWER",
        "artist" : "Kanye West",
        "album": "My Beautiful Dark Twisted Fantasy",
        "image_url" : "https://i.scdn.co/image/ab67616d0000b273d9194aa18fa4c9362b47464f",
        "duration" : 292,
        "platform_id" : 1,
        "url": "https://open.spotify.com/track/2gZUPNdnz5Y45eiGxpHGSc"
    },
    {
        "id": "3FjYAj4hg5KiXpVAMjpLKq",
        "name": "This Is the Day",
        "artist": "The The",
        "album": "Soul Mining",
```

```
        "image_url": "https://i.scdn.co/image/ab67616d0000b2731bbff7a308778b39fc8c08ba",
        "duration": 297,
        "platform_id": 1,
        "url": "https://open.spotify.com/track/3FjYAj4hg5KiXpVAMjpLKq"
    }
]
```


API Agrégatrice :

Recherche de chanson équivalente sur Spotify/Deezer

Les méthodes s'appliquent pour un utilisateur connecté ou non

Obtention d'une chanson sur Spotify/Deezer avec une chanson sur Deezer/Spotify :

- URL : */api_agg/aggreg*
- Méthode : POST
- Payload attendu :

```
{
   "song" : "chanson voulue"
   "platform" : "plateforme sur laquelle on veut la chanson"
}
```

- Retour : Le lien vers la chanson voulue par l'utilisateur sur la plateforme requise (Spotify ou Deezer) du type : *spotify://track/track_id* ou *deezer://www.deezer.com/track/track_id* qui permet d'ouvrir les applications Spotify ou Deezer directement sur l'appareil de l'utilisateur.