NodeJS : lien avec MySQL

- Express Generator
- Formulaire de contact – comment conserver les données ?
- **Liste des contacts avec une base de données**
- Création d’une base de données MySQL
- Configuration et connexion
- Création d’un modèle

Liste de contacts

- Conserver les contacts entrés via le formulaire, au moyen d’une base de données
- Recréer le projet avec le formulaire, avec les vues, les routes et les styles
- Ajouter une base de données MySQL et établir le lien avec l’application
- Stocker les données du formulaire lors de l’envoi et de l’affichage
- Afficher l’historique de tous les messages

Express Generator

- **Nouveau projet** avec **Express Generator**
- Pas besoin de l'installer de nouveau si global
  - **express listeContacts --view=ejs**
  - **npm install**
  - **set DEBUG=listeContacts:* && npm start**
- L'arborescence des fichiers va être créée et le moteur de template choisi est **EJS**
- Les modules sont installés
- Démarrage en mode DEBUG

Installation de l'application

```
E:\TELETRAVAIL\nodeProjects\demo_mysql>express listeContacts --view=ejs

   create : listeContacts\
   create : listeContacts\public\
   create : listeContacts\public\javascripts\
   create : listeContacts\public\images\
   create : listeContacts\public\stylesheets\
   create : listeContacts\public\stylesheets\style.css
   create : listeContacts\routes\
   create : listeContacts\routes\index.js
   create : listeContacts\routes\users.js
   create : listeContacts\views\
   create : listeContacts\views\error.ejs
   create : listeContacts\views\index.ejs
   create : listeContacts\app.js
   create : listeContacts\package.json
   create : listeContacts\bin\
   create : listeContacts\bin\www

   change directory:
     > cd listeContacts

   install dependencies:
     > npm install

   run the app:
     > SET DEBUG=listecontacts:* & npm start
```

Modification de l’application

- Modifier le **package.json**, pour ajouter nodemon
- Supprimer les routeurs inutiles (dans **app.js** et **users.js** dans /routes)
- Construire les **vues** (index.ejs, contact_form.ejs, traiter_form.ejs, error.ejs) et adapter **style.css**
- Ajouter des nouvelles routes et des nouvelles vues pour le stockage et l’affichage des données dans une base MySQL
- Se rapprocher du modèle MVC

Création d'une base de données

- Ajout d'un nouveau module : **mysql**

  npm install mysql --save

- Création avec phpMyAdmin de la base **nodejs** et de la table **messages**

```
DROP TABLE IF EXISTS `messages`;
CREATE TABLE IF NOT EXISTS `messages` (
  `id` smallint(5) UNSIGNED NOT NULL AUTO_INCREMENT,
  `nom` varchar(50) NOT NULL,
  `message` varchar(500) NOT NULL,
  `datemessage` datetime NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
  PRIMARY KEY (`id`)
) ENGINE=MyISAM DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
COMMIT;
```

Configuration MySQL

- Ajout d'un dossier **config** et d'un fichier **dbconfig.js** qui peut être importé dans les autres fichiers
- Ce fichier contient les paramètres permettant de se connecter à la base de données

```
module.exports = {
    DB_HOST: "localhost",
    DB_USER: "root",
    DB_PASSWORD: "",
    DB_NAME: "nodejs",
    DB_PORT: 3306
}
```

Connexion à la DB

- On ajoute un dossier **models** (MVC) qui contient tous les fichiers liés à la base de données et on crée un fichier **db.js** avec la connexion

```
const mysql = require("mysql");
const dbConfig = require("../config/dbconfig.js");  // fichier de configuration de la DB

// Création de la connexion avec la DB
// 1. Chargement de la configuration pour la connection
const connection = mysql.createConnection({
    host : dbConfig.DB_HOST,
    user: dbConfig.DB_USER,
    password: dbConfig.DB_PASSWORD,
    database: dbConfig.DB_NAME,
    port: dbConfig.DB_PORT
});

// 2. Ouverture de la connexion
connection.connect(function(error){
    if (error) throw error;         // si erreur de connexion, ça s'arrête ici
    console.log("Connecté avec succès à la base de données !");
});

// On exporte pour les autres modules, la connexion à la base de données
module.exports = connection;
```

Création d’un modèle

- Pour ajouter un nouveau message dans la base de données, il faut créer un **modèle** correspondant à la table. On ajoute un fichier **message.model.js** dans le dossier **models**.

```
const sql = require("./db.js");

console.log("Je passe dans models/message.model.js")

// Constructeur
const Message = function(lemessage) {
    this.nom = lemessage.nom;
    this.msg = lemessage.msg;
    this.date_creation = new Date();
};




module.exports = Message;
```

Ajouter un message

- Dans le modèle, on ajoute une méthode **create** pour insérer un nouveau message

```
// Méthode pour créer un message et le sauvegarder dans la base de données
// newMsg : l'objet Message à créer et sauver dans la DB
// resultat : la réponse du serveur de DB quand je fais l'insertion (OK ou erreur)
Message.create = function(newMsg, resultat){
    sql.query("INSERT INTO messages(nom,message) VALUES (?,?)", [newMsg.nom, newMsg.msg],
    function(err,res){
        // si on a une erreur lors de l'insertion, on reçoit les données dans err
        // sinon, si tout se passe bien, on reçoit les données dans res
        if (err) {
            console.log("Erreur Message.create : ", err);
            resultat(err,null);
            return;
        }
        console.log("Réponse Message.create : ", res);
        resultat(null,res);
    });
};
```

Lire tous les messages

- Dans le modèle, on ajoute une méthode **readAll** pour lire tous les messages

```
// Méthode pour lire tous les messages dans la DB
Message.readAll = function(resultat) {
    sql.query("SELECT * FROM messages ORDER BY datemessage DESC", function(err,res){
        // Si erreur dans la lecture des données
        if (err) {
            console.log("Erreur Message.readAll : ", err);
            resultat(err,null);
            return;
        }
        // Si données reçues
        console.log("Réponse Message.readAll : ", res);
        resultat(null,res);
    });
};
```