



## 1 OBJECTIFS

Cette activité doit permettre :

- de pratiquer le framework Express.JS
- d'utiliser un système de base de données (MySQL pour simplifier)
- de mettre en œuvre un système C.R.U.D. (Create, Read, Update, Delete)

Cette activité est inspirée d'une vidéo sur Youtube :



<https://www.youtube.com/watch?v=4fWWn2Pe2Mk>

Par CodAffection

## 2 DÉROULEMENT

Node.JS doit être actif, un serveur MySQL doit être actif, une extension (RESTer) doit être ajoutée au navigateur :

- Chrome : <https://chrome.google.com/webstore/detail/rester/ejfoncpjfgmeleakejdcanedmefagga>
- FireFox : <https://addons.mozilla.org/en-US/firefox/addon/rester/>

### 2.1 APPLICATION LIVRE D'OR

L'application doit permettre de voir l'ensemble des messages, d'ajouter des messages et supprimer un message particulier.

action	vue	méthode
Afficher les commentaires	/livreOr	get
Afficher un commentaire par ID	/livreOr/:id	get
Supprimer un commentaire par ID	/livreOr/:id	delete
Ajouter un commentaire	/livreOr	post
Chercher un commentaire	/search/?msgSearch=	get



## 3 CRÉATION DU PROJET

### 3.1 EXPLICATIONS

*(Vous ne devez pas tenter la création en suivant ces lignes, ce sont des informations de préparation)*

Les étapes classiques de création de notre projet Node.JS sont :

1. Créer un dossier LivreOr pour l'application
2. Dans ce répertoire, lancer la commande **npm init** pour initialiser le projet, avec **index.js** comme fichier principal
3. lancer la commande **npm install express ejs --save** pour installer Express et la dépendance ejs (vues)
4. lancer la commande **npm install mysql2 --save** pour installer la dépendance MySQL
5. Créer un répertoire **views** qui contiendra les fichiers EJS d'affichage

Il faut également préparer une base MySQL :

1. Lancer le serveur MySQL (WAMP, XAMPP, ...)
2. Créer une base SLAM5 en UTF-8
3. Dans la base SLAM5, créer une table tlivre
  1. créer les champs suivant : id (int), name (varchar), message (varchar) et evaluation (int)
4. Créer un utilisateur admntlivre avec un mot de passe simple pour le projet (adm123)
5. Affecter les droits ALL sur toute la base

Les informations pour la base de données sont en annexes.



## 3.2 ÉTAPE 1 : CODE SOURCE INDEX.JS

Le fichier principal contient les éléments suivants :

```
const mysql = require('mysql')
let mysqlconnexion = mysql.createConnection( )
```

Objet JSON avec 4 paramètres :  
Host, user, password, database

```
{host : 'localhost' , user : 'admtlivre' , password : 'adm123' , database : 'SLAM5'}
```

Voici donc le premier script, incomplet mais permettant de tester le serveur et la base de données :

```
// inclure les dépendances et middlewares
const mysql = require('mysql2')
const express = require('express')
const ejs = require('ejs')
const bodyParser = require('body-parser')
const { urlencoded } = require('body-parser')

// activer les dépendances
let app = express()
let mysqlconnexion = mysql.createConnection({
  host:'localhost',
  user:'admtlivre',
  password:'adm123',
  database:'SLAM5'
})
mysqlconnexion.connect((err) => {
  if (!err) console.log('BDD connectée.')
  else console.log('BDD connexion échouée \n Erreur: '+JSON.stringify(err))
})

// activer le middleware et lancer l'application sur le port 3000
app.use(express.json())
app.use(bodyParser.urlencoded({ extended: true }));
app.listen(3000, () => console.log('le serveur Livre d\'Or est prêt.'))

// utiliser les routes
app.get('/', (req, res) => {
  res.send('Livre d\'Or est actif')
})
```

lancer le script avec la commande **nodemon index.js**

*Il n'est pas nécessaire de ré-installer nodemon puisque nous avons déjà fait une activité utilisant la commande -g pour rendre l'installation globale.*

Ouvrir le site : <http://localhost:3000/>



### 3.3 ÉTAPE 2 : GESTION CONFIGURATION

Il faut maintenant améliorer notre code et ajouter les routes nécessaires pour faire une véritable application. De plus, il n'est pas très pratique de mettre les paramètres de connexion à la base, directement dans le code source (surtout s'il y aura plusieurs fichiers). Nous allons sécuriser ceci avec un middleware appelé iniparser.

1. Installer le middleware iniparser (**npm install iniparser**)
2. créer un fichier DB.ini contenant les informations de connexion
3. modifier le fichier index.js pour utiliser ce fichier INI

#### 3.3.1 Code DB.ini

C'est un simple fichier texte qui contient ceci :

```
[dev]
host = localhost
user = admtlivre
password = adm123
database = SLAM5
```

#### 3.3.2 Code index.js (pour utiliser iniparser)

Il faut donc utiliser le middleware pour lire et extraire les informations du fichier INI. Il faut ajouter :

```
let iniparser = require('iniparser')
let configDB = iniparser.parseSync('./DB.ini')
```

Et remplacer la connexion MySQL par :

```
let mysqlconnexion = mysql.createConnection({
  host:configDB['dev']['host'],
  user:configDB['dev']['user'],
  password:configDB['dev']['password'],
  database:configDB['dev']['database']
})
```

Désormais, les données de connexion sont stockées dans le fichier INI et séparées du code.

*Astuce pour GIT : Il est préférable de mettre les fichiers ayant une extension \*.ini dans le fichier .gitignore.*



## 3.4 ÉTAPE 3 : AJOUT DES SERVICES

Voici deux méthodes, l'une affiche un seul message (par son ID), l'autre permet une recherche.

### 3.4.1 afficher tous les messages

Il suffit d'ajouter une vue Express.JS après le bloc `app.get`, comme dans le code suivant :

```
// voir tous les messages
app.get('/LivreOr', (req, res) => {
  mysqlconnexion.query('SELECT * FROM tlivre', (err, lignes, champs) => {
    if (!err) {
      console.log(lignes)
      res.send(lignes)
    }
  })
})
```

Sauvegarder et testez les deux liens suivants :

<http://localhost:3000/>

<http://localhost:3000/LivreOr>



### 3.4.2 Rechercher : un message ou les messages contenant...

Nous ajoutons 2 services différents à notre application, basé sur 2 modèles d'URI différentes :

- **PARAMS** : Afficher un message particulier (par son index), <http://localhost:3000/LivreOr/2>
- **QUERY** : Chercher les messages qui contiennent une chaîne, <http://localhost:3000/search/?msgSearch=you>

```
// voir un message choisi par ID (parameters)
app.get('/LivreOr/:id', (req, res) => {
  let critere = req.params.id
  console.log("ID = "+critere)
  mysqlconnexion.query('SELECT * FROM tlivre WHERE Id = ?', [critere], (err,
lignes, champs) => {
    if (!err) {
      console.log(lignes)
      res.send(lignes)
    }
  })
})

// chercher les messages contenant ElementSearch dans le champ message (query)
app.get('/search/', (req, res) => {
  let critere = '%'+req.query.msgSearch+'%' //query. Exemple : %you%
  console.log("Find this : "+critere)
  mysqlconnexion.query('SELECT * FROM tlivre WHERE message LIKE ?', [critere],
(err, lignes, champs) => {
    if (!err) {
      console.log(lignes)
      res.send(lignes)
    }
  })
})
```

Sauvegarder votre code et tester les pages suivantes :

<http://localhost:3000/LivreOr/2>

<http://localhost:3000/search/?msgSearch=you>

Vous constaterez que **req.params.nomParametre** utilise une URL en prenant le dernier paramètre comme valeur, tandis que **req.query.msgSearch** s'appuie sur les requêtes GET.

*Note : il existe aussi la récupération de paramètres dans une méthode POST, par **req.body.nomVar** que nous verrons plus loin (ajout).*



### 3.4.3 Effacer un message

Il s'agit ici de recopier le code permettant de choisir un message par son ID, mais de changer la méthode d'accès du navigateur : **app.get** devient **app.delete**. La requête SQL est modifiée en conséquence.

Code index.js

```
// effacer un message choisi par ID (parameters)
app.delete('/LivreOr/:id', (req, res) => {
  let critere = req.params.id
  console.log("ID = "+critere)
  mysqlconnexion.query('DELETE FROM tlivre WHERE Id = ?', [critere], (err,
lignes, champs) => {
    if (!err) {
      console.log("Effacement terminé")
      res.send("Effacement terminé")
    } else {
      console.log("Erreur lors de l'effacement")
      res.send("Erreur effacement : "+JSON.stringify(err))
    }
  })
})
```

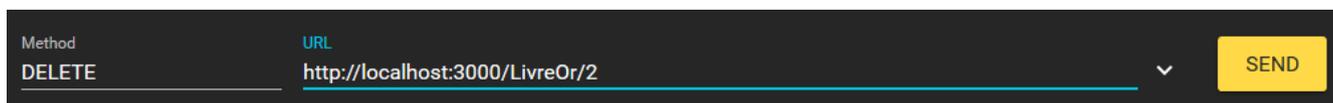
Sauvegarder.

Pour tester cette nouvelle, il faut utiliser l'extension RESTer : 

Extension Chrome	<a href="https://chrome.google.com/webstore/detail/rester/eeifoncpifgmeleakeidcanedmefagga">https://chrome.google.com/webstore/detail/rester/eeifoncpifgmeleakeidcanedmefagga</a>
Extension Firefox	<a href="https://addons.mozilla.org/fr/firefox/addon/rester/?utm_source=addons.mozilla.org&amp;utm_medium=referral&amp;utm_content=search">https://addons.mozilla.org/fr/firefox/addon/rester/?utm_source=addons.mozilla.org&amp;utm_medium=referral&amp;utm_content=search</a>

Dans RESTer, choisir la méthode DELETE

Saisir l'URL <http://localhost:3000/LivreOr/2>



Une fois validé, vérifier que la liste ne compte plus le message n°2 de KENT Clark.

<http://localhost:3000/LivreOr>



### 3.4.4 Insérer un message

Comme pour l'effacement, l'URI reste la même mais l'appel sera fait avec la méthode **POST**.

Node code utilisera donc **app.post** et La requête SQL est modifiée en conséquence.

Code index.js

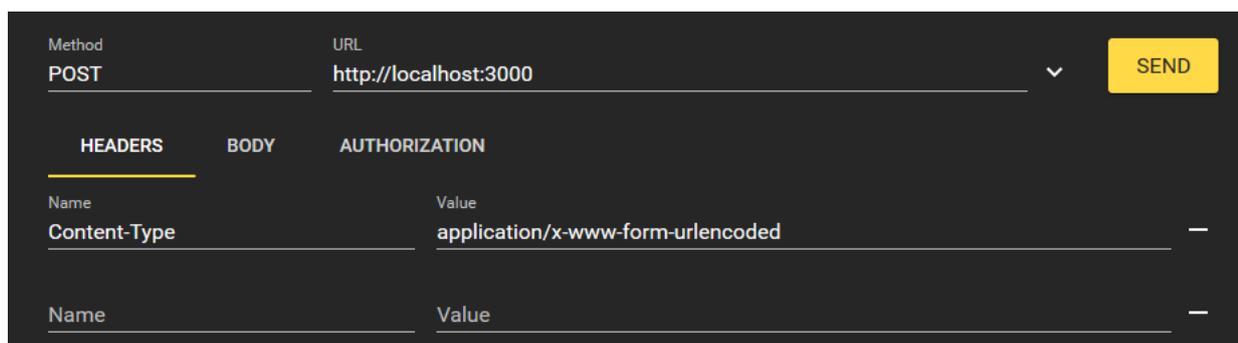
```
// ajouter un message (query) NOTE: formulaire : application/x-www-form-urlencoded
.post('/LivreOr', (req, res) => {
  let msgID = req.body.id
  let msgName = req.body.name
  let msgMsg = req.body.msg
  let msgNote = req.body.note

  console.log(`Ajout msg ID ${msgID} de ${msgName} contenant ${msgMsg} et noté $
{msgNote}`)
  let requeteSQL = "INSERT INTO tlivre (id, name, message, evaluation) VALUES"
  requeteSQL =requeteSQL + ' (' + msgID + ', "' + msgName + '", "' + msgMsg + '", '
+ msgNote + ' )'
  console.log("Requete : "+requeteSQL)
  mysqlconnexion.query( requeteSQL, (err, lignes, champs) => {
    if (!err) {
      console.log("Insertion terminé");
      res.redirect("/LivreOr");
    } else {
      console.log("Erreur lors de l'enregistrement")
      res.send("Erreur ajout : "+JSON.stringify(err))
    }
  })
})
```

Sauvegarder. Pour tester cette nouvelle fonction, il faut utiliser à nouveau l'extension Rester : 

Les données ne doivent pas être transmises dans l'URL (méthode GET) donc il faut choisir la méthode POST pour l'URL <http://localhost:3000/LivreOr>

Dans l'onglet HEADERS, utiliser le champ Content-Type et choisissez application/x-www-form-urlencoded.



The screenshot shows the Rester REST client interface. At the top, the Method is set to POST and the URL is http://localhost:3000. A yellow SEND button is visible. Below this, there are three tabs: HEADERS, BODY, and AUTHORIZATION. The HEADERS tab is selected, showing a table with columns for Name and Value. One header is defined: Content-Type with the value application/x-www-form-urlencoded. There is also an empty row for Name and Value.

Name	Value
Content-Type	application/x-www-form-urlencoded
Name	Value



Il faut ensuite créer les valeurs dans l'onglet BODY :

Cliquer sur l'onglet BODY

Cliquer sur l'icône des paramètres (3 points verticaux)

Choisir FORM

Remplir les champs (id / 2, name / KENT Clarke, msg / Yo superman !, note / 4

Method	URL	
POST	http://localhost:3000/LivreOr	SEND
HEADERS BODY AUTHORIZATION		
Name	Value	Type
id	2	Text
Name	Value	Type
name	KENT Clarke	Text
Name	Value	Type
msg	Yo superman !	Text
Name	Value	Type
note	4	Text
Name	Value	Type
		Text

Il est évident que l'extension RESTer simule un client (une application Angular.JS ou EJS + Node.JS par exemple). Cela permet cependant de développer rapidement et tester son code facilement.

**Le codage de l'application sera vu dans une autre activité : ne supprimez pas le code utilisé dans celle-ci.**



## 4 UTILISATION D'EJS

### 4.1 CRÉATION DU FORMULAIRE

Voici maintenant comment remplacer les requêtes de l'extension REXter par un formulaire valide, qui utilisera la méthode POST réelle. Contrairement à la requête écrite avec REXter, les données seront transmises dans le corps de la requête, ce qui implique l'usage de body-parser et de ses méthodes.

1. Dans le répertoire views, créer un nouveau fichier 'formulaire.ejs'
2. Dans le fichier `index.js`, ajouter la dépendance 'ejs' et activer le 'engine view' sur ejs
3. Activer une route statique sur le répertoire views
4. installer la dépendance express-validator pour la validation et la sanitisation du formulaire (`npm install express-validator --save`)
5. Ajouter deux nouvelles vues :
  1. nouvelle vue pointant sur le formulaire '/formulaire' : le fichier formulaire.ejs utilisera la method post et l'encodage `enctype="application/x-www-form-urlencoded"`
  2. nouvelle vue pointant sur le formulaire de recherche '/search'
6. Remplacer les instructions `req.query` par des `req.body` et définir body-parser sur des données encodées ()

Afin d'avoir un résultat plus propre, nous utiliserons semantic-UI :

1. Créer un répertoire **public** (et l'ajouter aux routes statiques)
2. Y copier le répertoire semantic
3. Utiliser les balises semantic-UI dans les vues EJS

Exemple de résultat :

*A noter : sous Visual Studio Code, il faut ajouter une extension pour que le code EJS soit affiché sans erreur.*

[EJS Language Support ↗](#)

**Livre d'Or**

Formulaire    Résultats

id:

nom:

message:

Evaluation:



```
// inclure les dépendances et middlewares
const mysql = require('mysql')
const express = require('express')
const ejs = require('ejs')
const iniparser = require('iniparser')
const bodyparser = require('body-parser')
const { urlencoded } = require('body-parser')

// activer les dépendances
let configDB = iniparser.parseSync('./DB.ini')
let app = express()
app.set('view engine', 'ejs')
app.use(express.static('views'))
app.use(express.static('public'))

let mysqlconnexion = mysql.createConnection({
  host:configDB['dev']['host'],
  user:configDB['dev']['user'],
  password:configDB['dev']['password'],
  database:configDB['dev']['database']
})

mysqlconnexion.connect((err) => {
  if (!err) console.log('BDD connectée.')
  else console.log('BDD connexion échouée \n Erreur: '+JSON.stringify(err))
})

// activer le middleware et lancer l'application sur le port 3000
app.use(express.json());
app.use(express.urlencoded());

app.listen(3000, () => console.log('le serveur Livre d'Or est prêt.))

// utiliser les routesrs

app.get('/', (req, res) => {
  res.send('Livre d'Or est actif')
})

// voir tous les messages
app.get('/LivreOr', (req, res) => {
  mysqlconnexion.query('SELECT * FROM tlivre', (err, lignes, champs) => {
    if (!err) {
      console.log(lignes)
      res.render("affichage", {LivreOr : lignes})
    }
  })
})
```



```
    })
  })

// chercher les messages contenant ElementSearch dans le champ message (query)
app.get('/search/', (req, res) => {
  let critere = '%'+req.query.msgSearch+'%' //query. Exemple : %you%
  console.log("Find this : "+critere)
  mysqlconnexion.query('SELECT * FROM tlivre WHERE message LIKE ?', [critere],
(err, lignes, champs) => {
  if (!err) {
    console.log(lignes)
    res.send(lignes)
  }
  })
})

// voir un message choisi par ID (parameters)
app.get('/LivreOr/:id', (req, res) => {
  let critere = req.params.id
  console.log("ID = "+critere)
  mysqlconnexion.query('SELECT * FROM tlivre WHERE Id = ?', [critere], (err,
lignes, champs) => {
  if (!err) {
    console.log(lignes)
    res.send(lignes)
  }
  })
})

// afficher le formulaire
app.get('/formulaire', (req, res) => {
  res.render('./formulaire')
})

// effacer un message choisi par ID (parameters)
app.delete('/LivreOr/:id', (req, res) => {
  let critere = req.params.id
  console.log("ID = "+critere)
  mysqlconnexion.query('DELETE FROM tlivre WHERE Id = ?;', [critere], (err,
lignes, champs) => {
  if (!err) {
    console.log("Effacement terminé")
    res.send("Effacement terminé")
  } else {
    console.log("Erreur lors de l'effacement")
    res.send("Erreur effacement : "+JSON.stringify(err))
  }
  }
})
```



```

    })
  })

  // ajouter un message (query)
  .post('/LivreOr', urlencoded, (req, res) => {
    let msgID = req.body.id
    let msgName = req.body.name
    let msgMsg = req.body.msg
    let msgNote = req.body.note

    console.log(`Ajout msg ID ${msgID} de ${msgName} contenant ${msgMsg} et noté $
{msgNote}`)
    let requeteSQL = "INSERT INTO tlivre (id, name, message, evaluation) VALUES"
    requeteSQL =requeteSQL + ' (' + msgID + ', "' + msgName + '", "' + msgMsg + '", '
+ msgNote + ' )'
    console.log("Requete : "+requeteSQL)
    mysqlconnexion.query( requeteSQL, (err, lignes, champs) => {
      if (!err) {
        console.log("Insertion terminé");
        res.redirect("/LivreOr");
      } else {
        console.log("Erreur lors de l'enregistrement")
        res.send("Erreur ajout : "+JSON.stringify(err))
      }
    }
  })
})

```

Ajoutons ensuite la vue du formulaire

Formulaire.ejs

```

<%- include('header') %>

<div class="ui main container">
  <form action="/LivreOr" method="POST" enctype="application/x-www-form-
urlencoded" class="ui form">
    <div class="field">
      <label for="id">id : </label>
      <input type="text" name="id" id="id" value="11">
      <label for="nom">nom : </label>
      <input type="text" name="name" id="nom" value="ROUMANET">
      <label for="message">message : </label>
      <textarea name="msg" id="msg">Test</textarea>
      <label for="note">Evaluation : </label>
      <input type="text" name="note" id="note" value="4">
    </div>
    <button type="submit" class="ui blue animated submit large button">
      <div class="visible content">Envoyer</div>

```



```
        <div class="hidden content">
            <i class="right arrow icon"></i>
        </div>
    </button>
</form>
</div>
<%- include('footer') %>
```

Note : les notations `<%=` et `<%-` se ressemblent mais fonctionnent différemment.

- La balise `<%=` sort le modèle en échappant le code HTML
- La balise `<%-` sort le modèle sans échappement
- La balise `<%#` permet de mettre en commentaire la section EJS

Les informations sur le fonctionnement d'EJS sont disponibles dans sa documentation <https://ejs.co/#docs>

Maintenant, créons l'entête et le pied de page HTML dans les fichiers header.ejs et footer.ejs

header.ejs

```
<!DOCTYPE html>
<html lang="fr">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Livre d'or (SLAM5)</title>
  <link rel="stylesheet" href="/semantic/semantic.min.css">
</head>
<body>
  <div class="ui inverted segment">
    <div class="ui inverted">
      <h1>Livre d'Or</h1>
    </div>
    <div class="ui inverted divider"></div>
    <div class="ui inverted menu">
      <a href="/formulaire" class="header item">Formulaire</a>
      <a href="/LivreOr" class="header item">Résultats</a>
    </div>
  </div>
```

footer.ejs

```
  <div class="ui inverted segment">
    <p>auteur : D. Roumanet</p>
  </div>
</body>
</html>
```



#### Note

L'intégration de frameworks CSS ne fonctionne pas toujours bien, vous pouvez remplacer la ligne

```
<link rel="stylesheet" href="/semantic/semantic.min.css">
```

par son accès CDN :

```
<link rel="stylesheet"
href="https://cdnjs.cloudflare.com/ajax/libs/semantic-ui/2.5.0/semantic.min.css">
```

## 4.2 CRÉATION DE LA VUE DYNAMIQUE

La vue portera le nom d'affichage.ejs et affiche la liste des messages du livre d'Or.

Nous créons une vue "affichage" qui affiche les messages.

affichage.ejs

```
<%- include('header') %>
<h1>Affichage : </h1>
<div class="ui main container">
  <ul> <% LivreOr.forEach(function(user) { %>
    <li>
      <a href="http://localhost:3000/LivreOr/<%=user.id%" >&#10006;</a>
      <strong> <%= user.name %> </strong> a dit <%= user.message %>
    </li>
    <% } ) %>
  </ul>
</div>
<%- include('footer') %>
```

Début de la fonction

Fin de la balise EJS

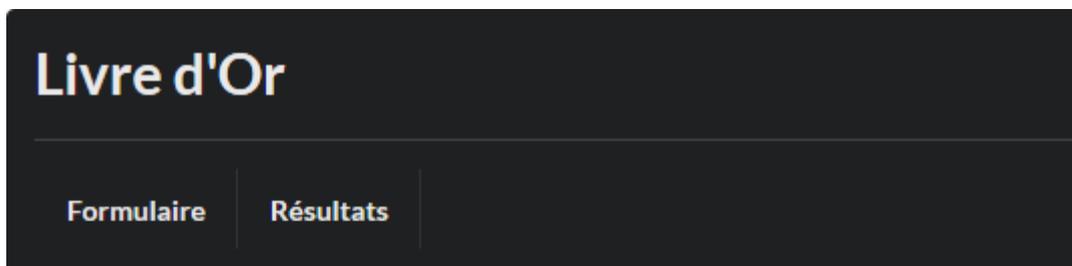
Fin de la fonction

Cette vue agit comme suit :

- inclure un entête EJS dont le nom est header.ejs : `<%- include('header') %>`.
- pour l'objet 'LivreOr' transmis on parcourt la liste en récupérant l'objet courant dans 'user' avec la fonction `<% LivreOr.forEach(function(user){ %>`
- affiche la propriété 'id' de l'objet courant avec `<%=user.id%>`
- ...
- inclure un fichier de fin, footer.ejs : `<%- include('footer') %>`



Si tout fonctionne correctement, vous devriez obtenir une vue similaire ;



## Affichage :

- ✕ NORRIS Chuck a dit Best web site David ;)
- ✕ PARR Hélène a dit Hey, nothing about girl in computers science!!!
- ✕ DE LA VEGA Don Diego a dit sometime Zorro? Your website is zero
- ✕ M. BEANS a dit Are you serious? Do not play with security
- ✕ KENT Clark a dit yo Superman
- ✕ KENT Clark a dit yo Superman
- ✕ ROUMANET a dit Test
- ✕ ROUMANET a dit Test multipart/form-data

auteur : D. Roumanet

Chaque croix dispose d'un lien personnalisé pour voir les données de la ligne sélectionnée.

**Attention** : il n'est pas possible de faire un appel à la méthode DELETE dans une simple URL. Si nous voulons supprimer la ligne avec la croix, il faut créer une nouvelle vue, par exemple <http://localhost:3000/LivreOr/delete/>



### 4.3 ÉTAPE 5 : LISTE ET CHOIX

Pour aller plus loin, voici quelques modifications simples : elles vous permettront de vous approcher d'une application C.R.U.D. (il manque la mise à jour de données, UPDATE).

#### 4.3.1 Effacement par méthode GET

Il faut remplacer la route d'effacement par la fonction suivante :

```
// effacer un message choisi par ID (parameters)
app.get('/LivreOr/delete/:id', (req, res) => {
  let critere = req.params.id
  console.log("ID = "+critere)
  mysqlconnexion.query('DELETE FROM tlivre WHERE Id = ?;', [critere], (err,
lignes, champs) => {
    if (!err) {
      console.log("Effacement terminé")
      res.redirect("/LivreOr")
    } else {
      console.log("Erreur lors de l'effacement")
      res.send("Erreur effacement : "+JSON.stringify(err))
    }
  })
})
```

la redirection renvoie vers la page principale du site (celle qui affiche la liste).

#### 4.3.2 Options d'affichage de modification et suppression dans la vue

Il suffit de créer les liens et les symboles utiles dans la boucle de la vue 'affichage' :

```
<%- include('header') %>
<h1>Affichage : </h1>
<div class="ui main container">
  <ul> <% LivreOr.forEach(function(user){ %>
    <li>
      <a href="http://localhost:3000/LivreOr/<%=user.id%" >#128065;</a>
      <a href="http://localhost:3000/LivreOr/update/<%=user.id%"
>#9998;</a>
      <a href="http://localhost:3000/LivreOr/delete/<%=user.id%"
>#10006;</a>
      <strong> <%= user.name %> </strong> a dit <%= user.message %>
    </li>
  <% })%>
  </ul>
</div>
<%- include('footer') %>
```



La vue doit normalement afficher les icônes suivantes :

## Affichage :

-   X NORRIS Chuck a dit Best web site David ;)
-   X PARR Hélène a dit Hey, nothing about girl in computers science!!!
-   X DE LA VEGA Don Diego a dit sometime Zorro? Your website is zero
-   X M. BEANS a dit Are you serious? Do not play with security
-   X KENT Clark a dit yo Superman
-   X ROUMANET a dit Test
-   X ROUMANET a dit Test multipart/form-data

Chacune d'entre elle est cliquable et dispose d'un lien propre.

### 4.3.3 Améliorations possibles

- Il serait intéressant d'ajouter la gestion des rôles, par l'utilisation d'un cookie et d'une table utilisateurs.
- Il faudrait aussi afficher un message d'avertissement avant l'effacement d'une donnée.
- La fonction et la vue pour les modifications ne sont pas réalisées.
- Etc.



## 4.4 BASE DE DONNÉES D'ESSAI

Voici les requêtes pour générer l'exemple :

```
CREATE DATABASE SLAM5 CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci;
USE SLAM5;

CREATE TABLE `tlivre` (
  `id` int(11) NOT NULL,
  `name` varchar(50) NOT NULL,
  `message` varchar(300) NOT NULL,
  `evaluation` int(1)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

--
-- Contenu de la table `matable`
--

INSERT INTO `tlivre` (`id`, `name`, `message`, `evaluation`) VALUES
(1, 'NORRIS Chuck', 'Best web site David ;)', 5),
(2, 'KENT Clark', 'Need informations about Batman please', 4),
(3, 'PARR Hélène', 'Hey, nothing about girl in computers science!!!', 0),
(4, 'DE LA VEGA Don Diego', 'sometime Zorro? Your website is zero', 0),
(5, 'M. BEANS', 'Are you serious? Do not play with security', 2);

CREATE USER 'admtlivre'@'localhost' IDENTIFIED BY 'adm123';
GRANT ALL PRIVILEGES ON * . * TO 'admtlivre'@'localhost';
```

Notez les paramètres pour l'utilisateur :

user : admtlivre

password : adm123



Par sécurité, il faut toujours utiliser un utilisateur différent de root.

## 4.5 SOURCES

req.body vs req.query : <https://dev.to/sidthesloth92/understanding-html-form-encoding-url-encoded-and-multipart-forms-3lpa>



## 5 RÉVISION

14/09/2021	Body-barser déprécié, modification pour utiliser le middleware express : app.use(express.json()); app.use(express.urlencoded());
17/09/2021	Correction en début de document + formulaire.js (pb de < %- include('') ) + urlencoded (POST)
18/10/2022	Correction urlencoded + < %- pour les EJS + bodyparser + simple quote pour 'footer'
13/10/2023	Correction bodyparser et bodyParser