I. Introduction

Au cours des précédentes semaine, nous avons réalisé la partie frontend de notre application
Maintenant il est temps de s'intéresser au backend.
Dans cette partie vous allez créer une api pour gérer vos films favoris
Les données seront stockées dans une base de données mongodb sur le cloud

II. Express.js

Express.js, parfois aussi appelé « Express », est un **framework backend** Node.js minimaliste et rapide qui offre des fonctionnalités et des outils robustes pour développer des applications **backend** évolutives. Il permet de gérer aisément le routage et offre de nombreuses fonctionnalités supplémentaires à l'objet HTTP utilisé avec node.js. Nous allons nous en servir pour développer des applications jouant un rôle de serveur de données pour nos applications React.js

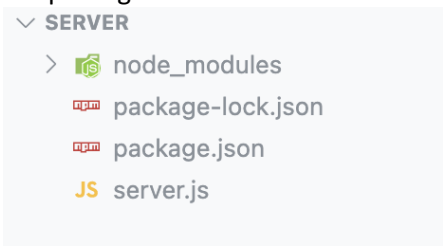
A. Initialisation du projet

Dans le dossier du projet filmotheque, nous avons précédemment créé un dossier frontend

- Placez-vous à la racine du projet
- Créez-y un dossier backend
- Lancez votre éditeur de code préféré dans le dossier backend

1. Commandes utiles

- La commande **npm init -y** initialise le projet et génère la version initiale du fichier package.json qui contiendra nos packages
- La commande **npm install express** ajoutera le package express à votre application.
- Rappel : la commande **npm i** (raccourci de install), sans précision du package permet d'installer l'ensemble des packages qui apparaissent dans l'entrée dependencies du fichier package.json.
- Les packages installés se trouvent dans le dossier **node_modules**



- Exécutez les commandes

```
npm init
```

package name: filmotheque-backend
version: (1.0.0)
description: Api de gestion des favoris
author : XXXX
entry point: (index.js) server.js

NB: Dans author entrez votre adresse mail @unilim

```
npm install express
```

- Créez un fichier **server.js** contentant le code suivant
- Copiez le code suivant dans votre fichier **server.js** :

```
const express = require("express");

//instanciation d'express
const app = express();

//definition du port par défaut
const port = process.env.PORT || 4500;

//gestion des routes
app.get("/", (req, res) => {
  res.set("Content-Type", "text/html");
  res.send("Hello world !!");
});

app.listen(port, () => {
  console.log("Server app listening on port " + port);
});
```

- Pour démarrer le serveur exécutez la commande suivante dans le terminal:

node server.js.

Le message « Server app listening on port 500O » ,s'affiche ? ➔ Votre serveur fonctionne

Entrez l'URL suivante dans votre navigateur web : http://localhost:4500

Le message Hello world !! apparaît ? ➔ Votre serveur fonctionne.

- La commande res.set permet de fixer la valeur d'un champ de l'entête HTTP transmis au client.
- La commande res.send permet de transmettre une réponse au client.

B. Rappel sur le protocole http

Le protocole HTTP définit le mode de communication entre le client (votre navigateur) et le serveur (votre application *Express*). Il définit le format des requêtes pouvant être émises par le client. Elles peuvent être décomposées en deux éléments :

- le verbe, encore appelé *méthode*. Ici notre client n'utilisera que les verbes suivants :
  - **POST** : pour envoyer des données au serveur.
  - **PUT/PATCH** : pour modifier ou remplacer des données stockées sur le serveur.
  - **GET** : pour obtenir des données du serveur.
  - **DELETE** : pour supprimer des données stockées sur le serveur.
- l'*URI*
  - identifie le serveur en précisant le *domaine*.
  - indique le *port* utilisé : *80* par défaut.
  - précise la méthode utilisée : *GET* par défaut.
  - donne le *chemin* associé au verbe : / par défaut.

Exemples

- L'*URL* http://localhost:4500 définit une requête *GET*. localhost est le *domaine*, :4500 donne le port utilisé. Aucune autre information n'étant spécifiée, le chemin par défaut est /.
- L'*URL* http://localhost:4500/xx transmet une requête *GET* avec le chemin /xx.

III. Routage côté serveur

- Arrêtez votre application (ctrl-C).
- Saisissez la commande suivante : npm install cors morgan nodemon
- Sauvegardez le fichier server.js en server0.js
- Modifiez le fichier « server.js » comme suit :

```
const express = require("express");
//on délègue la gestion des routes à un fichier de
const router = require("./router");
const cors = require("cors");
const morgan = require("morgan");
const bodyParser = require("body-parser");
const app = express();


const port = 4500;


app.use(morgan("combined"));
app.use(cors());
app.use(bodyParser.json());
app.use(bodyParser.urlencoded({extended: true}));
app.use(router); // Requests processing will be defined in the file router
app.listen(port, () => console.log("Server app listening on port " + port));
```

Explications :

- Le package **morgan** permet de définir les informations que le serveur affiche dans la console à chaque fois qu'il reçoit une requête HTTP. Ce package est particulièrement utile en phase de développement de votre serveur.
- Le package **cors** permet de configurer la façon dont des applications web définies sur un autre domaine peuvent accéder aux ressources de votre serveur. Ce mécanisme est appelé CORS pour Cross-Origin Resource Sharing, d'où le nom de ce package. Faire appel à ce package sans lui passer d'arguments permet d'autoriser tous les accès à votre ressource. Pour des exemples d'utilisation plus élaborés, vous pouvez consulter cette page.
- Le package **body-parser** permet de décomposer les requêtes HTTP POST, PATCH, etc. afin de pouvoir extraire les informations transmises dans des formulaires. Ces informations apparaissent dans le champ **req.body**.
- L'instruction **app.use(router)** permet d'utiliser les routes définies dans le fichier router.js.
- **Nodemon** est un utilitaire qui relance automatiquement le serveur à chaque modification.

- Modifiez l'entrée scripts de votre fichier package.json comme suit . Ajouter dans la section « scripts » la ligne

```
"start": "nodemon server.js"
```

- **npm start** déclenchera donc l'exécution de la commande **nodemon server.js**.

- Créez un fichier « router.js » qui contient le code suivant

```
const express = require("express");
const router = express.Router();

router.get("/", (req, res) => {
  res.send("Bienvenue sur la page d'accueil");
});

module.exports = router;
```

Explications :

Ce code se compose de plusieurs parties qu'il convient de bien comprendre :

- **.get** est une méthode de l'objet router qui permet de répondre aux requêtes GET. Comme les autres méthodes permettant de répondre aux différentes requêtes (POST, PUT, …), cette méthode comprend deux arguments.
  - "/" est le premier argument de la méthode **router.get**, il définit le chemin auquel le routeur réagit.
  - (req,res) => {...} est le deuxième argument. Il donne la fonction middleware qui sera déclenchée par le routeur lorsque qu'une requête HTTP constituée de la bonne méthode et du bon chemin sera envoyée au serveur. Cette fonction comprend deux arguments :
    - L'objet **req** (request) est utilisé pour représenter la requête HTTP entrante,
    - L'objet **res** (response) est utilisé pour envoyer la réponse http au client

A. L'objet request

- **req.params**: Contient des propriétés définies dans la partie de l'URL de la route.
- **req.query**: Contient les paramètres de requête de l'URL.
- **req.body**: Contient les données envoyées dans le corps de la requête. Pour accéder à **req.body**, vous devez utiliser un body-parser middleware.
- **req.headers**: Contient les en-têtes HTTP de la requête.
- **req.cookies**: Contient les cookies envoyés par le client.
- **req.method**: Contient la méthode HTTP de la requête (GET, POST, PUT, DELETE, etc.).
- **req.path**: Contient le chemin de l'URL demandée.
- **req.url**: Contient l'URL demandée.
- **req.hostname**: Contient le nom d'hôte de l'URL demandée.
- **req.protocol**: Contient le protocole de la requête (HTTP ou HTTPS).
- **req.ip**: Contient l'adresse IP du client.

B. L'objet response

- **res.send()**: Envoie une réponse HTTP au client. Cette méthode peut envoyer divers types de données (texte, JSON, HTML, etc.).
- **res.sendFile**() : Envoie un fichier
- **res.json()**: Envoie une réponse JSON au client.
- **res.render()**: Rend un modèle (template) avec les données spécifiées et l'envoie au client.
- **res.status()**: Définit le code d'état HTTP de la réponse.
- **res.redirect()**: Redirige l'utilisateur vers une autre URL.
- **res.cookie()**: Définit un cookie dans la réponse.
- **res.clearCookie()**: Efface un cookie.
- **res.setHeader() / res.header()**: Définit un en-tête de réponse.
- **res.get()**: Récupère la valeur d'un en-tête de requête.
- **res.locals()**: Un objet contenant des variables locales accessibles dans les modèles lors du rendu.
- **res.download()**: Déclenche le téléchargement d'un fichier.
- **res.end()**: Termine le processus de réponse sans envoyer de données.

IV. Route inexistante

- Ouvrez la page http://localhost:4500/accueil. Que se passe-t-il ?
- Ajoutez la méthode suivante au routeur afin que celui-ci retourne le statut HTTP approprié au client :

```
router.use((req, res) => {
  res.status(404);
  res.json({ error: "Page not found"});
});
```

Le terme use signifie que la *fonction middleware* est exécutée quelle que soit la méthode *HTTP* utilisée. Ici, le premier argument (le *chemin*) n'est pas précisé. En conséquence, cette méthode sera exécutée systématiquement.

Elle doit donc être la dernière de la liste . Si aucune méthode du routeur n'a déclenché sa *fonction middleware*, alors le routeur retourne le message **Page not *found*** avec un statut d'erreur ***404***.

Le serveur doit respecter les codes de réponse HTTP, ce qui permet au client de réagir correctement en fonction du code qu'il reçoit.

V. Récupération des données des routes

Le client transmet des données au serveur pour que ce dernier exécute un traitement
Ces données peuvent être transmisses via la chaine d'url

Une URL se compose de différents fragments dont certains sont obligatoires et d'autres optionnels. Pour commencer, voyons les parties les plus importantes d'une URL :

http://www.exemple.com:80/chemin/vers/monfichier.html?clé1=valeur1&clé2=valeur2#QuelquePartDansLeDocument

http:// correspond au protocole. Ce fragment indique au navigateur le protocole qui doit être utilisé pour récupérer le contenu. Généralement, ce protocole sera HTTP ou sa version sécurisée : HTTPS

www.exemple.com correspond au nom de domaine. Il indique le serveur web auquel le navigateur s'adresse pour échanger le contenu. À la place du nom de domaine, on peut utiliser une adresse IP, ce qui sera moins pratique (et qui est donc moins utilisé sur le Web).

:80 correspond au port utilisé sur le serveur web. Il indique la « porte » technique à utiliser pour accéder aux ressources du serveur. Généralement, ce fragment est absent car le navigateur utilise les ports standards associés aux protocoles (80 pour HTTP, 443 pour HTTPS). Si le port utilisé par le serveur n'est pas celui par défaut, il faudra l'indiquer.

/chemin/vers/monfichier.html est le chemin, sur le serveur web, vers la ressource. Aux débuts du Web, ce chemin correspondait souvent à un chemin « physique » existant sur le serveur. De nos jours, ce chemin n'est qu'une abstraction qui est gérée par le serveur web, il ne correspond plus à une réalité « physique ».

?clé1=valeur1&clé2=valeur2 sont des paramètres supplémentaires fournis au serveur web. Ces paramètres sont construits sous la forme d'une liste de paires de clé/valeur dont chaque élément est séparé par une esperluette (&). Le serveur web pourra utiliser ces paramètres pour effectuer des actions supplémentaires avant d'envoyer la ressource. Chaque serveur web possède ses propres règles quant aux paramètres. Afin de les connaître, le mieux est de demander au propriétaire du serveur.

Dans notre serveur nous allons pouvoir récupérer la ressource et les paramètres

url	route	récupération
http://localhost:4500/favorite/115	/favorite/:attribut	**req.params**.attribut ➔115
http://localhost:4500/favorite/?attribut=115	/favorite/	**req.query**.attribut ➔115

NB : si les données proviennent d'un formulaire elles sont à récupérer via req.**body**

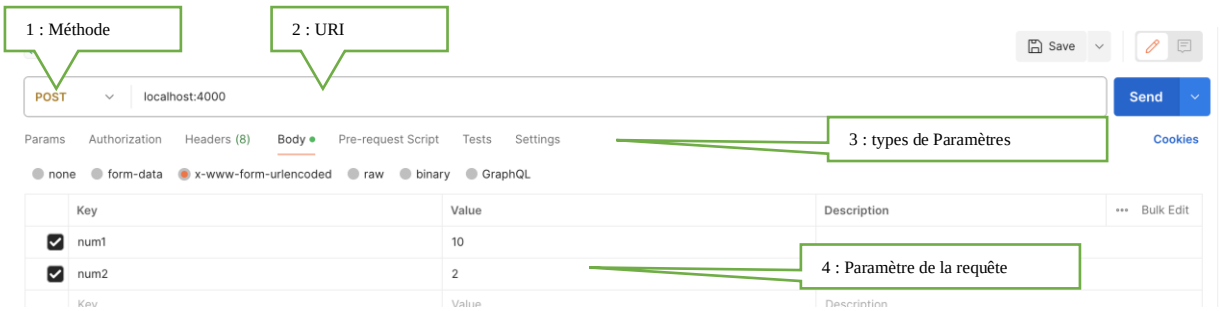
- Enregistrez le fichier « db.js » fourni dans votre dossier backend
- Remplacez le contenu du fichier « router.js » pour intercepter les routes

```
const express = require("express");
const {findFavoriteByAny,findFavoriteById,  findFavoriteByUser, findFavoriteByMovie, addFavorite,
updateFavorite,deleteFavorite} = require("./db.js");

const router = express.Router();
router.get("/favorite/", async(req, res) => {console.log("tous les favoris");});

router.get("/favorite/:val", async(req, res) => {  console.log("un favori par id :val");});

router.post("/favorite/", async(req, res) => {console.log("ajout d'un favori ");});

router.patch("/favorite/:val", async(req, res) => {console.log("modification d'un favori:val");});

router.delete("/favorite/:val", async(req, res) => {console.log("suppression d'un favori :val");});

router.use((req, res) => {
  res.status(404);
  res.json({
    error: `${req.method + ":" + req.originalUrl} Page not found`,
  });
});

module.exports = router;
```

VI. Utilisation de notre api avec postman

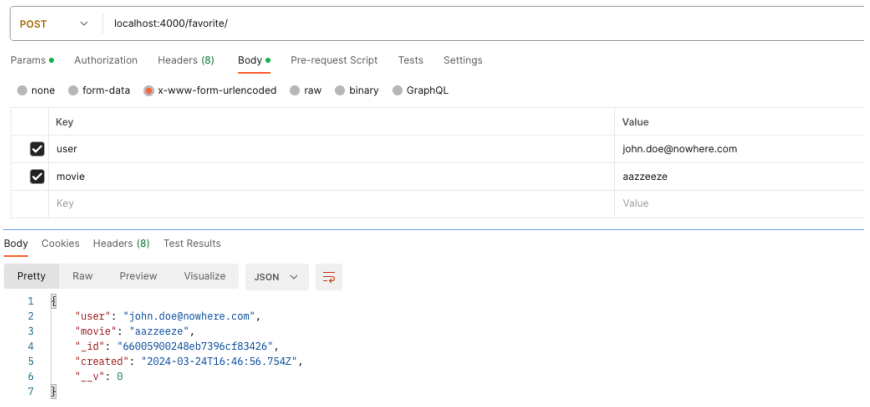
Postman, comme thunderclient , ou cUrl, est un client http qui permet de tester les requêtes http en dehors du navigateur.
L'intérêt est pouvoir utilisez les autres méthodes que GET

- Dans postman créez une nouvelle requête

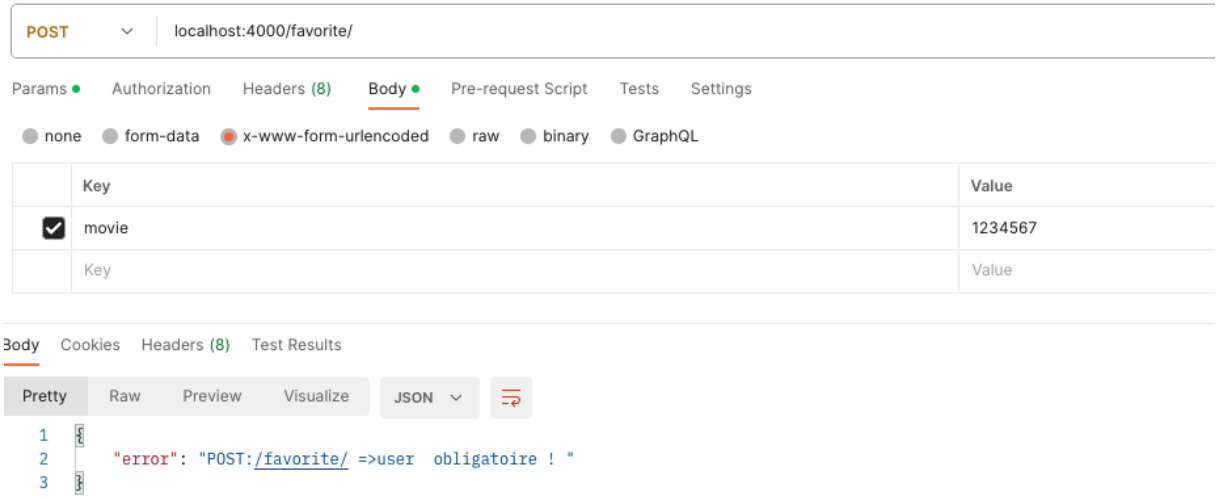


Le serveur construit précédemment respecte, à minima les recommandations d'une API REST. Nous n'étudions pas cette API en détail, néanmoins, le fait de respecter les bases nous permet des créer des interfaces que tout client peut utiliser. Pour plus de détails sur les API REST, vous pouvez vous référer à cette page .Les réponses obtenues respectant le format REST, n'importe quelle application peut interroger le serveur et exploiter les résultats obtenus.

- Testez les routes avec postman, le résultat s'affiche dans le terminal
- Modifiez le code pour renvoyer le message au client (postman) plutôt que dans la console
- Modifiez le code pour récupérer les données et les afficher

Verbe	Chemin	Action
GET	/12	get 12
POST	/	

- Modifier le code pour afficher un message d'erreur si tous les champs user et movie ne sont pas renseignés pour les requêtes de type POST et PATCH


POST
localhost:4000/favorite/
Params
Authorization
Headers (8)
Body
Pre-request Script
Tests
Settings
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
Key
Value
movie
1234567
Key
Value

Body
Cookies
Headers (8)
Test Results
Pretty
Raw
Preview
Visualize
JSON
1 {
2 "error": "POST:/favorite/ =>user obligatoire ! "
3 }

VII. Accès aux données

NB : Vous allez tous utiliser la même base de données, donc les favoris sont communs à toute la promo.
Par conséquent, évitez de faire n'importe quoi, comme supprimer les favoris qui ne vous appartiennent pas.

- Les fonctions à utiliser pour lire et manipuler les favoris sont fournies et appelées dans le routeur. Vous n'avez juste qu'à les appeler depuis vos actions, pour chaque route et retourner les données **au format json**

Actions	URL	verbe	Entrée	Fonction
Tous les favoris	http://localhost:4500/favorite/	GET	/	findFavoriteByAny()
favoris par utilisateur	http://localhost:4500/favorite/?author=aa			findFavoriteByUser (aa)
favoris par Film	http://localhost:4500/favorite/?movie=bb			findFavoriteByMovie(bb)
favoris par Auteur et Film	http://localhost:4500/favorite/?author=aa&movie=bb			findFavoriteByAny({user :'aa',movie : 'bb'})
favoris par Id	http://localhost:4500/favorite/xx		/favorite/:val	findFavoriteById(xx)
Ajout d'un favori	http://localhost:4500/favorite/	POST	/favorite/	addFavorite (aa,bb)
Modification d'un favori à partir de son id	http://localhost:4500/favorite/xx	PATCH	/favorite/:val	updateFavorite (xx,aa,bb)
Suppression d'un favori à partir de son id	http://localhost:4500/favorite/xx	DELETE	/favorite/:val	deleteFavorite (xx)

NB : Les fonctions sont **asynchrones**, il convient de les utiliser à bon escient ;-)
NB : Pensez à contrôler les actions et leurs résultats via postman,
NB : Ne pas oublier d'enregistrer votre mail dans le champ user

Prochaine étape, à partir du frontend :

- Création d'un composant Favorite, accessible depis la route /favorites qui sera chargé , en faisant appel à votre api , de :
  - Consulter la liste de vos favoris, et seulement les vôtres,
  - Rechercher un film et pouvoir l'ajouter dans vos favoris