

Persistance des données avec l'ORM Sequelize.

Sequelize est un module npm qui permet d'accéder à une BDD SQL sans jamais écrire le moindre code SQL. Toutes les actions habituelles du SQL sont accessible via des objets. Les programmes comme sequelize s'appelle des ORM (object-relational mapping) c'est une surcouche (un interface) du SQL qui permet un accès simple, rapide et orienté objet à la BDD.

A titre d'exemple une requête comme :

```
SELECT * FROM User WHERE User.id==1
```

S'écrit sous sequelize :

```
const user = await User.findByPk(1);
```

Une relation One to Many se crée comme suit :

```
Category.hasMany(Product);  
Product.belongsTo(Category);
```

Ce cours se déroulera en deux parties.

1. Découverte de sequelize, les fonctionnalités de bases et la documentation.
2. Projet : Création d'un projet Pokedex de zéro du front-end au back-end. Le back-end sera fait d'un serveur express et de sequelize pour l'accès simplifié à la BDD.

Installer sequelize

Dans un dossier spécifique à votre projet back-end faites :

```
npm init # Répondez au questions avant de faire la commande suivante...  
npm install express cors sequelize mysql2
```

Il est obligatoire d'installer mysql2 pour le fonctionnement de sequelize.

Créer une BDD dans PhpMyAdmin

Pour se connecter sequelize a besoin d'une BDD sql et d'un utilisateur ayant tout les droits sur la BDD.

Vous pouvez mettre en place rapidement un serveur mysql et phpMyAdmin avec docker.

```
docker network create lamp-net
```

```
docker run -d --name lamp-mysql --network=lamp-net -e MYSQL_ROOT_PASSWORD=root  
-p 3306:3306 mysql
```

```
docker run -d --name lamp-pma --network=lamp-net -e PMA_HOST=lamp-mysql  
-p 8080:80 phpmyadmin
```

```
docker start lamp-mysql  
docker start lamp-pma
```

Rendez-vous ensuite sur localhost:8080 pour accéder à phpMyAdmin.

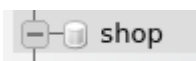
Les identifiants sont :

- id : root
- mdp : root

1. Une fois connectée à phpMyAdmin crée une BDD nommée shop



2. et un utilisateur ayant tout les droits sur cette BDD.



On met en mot de passe shop et en identifiant shop aussi

Ajouter un compte d'utilisateur

Informations pour la connexion

Nom d'utilisateur :


Saisir une valeur ▾

shop

Nom d'hôte :

Tout hôte ▾

%



Mot de passe :

Saisir une valeur ▾

....

Force :  Extrêmement faible

Saisir à nouveau :

....

Extension d'authentification

Mise en cache de l'authentification sha2 ▾

Générer un mot de passe:

Générer

Base de données pour ce compte d'utilisateur

☐ Créer une base portant son nom et donner à cet utilisateur tous les privilèges sur cette base.

☐ Accorder tous les privilèges à un nom passe-partout (utilisateur_%).

☒ Donner tous les privilèges sur la base de données shop.

Privilèges globaux ☒ **Tout cocher**

NB : les noms de privilèges sont exprimés en anglais.

On accorde tout les droits à cet utilisateur

✔ L'utilisateur a ajouté un utilisateur.

En résumé les identifiants pour vous connectez à la BDD sont :

- **bddName** : shop
- **username** : shop
- **mdp** : shop
- **host** : localhost

Se connecter à la BDD avec Sequelize

Dans un fichier nommé `database.js`

/back-end/database.js

```
const { Sequelize, DataTypes } = require("sequelize");

const login = {
  database : "shop",
  username : "shop",
  password : "shop"
};

// Connexion à la BDD
const sequelize = new Sequelize(login.database, login.username, login.password, {
  host: 'localhost',
  dialect: 'mysql'
});

// Vérifier la connexion
sequelize.authenticate()
  .then(() => console.log("Connexion à la base de donnée shop"))
  .catch(error => console.log(error));
```

Créer une table

Avec sequelize les tables sql sont représentées sous la forme d'un objet.

La création d'une table se fait en deux étapes :

1. La définition de la table
2. La synchronisation avec la base de donnée.

Définir une table avec `sequelize.define()`

Soit une table Product contenant nom, stock et prix.

/back-end/database.js

```

const {Sequelize,DataType} = require("sequelize");

/* ...après la connexion... */

// Création de la table Products
const Product = sequelize.define("Product",{
  name : DataTypes.STRING,
  price : DataTypes.NUMBER,
  stock : DataTypes.INTEGER
});

```

Une fois la table défini j'applique les changement à la BDD avec la fonction sync.

```

const {Sequelize,DataType} = require("sequelize");

/* ...après la connexion... */

// Création de la table Products
const Product = sequelize.define("Product",{
  name : DataTypes.STRING,
  price : DataTypes.NUMBER,
  stock : DataTypes.INTEGER
});

(async function(){
  // Application des changement à MySQL
  await sequelize.sync({force : true}); // +
})();

```

Puis j'exporte sequelize et ma table Product pour m'en servir dans d'autre fichier.

```

const {Sequelize,DataType} = require("sequelize");

/* ...après la connexion... */

// Création de la table Products
const Product = sequelize.define("Product",{
  name : DataTypes.STRING,
  price : DataTypes.NUMBER,
  stock : DataTypes.INTEGER
});

(async function(){
  // Application des changement à MySQL
  await sequelize.sync({force : true}); // +
})();

module.export.Product = Product;    // J'exporte le modèle Product
module.exports.sequelize = sequelize; // J'exporte aussi sequelize au cas où pour plus

```

La fonction sync fait parti des fonctions asynchrone de sequelize (elle effectue une action sur la BDD, cette action peut prendre du temps elle est donc encapsulé dans une Promise).

J'utilise la syntaxe `async await` pour attendre la fin de la fonction avant d'exécuter la suite du programme.

`{force : true}` permet d'écraser les données de la table quand le serveur redémarre ce qui est pratique en développement.

Vous devriez voir une table apparaitre dans PHPMysqlAdmin.

Ajouter un élément à la table

1. Créer un fichier `app.js` .
2. Importez l'objet `Product` dans le fichier `app.js`.

```

const {Product} = require("../database.js");

```

Ajoutez un produit avec la fonction `Product.create()` .

app.js

```
const {Product} = require("./database.js");

Product.create({
  name : "Nike air",
  price : 100,
  stock : 24
});
```

Et voilà un produit à été ajouté à la BDD.

Je peux encapsuler facilement tout ça dans une route express.

```
const express = require("express");
const app = express();
const {Product} = require("./database.js");

app.use(express.json());

app.post("/product", async (req, res) => {
  const newProduct = req.body;

  const product = await Product.create({
    name : newProduct.name,
    price : newProduct.price,
    stock : newProduct.price
  });
  res.status(200).json(product);
});
```

Et voilà rien de bien compliqué. Il manque encore les erreurs 400 et 500 mais tout ceci pourra se faire plus tard.

Récupérer tout les produits

```
const products = await Product.findAll();
```

Récupérer un produit via son id

```
const product = await Product.findByPk(3);
```

Récupérer un produit via son name

```
const nameToSearch = "Nike air max";
const product = await Product.findAll({
  where : {
    name : nameToSearch // name = "Nike air max"
  }
})
```

Les conditions

Pour construire des conditions, `sequelize` utilise un système d'objet représentant les opérateurs OR , AND , LIKE , etc.

```
// J'importe l'objet Op depuis sequelize.
const { Op } = require("sequelize");

Product.findAll({
  // L'objet where peut avoir comme attributs
  where: {
    // le nom d'une colonne
    name : "Puma taille 42",
    // Un opérateur conditionnel
    [Op.or]: [ // name="Air max" OR price=13
      { name: "Air max" },
      { price: 13 }
    ]
  }
});
```

Par défaut le nom d'une colonne vérifie l'égalité.


```
const { Op } = require("sequelize");

Product.findAll({
  // L'objet where peut avoir comme attributs
  where: {
    // le nom d'une colonne
    name : "Puma taille 42",    // name = "Puma taille 42"
  }
});
```

Mais je peux utiliser passer un objet en tant que valeur pour une colonne et placer d'autre opérateurs logiques à l'intérieur comme :

```
Product.findAll({
  // L'objet where peut avoir comme attributs
  where: {
    // je place un objet dans le nom de la colonne
    price : {
      [Op.gt] : 10    // price > 10 (greater)
    }
  }
});
```

Je peux placer autant d'opérateurs que je le souhaite en tant qu'attribut de la colonne et ainsi vérifier si plusieurs conditions sont vraies.

```
Product.findAll({
  // L'objet where peut avoir comme attributs
  where: {
    // Je place un objet dans le nom de la colonne
    price : {
      [Op.gt] : 10,    // price > 10 (greater)
      [Op.ne] : 10,    // price != 10 (not equal)
      [Op.gte] : 10,   // price >= 10 (greater equal)
      [Op.lt] : 10,    // price < 10 (greater equal)
      [Op.lte] : 10,   // price <= 10 (greater equal)
    }
  }
});
```

Regardez absolument la liste de tout les opérateurs possible avec sequelize !

Supprimer un produit

```
await Product.destroy({
  where: {
    firstName: "Puma taille 42"
  }
});
```

[Doc delete](#)

Modifier un produit

```
const newValues = {
  name : "Converse"
}
await Product.update(newValues, {
  where: {
    id: 2
  }
});
```

[Doc update](#)

Getting started sequelize

Dans la doc de sequelize consultez :

- <https://sequelize.org/docs/v6/getting-started/>
- <https://sequelize.org/docs/v6/core-concepts/model-basics/>
- <https://sequelize.org/docs/v6/core-concepts/model-instances/>
- <https://sequelize.org/docs/v6/core-concepts/model-querying-basics/>
- <https://sequelize.org/docs/v6/core-concepts/model-querying-finders/>
- <https://sequelize.org/docs/v6/core-concepts/getters-setters-virtuals/>

Projet Pokedex

Vous allez concevoir de A à Z une application responsive, du back-end au front-end.

Objectifs

- Créer le diagramme de cas d'utilisation d'un pokedex, appelez moi pour le valider.
- Mettre en place un back-end nodejs / express / sequelize
- Mettre en place un front-end JS pour ce pokedex.