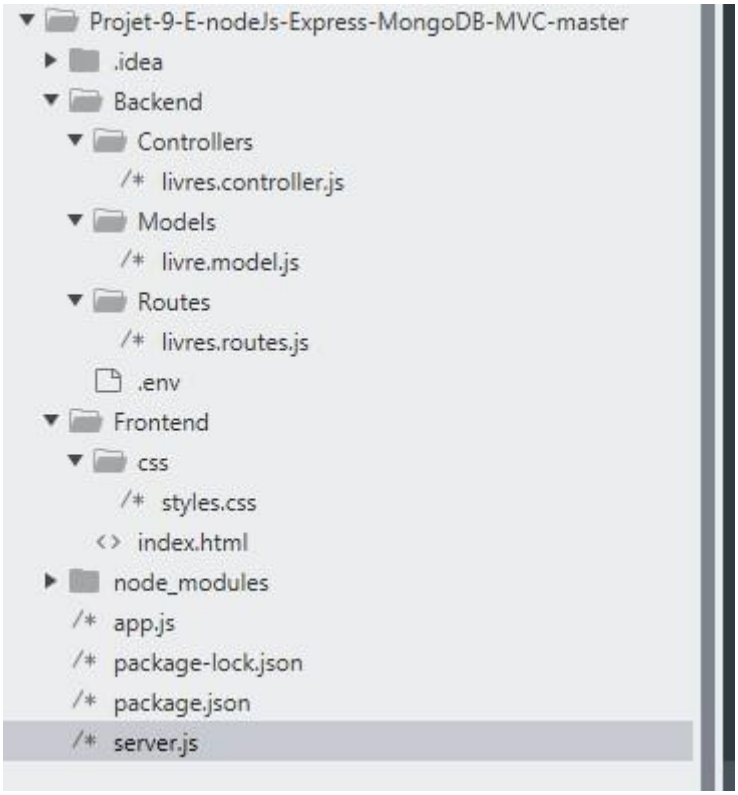
Projet 9 – E : NodeJs – Express - MongoDB – MVC (Modèle Vue Contrôleur) et SPA (Single Page Application)

Objectif :

Construire une application web à l'aide de NodeJS, Express et une connexion à MongoDb en utilisant le patron de conception (Design Pattern) Modèle Vue Controller.

EXEMPLE DE STRUCTURE :



Etapes :

- Générer un fichier package.json : npm init (votre point d'entrée est server.js)
- Installer les dépendances suivantes : Axios, express, mongoose et nodemon
- Créer votre fichier server.js :
    - Importer mongoose
    - Créer une connexion a MongoDB via mongoose.connect + URL locale et passer les options suivante : useUnifiedTopology: true, useNewUrlParser: true,useFindAndModify: false
    -
    - Connecter votre application à MongoDB via Mongoose (3 statuts : connected + error + disconnected)
    - Créer un dossier modèle et un fichier livre.model.js et importer le dans server.js
    - Créer un fichier app.js a la racine du projet et importer le dans server.js
    - Lancer votre serveur à l'aide de app.listen(sur le port 3000)

app.js | package.json | server.js

```
//Connexion a mongoose + url locale de mongodb
mongoose.connect("mongodb://localhost:27017/ecommerce",{
    useUnifiedTopology: true,
    useNewUrlParser: true,
    useFindAndModify: false
});

//Creation d'une promesse = ES6
mongoose.Promise = global.Promise;

//Message dans la console si la connexion est réussie
mongoose.connection.on('connected', () => {
    console.log(`Connexion a MongoDB avec succès !`);
});

//Detection de erreur de connexion
mongoose.connection.on('error', (err) => {
    console.log(`Erreur de connexion a MongoDB -> ${err.message}`);
});

//Appel du model (livreSchema)
require('./Backend/Models/livre.model');

//demarrer le serveur : import du fichier de configuration app.js
const app = require('./app')

//Port d'coute du server et message de succès
const server = app.listen(3000, () => {
    console.log(`Le serveur est demarré sur le port ${server.address().port}`)
})
```

- Dans votre fichier de configuration de l'application app.js :
    - Import d'Express
    - Instance du Framework : const app = express()
    - Import d'Axios : (Promesse basée sur client http pour les navigateurs et nodeJS
    - https://github.com/axios/axios et configurer Express

app.js | package.json | server.js

```
//Import de framework express
const express = require('express');
//Instance du framework dans une constante
const app = express();

const axios = require('axios');
//Appel des dossier et fichier contenu dans le dossier Public du Backend
app.use(express.static(__dirname + '/Frontend'));

//reconait les requète Objets comme des json Object
app.use(express.json());

//Reconnais les requètes Objet comme des strings et des arrays
app.use(express.urlencoded({
    extended: true
}))

//Option des requètes HTTP dans le headers
app.use((request, response, next) => {
    response.header("Access-Control-Allow-Origin", "*");
    response.header("Access-Control-Allow-headers", "Origin, X-Requested-With, Content-Type, Accept, Authorization");
    if(request.method === 'OPTIONS'){
        response.header('Access-Control-Allow-Methods', 'PUT, POST, PATCH, DELETE, GET');
        return response.status(200).json({});
    }
    next();
})

//Import du fichier  Routes/routes.js
const routes = require('./Backend/Routes/livres.routes');

//Utilisé les routes au point d'entrée
app.use('/', routes)

//Export du modules App.js
module.exports = app;
```

- Dans votre fichier modèles/livre.model.js :
    - Import de mongoose
    - Création d'un Schéma (new mongoose.Schema({Eléments de la collection livre}))
    - Export du module et donc du modèle (Schéma)
    - Module.export = mongoose.model('livres', LivresSchema, 'livres') ;
- Créer un dossier Controllers :
    - Créer un fichier livres.controller.js
    - Import de mongoose
    - Import de path = Le module path fournit de nombreuses fonctionnalités très utiles pour accéder et interagir avec le système de fichiers. Il n'est pas nécessaire de l'installer. Faisant partie du noyau Node.js, il peut être utilisé en le demandant simplement : copie JS. const chemin = require('chemin')
    - Import du livre modèle (Schéma)
    - Appel du frontend via un seul fichier index.html (SPA – Single Page Application)

```
exports.baseRoute = async (request, response) => {
    response.sendFile(path.resolve(__dirname + "../../../Frontend/index.html"))
}
```

    - Création des 5 fonctions de CRUD qui sont toutes exportées par défaut.
    - Afficher les livres = requête asynchrone = modele.find() proposé par Mongoose
    - Création d'une promesse qui retourne le modèle au format json

```
//Fonction pour recup tous livres de la collection livre au format json

exports.getLivres = async (request, response) => {
    const livres = await Livres.find();
    response.json(livres)
}
```

    - Livre Par ID : fonction asynchrone = stock et récupération de l'ID grâce aux routes et request.params.id
    - A l'aide du modèle : modele.findById({ _id : request.params.id }) proposé par Mongoose
    - Création d'une promesse, si ça marche on retourne un statut 200 et on stocke les valeurs dans data, sinon on retourne une erreur serveur 500

```
//Details de chaque
exports.getSingleLivre = async (request, response) => {
    //Requete de recup de l'id
    let livdreId = request.params.id;

    //MongoDB propose la fonction findById()
    await Livres.findById({
        _id: livdreId
    },
        //Si on detecte une erreur
        (err, data) =>{
            if(err){
                response.status(500).json({
                    message: 'Une erreur est survenue lors de la recherche du livre'
                });
            }else{
                response.status(200).json({
                    message: "Votre livre a été touvé",
                    data
                })
            }
        }
        )
}
```

- Ajouter des livres = requête asynchrone = instance du modèle + récupération des données dans le corp de la requête (request.body) et appel de la fonction save() de Mongoose
- On créer une promesse qui retourne un statut 201 au format json et une redirection sinon une erreur serveur code 500

```
//Ajouter un livres
exports.createLivre = async (request, response) => {
    //MongoDb et mongoose porose la method save()
    await new Livres(request.body).save((err, data) =>{
        if(err){
            response.status(500).json({
                message: 'Une erreur est survenue lors de la creation du livre'
            });
        }else{
            console.log("livre ajouté :" + response.status(201))
            response.redirect('/')
        }
    })
}
```

- Supprimer un livre = requête asynchrone + récupération de id passé en paramètre dans la route + request.params.id
- On utilise le modèle + la fonction deleteOne de Mongoose et on passe l'id en paramètre dans l'objet
- Si ça fonctionne on retourne un objet au format json + le statuts 200 sinon une erreur avec le code statuts 500 et un message

```
//Supprimer un livre

exports.deleteLivre = async (request, response) => {
    //Requet de recup de l'id
    let livdreId = request.params.id;
    //Mongoose  propose delete One
    await Livres.deleteOne({
        _id: livdreId
    },(err, data) => {
        if(err){
            response.status(500).json({
                message: 'Une erreur est survenue lors de la supression du livre'
            });
        }else{
            response.status(200).json({
                message: "Livre supprimer",
                data
            })
        }
    })
}
```

- La mise à jour d'un livre : requête http asynchrone + création d'un objet livreData qui est égale à l'instance du modèle + de nouvelles données récupérées dans le corps de la requête (donc dans le formulaire)
- On exécute une promesse et la mise a jour à l'aide du modèle et la fonction updateOne fournie par Mongoose
- Si ça marche on créer un nouvel objet qui remplace l'ancien donc un code statuts 201 sinon une erreur de type 500

```
//Mise a jour d'un livre

exports.updateLivre =  async (request, response) => {
    //Recuperer id
    let livresData = new Livres({
        _id: request.params.id,
        nomLivre: request.body.nomLivre,
        descriptionLivre: request.body.descriptionLivre,
        prixLivre: request.body.prixLivre,
        imageLivre: request.body.imageLivre
    });
    //Mongoose propose le methode findByIdAndUpdate(3 paramètres id + data + callback)
    await Livres.updateOne({_id: request.params.id}, livresData)
        .then(() => {
            response.status(201).json({
                message: 'Livre est a jour! c ok good',
            })
            console.log(response.data)
        })

        .catch((error) => {
                response.status(500).json({
                    error: error
                });
            }
        );
};
```

TOUTES LES REQUETES http SONT STOCKEE DANS DES FONCTIONS QUI SONT EXPORTEES, C'EST LE ROLE DU ROUTER D'APPELER LA FONCTION CONCERNEE EN FONCTION DE URL

- Créer un dossier Routes/ et un fichier livres.routes.js
- Import de mongoose d'express et d'express Router()
- Import du Controller livres.controller.js
- Router permet d'accéder aux méthodes API REST (GET, POST, PUT-PATCH et DELETE)
- La méthode prend 2 paramètres (URL + fonction du Controller)
- Ex : pour la route de base : router.get('/', livreController.baseRoute)
- Notre fichier de frontend (index.html) jour le rôle de notre SPA (Single Page Application) et affiche du contenu différent en fonction des url (une URL = Une requête http)

```
const mongoose = require('mongoose');
//const Livres = mongoose.model('livres')

//Appel du framework
const express = require('express');
//Creation des routes
const router = express.Router();

//Appel du fichier controller.js

const livreController = require('../Controllers/livres.controller');

//Route de base
router.get('/', livreController.baseRoute)


//Afficher tous le livrs au format json
router.get('/livres', livreController.getLivres)

//Afficher les details de chaque livre
router.get('/livres/:id', livreController.getSingleLivre)

//Creer un livre
router.post('/ajouter-livre', livreController.createLivre)

//Supprimer un livre
router.delete('/supprimer/:id', livreController.deleteLivre)

//Mettre a jour un livre
router.put('/edit-livres/:id/', livreController.updateLivre);

//Export du module router
module.exports = router;
```

LE FRONTEND (index.html)

- Squelette de base HTML 5
- Appel du Framework css Materialize et du jeu d'icones
- On charge le DOM et on utilise Axios pour afficher les livres (depuis url = localhost :3000/livres)
- On récupère notre vide (conteneur) avec document et querySelector
- On utilise Axios pour créer une requête http (méthode GET) pour récupérer les données depuis notre URL localhost :3000/livres au format json
- On créer une promesse et bouclé sur les valeurs à l'aide de forEach() qui appel la fonction afficherLivre via un callback()

```
<script>
    const livres = document.querySelector('#livres_row')
    axios.get('http://localhost:3000/livres')
    .then(livres => livres.data.forEach(afficherLivres))
```

- La fonction afficherLivre(livre) : Créer une et ajouter un ID et des classes avec Javascript

- Ajouter du contenu HTML avec innerHTML et la concaténation ES6 (back quottes Alt Gr + 7)

- A l'aide de la boucle forEach et livre passer en paramètre de la fonction : afficher les éléments 1 à 1.

```
function afficherLivres(livre){
    const livreItem = document.createElement('div')
    livreItem.id = "livres" + livre._id;
    livreItem.className = 'col s4';

    livreItem.innerHTML =`
        <div class="card col s12 hoverable">
            <div class="card-image">
                <img src="${livre.imageLivre}" alt="${livre.nomLivre}" title="${livre.nomLivre}" width="15%" style="padding-top: 10px">
            </div>
            <div class="card-content">
                <h3 class="blue-text lighten-1">${livre.nomLivre}</h3>
                <p>Référence : ${livre._id}</p>
                <p>Description : </p>
                <p>${livre.descriptionLivre}</p>
                <p class="red-text lighten-3">Prix : ${livre.prixLivre} €</p>
            </div>
            <div class="card-action">

            </div>
        </div>`
    livres.appendChild(livreItem);

    addBtnDelete(livre);
    addBtnDetails(livre);
    addBtnUpdate(livre);
}
```

AJOUTER DES LIVRES :

- Créer un formulaire avec les attributs suivant :
- Action = url du back (ajouter-livre)
- Method = POST
- Lors de la soumission, le formulaire appel url du back (Routes/livres.route.jss) qui appelle Controller (Controller/livres.controller.js) et effectue la requête http de manière asynchrone

```
<!--FORMULAIRE AJOUT-->
<form action="ajouter-livre" id="ajouterForm" method="post">
    <div class="form-group">
        <label>Nom du Livre</label>
        <input class="form-control" type="text" name="nomLivre">
    </div>

    <div class="form-group">
        <label>Description du Livre</label>
        <input class="form-control" type="text" name="descriptionLivre">
    </div>

    <div class="form-group">
        <label>Prix du Livre</label>
        <input class="form-control" type="text" name="prixLivre">
    </div>

    <div class="form-group">
        <label>Image du Livre</label>
        <input class="form-control" type="text" name="imageLivre">
    </div>

    <div class="form-group">
        <button class="col s6 waves-effect waves-light btn orange lighten-2" type="submit">Ajouter</button>
        <button type="reset" class="col s6 waves-effect waves-light btn purple lighten-2">vider les champs</button>
    </div>
    <br />
    <br />
</form>
```

LE CRUD DEPUIS NOTRE SPA :

- Créer 3 fonctions pour ajouter les boutons du CRUD (Details, Mise à jour et supprimer)

- La fonction addButtonsDelete(livre) : Créer un élément HTML <button>

- Ajouter un attribut id dynamique, du texte, des classe CSS.

- Récupérer dynamiquement le parent à l'aide de son ID

- Ajouter un événement clic qui appel la fonction deleteLivre

- Répéter l'opération pour les boutons détails et édition

```
function addBtnDelete(livre){
    //Une contante pour creer le bouton
    const btnDelete = document.createElement('button');
    //Id unique a chaque bouton
    btnDelete.setAttribute('id', `btnDelete${livre._id}`);
    //Texte du bouton
    btnDelete.innerHTML = 'Supprimer';
    //Classe css pour mise en page
    btnDelete.className = 'col s12 center btn orange';
    //Recup de id de chaque livre
    let livreDIV = document.getElementById(`livres${livre._id}`)
    livreDIV.appendChild(btnDelete);

    //Au click sur le bouton on declenche un event (une fonction)
    btnDelete.addEventListener('click', () => deleteLivre(livre))
}

function addBtnDetails(livre){
    //Une contante pour creer le bouton
    const btnDetails = document.createElement('button');
    //Id unique a chaque bouton
    btnDetails.setAttribute('id', `btnDetails${livre._id}`);
    //Texte du bouton
    btnDetails.innerHTML = 'Détails';
    //Classe css pour mise en page
    btnDetails.className = 'col s12 btn purple';
    //Recup de id de chaque livre
    let livreDIV = document.getElementById(`livres${livre._id}`)
    livreDIV.appendChild(btnDetails);

    //Au click sur le bouton on declenche un event (une fonction)
    btnDetails.addEventListener('click', () => detailsLivre(livre))
}

function addBtnUpdate(livre){
    //Une contante pour creer le bouton
    const btnUpdate = document.createElement('button');
    //Id unique a chaque bouton
    btnUpdate.setAttribute('id', `btnUpdate${livre._id}`);
    //Texte du bouton
    btnUpdate.innerHTML = 'Mise a jour';
    //Classe css pour mise en page
    btnUpdate.className = 'col s12 btn green';
    //Recup de id de chaque livre
    let livreDIV = document.getElementById(`livres${livre._id}`)
    livreDIV.appendChild(btnUpdate);

    //Au click sur le bouton on declenche un event (une fonction)
    btnUpdate.addEventListener('click', () => afficherUpdateForm(livre))
}
```

- Chaque bouton déclenche donc une fonction et donc une requête http qui communique avec le backend pour effectuer des opérations :
- Supprimer un livre :
    - Récupérer le conteneur avec un querySelector
    - Exécuter une requête Axios qui appel url du backend (méthode delete)
    - Retourner une promesse qui effectue un remove() en cas de resolve ou une erreur en cas de reject à l'aide d'un catch

```
function deleteLivre(livre){
    const livreID = document.querySelector(`#livres${livre._id}`)

    axios.delete(`http://localhost:3000/supprimer/${livre._id}`)
    .then(() => livreID.remove())
}
```

- Afficher les détails d'un livre (en 2 temps)
    - Dans une fonction : effectuer une requête http avec Axios (méthode GET) en passant l'id du livre en paramètre
    - Dans une promesse : appeler une seconde fonction en callback avec les données du livre en paramètre
    - Dans la seconde fonction :
    - Récupérer les parents HTML grâce querySelector
    - Afficher les valeurs grâce à la concaténation ES6
    -

```
function detailsLivre(livre){
    axios.get(`http://localhost:3000/livres/${livre._id}`)
    .then(livre => afficherDetailsLivres(livre.data))
}

function afficherDetailsLivres(livre){
    const livreRow = document.querySelector('#livres_row');
    const detailsLivre = document.querySelector('#detailsLivre');

    livreRow.style.display = 'none';
    detailsLivre.innerHTML =
        `
        <div class="card col s6">
          <div class="card-image">
            <img src="${livre.data.imageLivre}" alt="${livre.data.nomLivre}" title="${livre.data.nomLivre}" width="25%" />
          </div>
          <div class="card-content">
            <h3 class="blue-text lighten-1">${livre.data.nomLivre}</h3>
            <p>Référence : ${livre.data._id}</p>
            <p>Description : </p>
            <p>${livre.data.descriptionLivre}</p>
            <p class="red-text lighten-3">Prix : ${livre.data.prixLivre} €</p>
          </div>
          <div class="card-action">
                <a href="/" class="btn red lightn-2">Retour</a>
          </div>
        </div>`
}
```

LA MSIE A JOUR :

- Dans une première fonction : recréer un formulaire avec les 4 champs
- A la soumission grâce à addEventListener ('submit'), appeler une seconde fonction de traitement

- La seconde fonction de traitement :
    - Supprimer le comportement par défaut avec event.preventDefault()
    - Recréer un objet livre Data avec les nouvel valeur récupérée du formulaire a l'aide de event.currentTarget.VotreChamp.value
    - Réalisé une requête http méthode PUT qui va convertir une chaine de caractère au format json (JSON.Stringify) les valeurs du body
    - Retournée les nouvelles valeurs au format json
    - Effectuer un rafraichissement de la page
    - Appeler la fonction afficherLivre pour tous mettre à jour

```
function afficherUpdateForm(livre){
    const livreRow = document.querySelector('#livres_row');
    const updateForm = document.querySelector('#updateForm');

    livreRow.style.display = 'none';

    const editForm = document.createElement('form')
    editForm.id = 'edit-form';

    editForm.innerHTML =

      <h3 class="purple-text center lighten-1 card-panel">EDITER LE LIVRE</h3>
      <div class="form-group">
        <label>Nom du Livre</label>
        <input class="form-control" value="${livre.nomLivre}" type="text" name="nomLivre">
      </div>

      <div class="form-group">
        <label>Description du Livre</label>
        <input class="form-control" value="${livre.descriptionLivre}" type="text" name="descriptionLivre">
      </div>

      <div class="form-group">
        <label>Prix du Livre</label>
        <input class="form-control" type="text" value="${livre.prixLivre}" name="prixLivre">
      </div>

      <div class="form-group">
        <label>Image du Livre</label>
        <input class="form-control" type="text" value="${livre.imageLivre}" name="imageLivre">
      </div>

      <div class="form-group">
        <button class="col s6 waves-effect waves-light btn green lighten-2" type="submit">Mettre a jour</button>
        <button type="reset" class="col s6 waves-effect waves-light btn red lighten-2">vider les champs</button>
      </div>

    updateForm.appendChild(editForm);
    editForm.addEventListener('submit', (event) => addNewDataLivre(event, livre));
}

function addNewDataLivre(event, livre){
    event.preventDefault();

    livreDatas = {
        nomLivre: `${event.currentTarget.nomLivre.value}`,
        descriptionLivre: `${event.target.descriptionLivre.value}`,
        prixLivre: `${event.target.prixLivre.value}`,
        imageLivre: `${event.target.imageLivre.value}`,
    }
    fetch(`http://localhost:3000/edit-livres/${livre._id}`,{
        method: 'PUT',
        headers: {
            "Content-Type": "application/json"
        },
        body: JSON.stringify(livreDatas)
    })
    .then(response => response.json())
    .then(() => {
        console.log('livre est mis a jour')
        window.location.reload();
        afficherLivres(livre)
    })
}
```