NodeJs : Compte rendu TP10

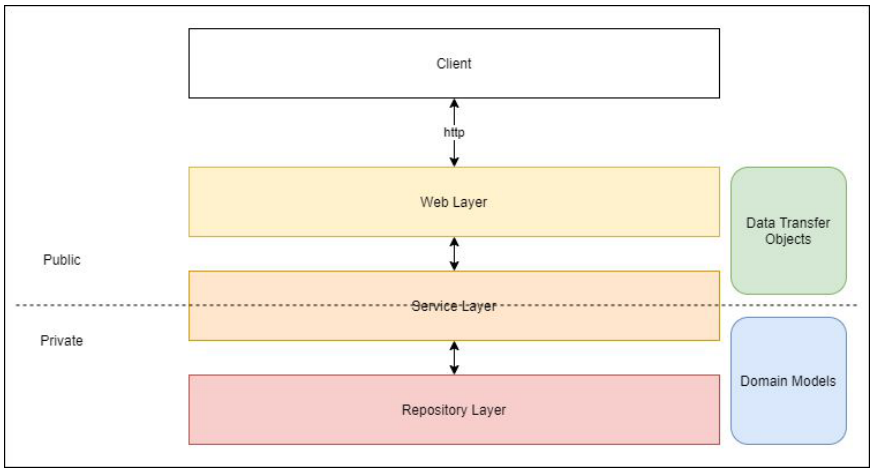
Objectif

Le but de ce TP est de créer une API permettant de gérer une liste de chansons dans une base de données MySql. Nous y trouverons des actions de recherche, d'ajout et de modification.

Mise en place d'une architecture 3 couches

L'architecture mise en place est composée de trois couches fonctionnelles qui représentent chacune une entité logique :

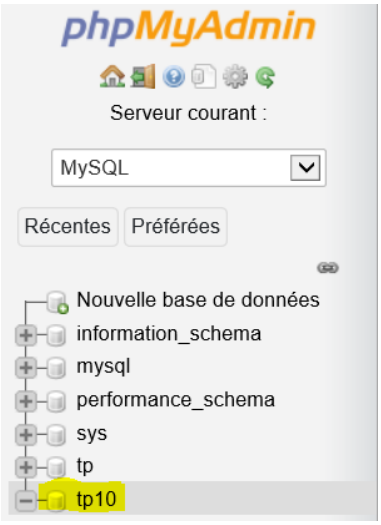
- La couche **Controller** (ou Web) : Prend en charge le traitement des entrées de l'utilisateur et du renvoi de la réponse. Prend également en charge la gestion des exceptions.
- La couche **Service** : Prend en charge les traitements et la logique métiers, les transactions et les appels à des services externes (ex: Mailer).
- La couche **Repository** : Prend en charge les opérations de lecture et d'écriture avec les bases de données.



Configuration de la base de données

Notre API requiert un accès à une base de données dans laquelle seront stockées les chansons. Pour ce faire, nous avons tout d'abord installé un serveur MySql sur lequel on crée une base.

Ensuite au niveau de l'API, nous avons installé Sequelize, un ORM (Object Relational Mapping) permettant d'interagir avec plusieurs types de base de données dont MySql. L'accès à la base de données est configuré dans un fichier /config/database.js. On y précise les caractéristiques telles que le nom de la base de données, ou les informations de connexion login/MDP.




```
module.exports = {
  HOST: "localhost",
  USER: "root",
  PASSWORD: "",
  DB: "tp10",
  dialect: "mysql",
  pool: {
    max: 5,
    min: 0,
    acquire: 30000,
    idle: 10000,
  },
};
```

Conception du modèle de chanson “Song”

Pour pouvoir intégrer le concept d’une chanson dans notre base de données, il est primordial de faire une modélisation. Nous avons déterminé qu’une chanson devait posséder :

- un guid (obligatoire et généré par l’application)
- un genre (chaîne de caractère de 80 caractères maximum)
- un titre (chaîne de caractère, obligatoire et de 80 caractères maximum)
- une durée (nombre entier)
- un auteur (chaîne de caractère, obligatoire et de 100 caractères maximum)

Concrètement, nous utilisons le mot clé “type” pour définir le type, “required” pour définir l'obligation, et “maxlength” pour définir la taille maximum des chaînes de caractères.

```
module.exports = (sequelize, Sequelize) => {
  const Song = sequelize.define("song", {
    id: {
      type: Sequelize.UUID,
      defaultValue: Sequelize.UUIDV1,
      primaryKey: true,
    },
    genre: {
      type: Sequelize.STRING,
      maxlength: 80,
      required: false,
    },
    titre: {
      type: Sequelize.STRING,
      maxlength: 80,
      required: true,
    },
    duree: {
      type: Sequelize.INTEGER,
      required: false,
      validate: {
        min: 0
      },
    },
    auteur: {
      type: Sequelize.STRING,
      maxlength: 100,
      required: true,
    },
  });
```

Création des API

Pour pouvoir gérer la liste des chansons, notre API doit posséder plusieurs routes correspondant à une opération donnée :

- [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 chansons d'un artiste donné. A noter que le nom de l'artiste doit être identique à celui en base de données pour être affiché (pas de Contains).

Vous trouverez en annexe des exemples à ces requêtes.

Couche de validation de données

Avant d'intégrer des données dans notre base, il est souhaitable de les valider pour être sûr que ces données correspondent réellement au format attendu. Nous souhaitons valider les données lors des phases de création et de mise à jour des chansons.

Pour cela nous utilisons la librairie Joi qui permet de créer des schémas de description des données, ainsi que le middleware express-validation qui valide les requêtes et envoie des messages d'erreurs en cas d'échec de validation.

```
const { Joi } = require("express-validation");

const SongValidators = {
  validateCreate: {
    body: Joi.object({
      genre: Joi.string().max(80),
      titre: Joi.string().max(80).required(),
      duree: Joi.number().integer().min(0),
      auteur: Joi.string().max(100).required()
    }),
  },
  validateUpdate: {
    params: Joi.object({
      guid: Joi.string().guid().required(),
    }),
    body: Joi.object({
      genre: Joi.string().max(80),
      titre: Joi.string().max(80).required(),
      duree: Joi.number().integer().min(0),
      auteur: Joi.string().max(100).required()
    }),
  }
};

module.exports = SongValidators;
```

CORS : L'API accessible depuis une application web

Le « Cross-origin resource sharing » (CORS) ou « partage des ressources entre origines multiples » est un mécanisme qui consiste à ajouter des en-têtes HTTP afin de permettre à un utilisateur d'accéder à des ressources d'un serveur situé sur une autre origine que le site courant.

Un agent utilisateur réalise une requête HTTP multi-origine (cross-origin) lorsqu'il demande une ressource provenant d'un domaine, d'un protocole ou d'un port différent de ceux utilisés pour la page courante.

```
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);
```

En NodeJs, il existe un module CORS permettant de gérer rapidement le Cross Origin Resources Sharing en ajoutant un middleware configurable. Malheureusement, nous ne possédons pas de front dans ce TP pour tester qu'il soit bien opérationnel.

Test unitaires

Pour s'assurer du bon fonctionnement du programme, nous avons réalisé des tests unitaires. Ces tests ont pour but de prouver que les méthodes implémentées dans le service fonctionnent correctement. Pour cela nous devons tester le maximum de possibilités possibles.

```
33  describe("SongService", () => {
34    describe("isLonger", () => {
35      it("should return true", () => {
36        const result = SongService.isLonger(song1, song2);
37        expect(result).to.be.true;
38      });
39      it("should return false", () => {
40        const result = SongService.isLonger(song2, song1);
41        expect(result).to.be.false;
42      });
43      it("should return false", () => {
44        const result = SongService.isLonger(song3, song4);
45        expect(result).to.be.false;
46      });
47    });

PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE

SongService
  isLonger
    ✓ should return true
    ✓ should return false
    ✓ should return false
```

Dans le cas de la méthode **isLonger()**, déterminant si une chanson A est plus longue qu'une chanson B, nous devons tester 3 possibilités :

- A est plus longue que B.
- A est moins longue que B.
- A et B sont de même longueur.

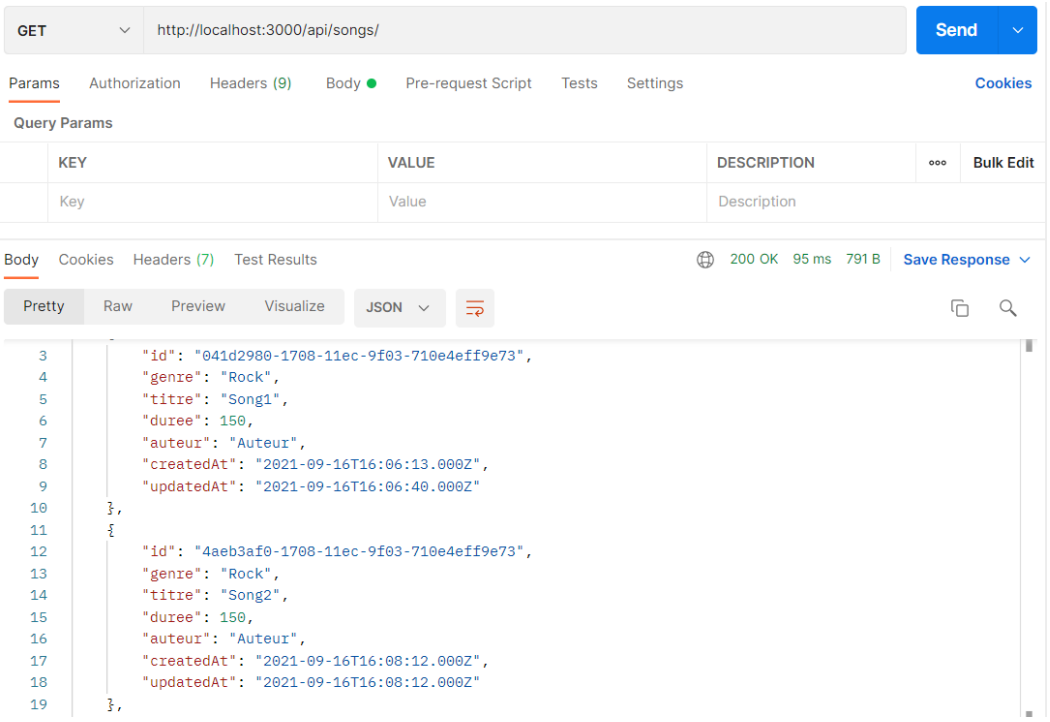
Avec nos tests, nous nous attendons donc respectivement aux résultats true, false et false puisque nous avons (Annexe) :

- Song1 > Song2.
- Song3 = Song4.

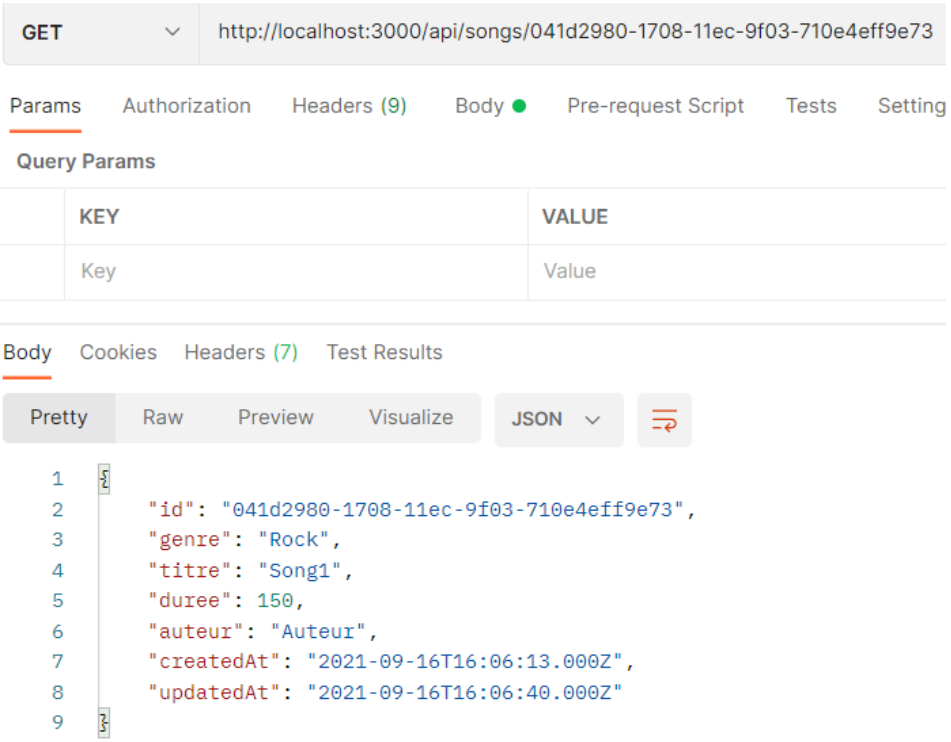
Nous avons également un test pour vérifier si deux chansons proviennent du même artiste, visible en Annexe.

Annexes

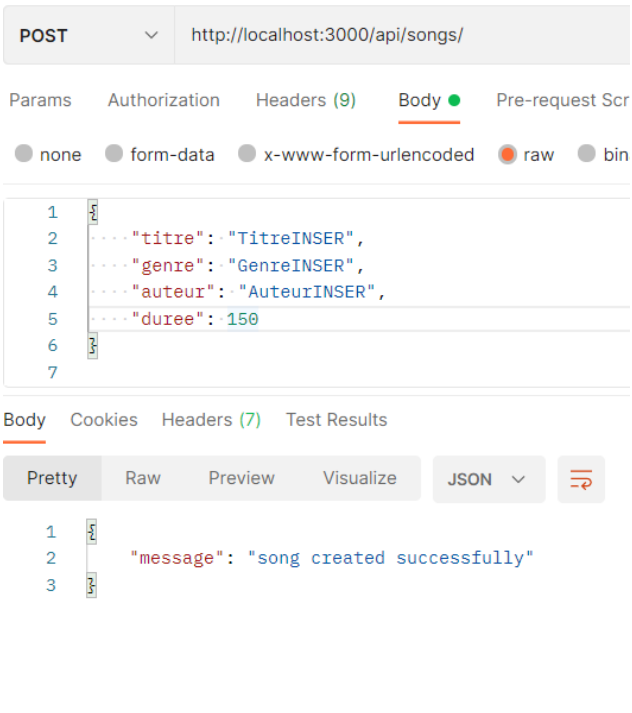
1. [GET] /songs

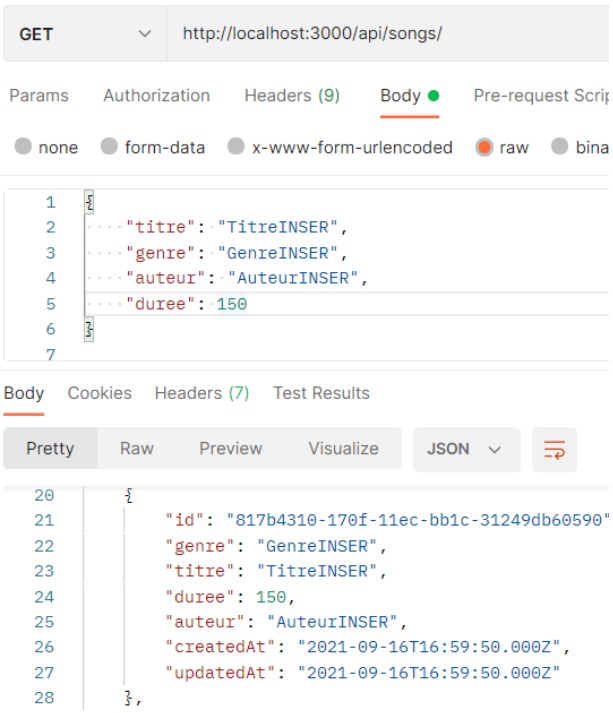


2. [GET] /songs/{guid}

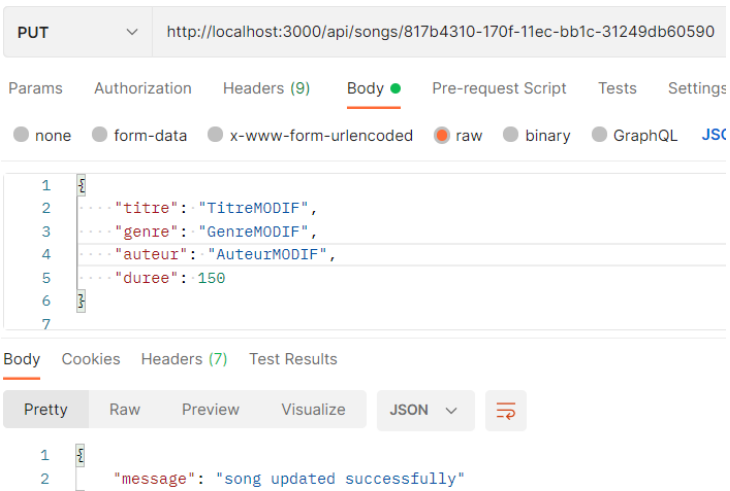


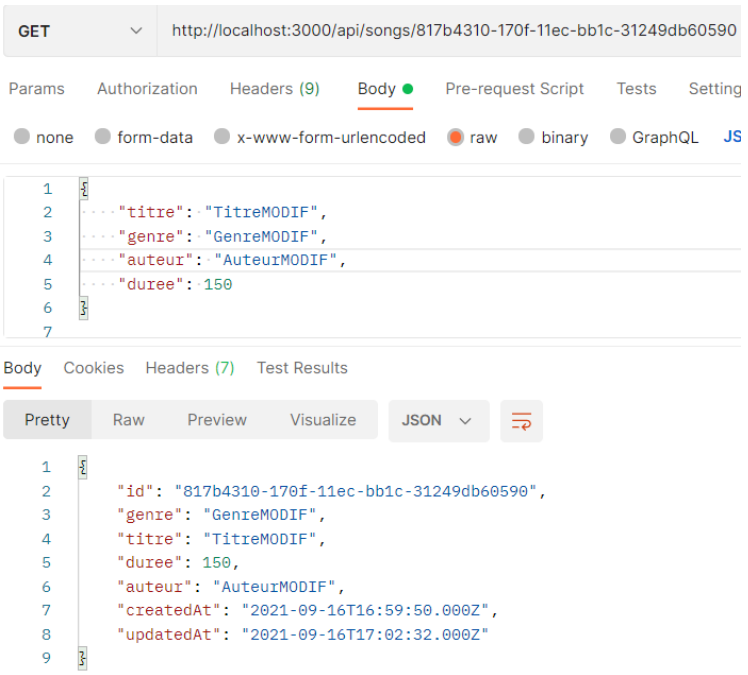
3. [POST] /songs



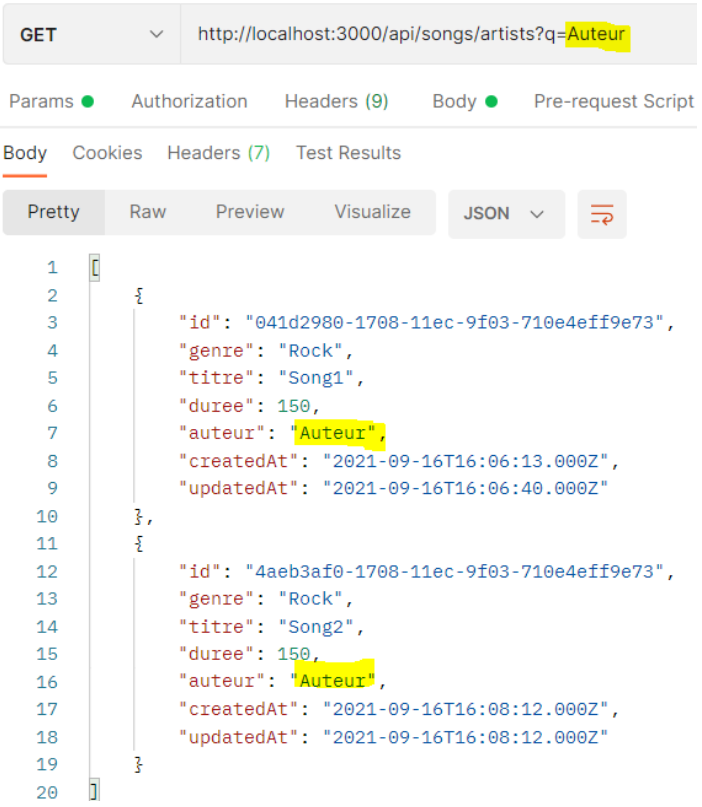


4. [PUT] /songs/{guid}





5. [GET] /songs/artists?q={artist}



6. Description de Song1, Song2, Song 3 et Song 4

```
const song1 = {
  genre: "Pop",
  titre: "La banane",
  duree: 200,
  auteur: "Philippe Katerine",
};
const song2 = {
  genre: "Pop",
  titre: "Anissa",
  duree: 100,
  auteur: "Wejdenne",
};
const song3 = {
  genre: "Pop",
  titre: "Louxor j'adore",
  duree: 150,
  auteur: "Philippe Katerine",
};
const song4 = {
  genre: "Rock",
  titre: "Allumez le feu",
  duree: 150,
  auteur: "Johnny Haliday",
};
```

7. Tests de isLonger() et isFromSameArtist()

```
describe("SongService", () => {
  describe("isLonger", () => {
    it("should return true", () => {
      const result = SongService.isLonger(song1, song2);
      expect(result).to.be.true;
    });
    it("should return false", () => {
      const result = SongService.isLonger(song2, song1);
      expect(result).to.be.false;
    });
  });
  describe("isFromSameArtist", () => {
    it("should return true", () => {
      const result = SongService.isLonger(song1, song3);
      expect(result).to.be.true;
    });
    it("should return false", () => {
      const result = SongService.isLonger(song2, song1);
      expect(result).to.be.false;
    });
  });
});
```

```
SongService
  isLonger
    ✓ should return true
    ✓ should return false
  isFromSameArtist
    ✓ should return true
    ✓ should return false


4 passing (13ms)
```