

## SOMMAIRE :

1-	Présentation du tp	2
2-	Fonctionnement de l'application	3
3-	Le modèle Music	4
4-	Accès à la base de données MySQL	4
5-	APIs	5
6-	Validation	6
7-	CORS & tests unitaires.	6

## Accès au code du projet :

[https://github.com/PierreBourdeau/uv\\_intes\\_tp10.git](https://github.com/PierreBourdeau/uv_intes_tp10.git)

## Énoncé :

///// TP10 /////

Créer une API afin de gérer une liste de chansons dans une base de données MySQL

- Une chanson possède:
- un guid (string, obligatoire, généré par l'application)
- un genre (string, 80 caractères max)
- un titre (string, 80 caractères max, obligatoire)
- une durée en seconde (entier)
- un auteur (string, 100 caractères max, obligatoire)

Routes:

- [GET] /songs , retourne la liste des chansons
- [GET] /songs/{guid}, retourne une chanson
- [POST] /songs , ajoute une chanson
- [PUT] /songs/{guid} , modifie une chanson
- [GET] /songs/artists?q={artist} , retourne la liste des chanson d'un artiste donné

Bonus:

- [GET] /artists, retourne une liste d'artistes contenant chacune des chansons qu'il a faite

## 1- Présentation du tp

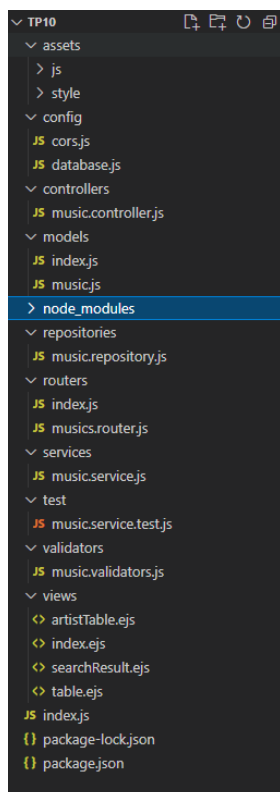
Le but de l'application est donc de pouvoir gérer une liste de chansons contenant 5 paramètres :

Un id (unique), un genre, un titre, une durée (en seconde, entier), un auteur. Ces musiques doivent être stockées dans une base de données MySQL.

L'application est donc réalisée sur base d'un serveur ExpressJS sur base de la plateforme logicielle NodeJS.

Les musiques sont enregistrées sur une base de données MySQL.

Voici comment se décompose l'architecture de l'application :



La partie front-end de l'application est réalisé avec le framework CSS Bootstrap, et de la bibliothèque JS jQuery embarqués via CDN, non embarqués via NPM.

Le module EJS « Embedded JavaScript templating », a également été utilisée afin de générer des templates HTML côté serveur embarquant du JS.

Le code javascript (hors framework) côté client est contenu dans le dossier **assets/js/mais.js**. Il contient principalement les scripts de requêtes Ajax envoyées vers le serveur.

La liaison avec la base de données est assurée par le module mysql2. Cette liaison est gérée par Sequelize ORM, pour l'association en modèle objet.

## 2- Fonctionnement de l'application

### TP - 10


Title :


Artist :


Genre :

Length : (in s)

 Send

 Song list

 Search for song

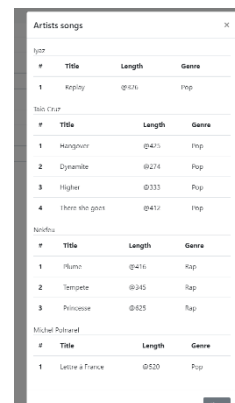
 Show artists songs

Formulaire pour créer une nouvelle musique nécessitant de renseigner les champs :

- Titre
- Artist
- Genre
- Length (s)

«[Send]» pour envoyer.

Ouvre un « modal » bootstrap pour afficher une liste des artistes et leurs musiques respectives via l'API `findByArtist`.

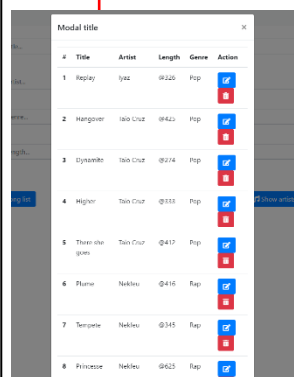


















#	Title	Length	Genre
Toto Cutugno			
1	Replay	@0:36	Pop
Hangingman			
1	Hangingman	@0:45	Pop
2	Dynamite	@0:24	Pop
3	Higher	@0:33	Pop
4	Three the goats	@0:12	Pop
Nekfeu			
#	Title	Length	Genre
1	Flamant	@0:16	Rap
2	Tempête	@0:45	Rap
3	Princesse	@0:25	Rap
Michael Polzella			
#	Title	Length	Genre
1	Lettre à France	@0:20	Pop

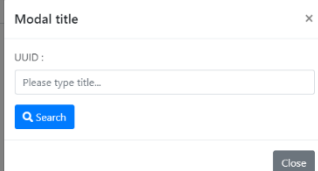
Ouvre un « modal » bootstrap pour afficher la liste de toutes les musiques (API `findAll`)

Permet également de :  
> supprimer les musiques de la base de données (bouton rouge) via l'API `delete`, l'guid étant déjà fourni.

> modifier une musique existante (ouvre un second modal avec un formulaire




#	Title	Artist	Length	Genre	Action
1	Replay	lyaz	@0:20	Pop	 
2	Hangingman	Toto Cutugno	@0:45	Pop	 
3	Dynamite	Toto Cutugno	@0:24	Pop	 
4	Higher	Toto Cutugno	@0:33	Pop	 
5	Three the goats	Toto Cutugno	@0:12	Pop	 
6	Flamant	Nekfeu	@0:16	Rap	 
7	Tempête	Nekfeu	@0:45	Rap	 
8	Princesse	Nekfeu	@0:25	Rap	 



Modal title

UUID :

 Search

Close

Ouvre un « modal » bootstrap pour afficher un formulaire demandant un guid, « [Search] » permet de lancer la recherche en fonction du guid fournit avec l'API `findById`.

Affiche la musique trouvée s'il y a un résultat, un message d'erreur sinon.

### 3- Le modèle Music

Voici comment sont construites les musiques dans le modèle Sequelize :

On y retrouve le champ « id » : généré automatiquement via Sequelize en tant que clef primaire UUID.

Les champs « STRING » : title, genre, artist ;

Le champs length, étant un entier « INTEGER ».

```
1 module.exports = (sequelize, Sequelize) => {
2   const Music = sequelize.define("music", {
3     id: {
4       type: Sequelize.UUID,
5       defaultValue: Sequelize.UUIDv1,
6       primaryKey: true,
7     },
8     title: {
9       type: Sequelize.STRING,
10    },
11    genre: {
12      type: Sequelize.STRING,
13    },
14    length: {
15      type: Sequelize.INTEGER,
16    },
17    artist: {
18      type: Sequelize.STRING,
19    }
20  });
21  return Music;
22 };
23
24
```

### 4- Accès à la base de données MySQL

```
1 module.exports = {
2   HOST: "localhost",
3   USER: "root",
4   PASSWORD: "",
5   DB: "tp10",
6   dialect: "mysql",
7   pool: {
8     max: 5,
9     min: 0,
10    acquire: 30000,
11    idle: 10000,
12  },
13 };
14
```

L'accès à la base de données est paramétré de la façon suivante.

La base de données est nommée « tp10 », en local.

L'utilisateur est root.

Cette configuration est transmise à Sequelize pour accéder à ta base de données afin d'assurer l'initialisation automatique de la table « music » (Cf. Image du modèle ci-dessus) et sa synchronisation via la fonction auto exécutante :

```
(async () => {
  await db.sequelize.sync();
})();
```

Ainsi si le modèle Music est modifié, la table music sera mise à jour.

```
const dbConfig = require("../config/database");

const Sequelize = require("sequelize");
const sequelize = new Sequelize(dbConfig.DB, dbConfig.USER, dbConfig.PASSWORD, {
  host: dbConfig.HOST,
  dialect: dbConfig.dialect,
  operatorsAliases: false,

  pool: {
    max: dbConfig.pool.max,
    min: dbConfig.pool.min,
    acquire: dbConfig.pool.acquire,
    idle: dbConfig.pool.idle,
  },
});

const db = {};

db.Sequelize = Sequelize;
db.sequelize = sequelize;

db.music = require("../music.js")(sequelize, Sequelize);

module.exports = db;
```

## 5- APIs

L'accès aux APIs est assuré via les routes suivantes :

```
1  const express = require("express");
2  const musicRouter = express.Router();
3  const MusicController = require("../controllers/music.controller");
4  const { validate } = require("express-validation");
5  const MusicValidator = require("../validators/music.validators");
6  const MusicService = require("../services/music.service");
7  |
8  // Begin Router
9
10 musicRouter
11   .route('/')
12   .get(async function(req,resp) {
13     resp.render('index', {musics : await MusicService.findAll()});
14   });
15
16 musicRouter
17   .route("/songs")
18   .get(MusicController.findAll) // GET all songs
19   .post(validate(MusicValidator.validateCreate), MusicController.create); // POST create a song
20
21 musicRouter.route("/songs/:guid").get(MusicController.findById); // GET find a specific song by id
22
23 musicRouter.route("/songs/:guid/delete").get(MusicController.delete); // GET delete the song with the specific guid
24
25 musicRouter.route("/songs/:guid").put(validate(MusicValidator.validateUpdate), MusicController.update); // PUT modify specific song (find by id)
26
27 musicRouter.route("/artists").get(MusicController.findByArtist); // GET find all songs of the specific artist
28 // end Router
29 module.exports = musicRouter;
```

Ces routes sont définies dans une instance de Express Router dans le dossier « **routers/music.routers.js** ».

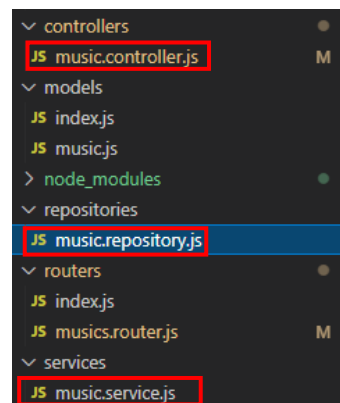
Nous retrouvons l'accès aux APIs suivantes :

- findAll → permet d'obtenir l'ensemble du contenu de la table « music », autrement dit toutes les musiques et leurs champs.
- create → permet de créer une nouvelle musique. Les champs du formulaire sont validés via le middleware : MusicValidator et sa fonction validateCreate.
- findById → permet de retrouver une musique spécifique en fournissant son guid.
- delete → permet de supprimer une musique de la base de données.
- update → permet de mettre à jour les informations d'une musique. De la même façon que pour « create », les informations des champs de saisie du formulaire sont vérifiées via le middleware MusicValidator et sa fonction validateUpdate.
- findByArtist → permet d'obtenir l'ensemble des artistes et les musiques qu'ils ont réalisé.

Ces APIs sont définies dans le contrôleur : MusicController qui fait appel aux différents services définis dans MusicService. Les services vont agir sur la base de données à travers le repository : MusicRepository.

Ils sont définis dans les fichiers suivants.

- MusicController : music.controller.js
- MusicService : music.service.js
- MusicRepository : music.repository.js



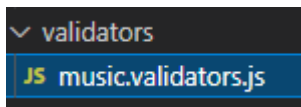
## 6- Validation

Comme explicité précédemment, les fonctions **create** et **update** nécessitent une validation côté serveur afin de s'assurer de la qualité des informations transmises par l'utilisateur dans le formulaire.

**Remarque :** il est également possible et préférable d'effectuer une première validation côté client (qui ne se suffit pas à elle-même) pour éviter l'envoi de requêtes non nécessaires. Cette première validation peut notamment vérifier le type des données inscrites dans les champs et leur correspondance, si les champs sont vides etc...

La validation côté client est assurée par le module « express-validation ».

Le middleware est défini dans le fichier suivant :



Il est défini de la façon suivante :

```
validators > JS music.validators.js > [0] MusicValidators > validateUpdate > body
1  const { Joi } = require("express-validation");
2  /**
3   * Music Validators
4   */
5
6  const MusicValidators = {
7    validateCreate: {
8      body: Joi.object({
9        title: Joi.string().max(80).required(),
10       genre: Joi.string().max(80).required(),
11       length: Joi.number().min(0).required(),
12       artist: Joi.string().max(100).required(),
13     }),
14   },
15   validateUpdate: {
16     params: Joi.object({
17       guid: Joi.string().guid().required(),
18     }),
19     body: Joi.object({
20       title: Joi.string().max(80).required(),
21       genre: Joi.string().max(80).required(),
22       length: Joi.number().min(0).required(),
23       artist: Joi.string().max(100).required(),
24     }),
25   },
26 };
27
28 module.exports = MusicValidators;
29
```

## 7- CORS & tests unitaires.

Le module npm cors est implémenté et paramétré dans l'application. Il est fonctionnel et en mesure de communiquer avec une autre application sur le port 4200.

### Compte rendu

```
const cors = require("cors");
const whitelist = ["http://localhost:4200"];

var corsOptionsDelegate = (req, callback) => {
  var corsOptions;

  if (whitelist.indexOf(req.header("Origin")) !== -1) {
    corsOptions = { origin: true };
  } else {
    corsOptions = { origin: false };
  }
  callback(null, corsOptions);
};

exports.corsWithOptions = cors(corsOptionsDelegate);
```

Néanmoins, j'ai fait le choix d'implémenter un front-end côté client avec le framework Bootstrap, la librairie jQuery. Pour gérer le templating, le projet utilise EJS. Ainsi le projet ne nécessite pas de communiquer avec une autre application.

Les réponses transmises ne sont pas des objets JSON, mais directement des rendus de documents HTML.