Créer une app web en nodeJS

Exercice 1 : Mettre en place le projet

Etape 1 : Initialiser le projet

Cet exercice sera aussi présenté en live coding

- Créer un dossier et l'ouvrir dans Visual Studio Code
- Initialiser le projet nodeJS avec npm

npm init -y. Il est important de se mettre à la racine de notre projet !

- Installer les dépendances nécessaires

Dépendances basiques : bcryptjs, connect-flash, ejs, express, express-ejs-layouts, express-session, mongoose, passport, passport-local.

Dépendances dev : nodemon

- Dans le fichier package.json, mettre en place les scripts : *start* et *dev*
- Créer le fichier *index.js* qui lance le serveur sur le port 8080

Un exemple du fichier index.js incomplet est à votre disposition

- Lancer l'application

Une fois l'application lancée, rendez-vous sur localhost:8080, si tout fonctionne la page devrait afficher le message d'erreur suivant : Cannot GET /

Etape 2 : Mettre en place nos premières routes

- Créer un dossier routes/ dans la racine
- Dans ce dossier, créer les fichier *basic.js* et *user.js*
- Dans le fichier basic.js, mettez en place une gestion simple de la route principale (localhost:8080/) afin qu'elle affiche « Bonjour ».

Il faudra aussi modifier le fichier index.js !
Les fichiers basic.js et index.js (à jour) incomplet sont mis à votre disposition.

- Dans le fichier user.js, mettez en place une gestion simple des routes user (localhost:8080/user/login & localhost:8080/user/register) afin qu'elles affichent respectivement « Connection » et « Inscription ».

Il faudra de nouveau modifier le fichier index.js !
Le fichier user.js incomplet est mis à votre disposition.

- Tester les routes

A ce point, si vous lancez l'application, les routes suivantes devraient vous afficher un petit mot :

localhost:8080/user/login
localhost:8080/user/register
localhost:8080/

Etape 3 : Mettre en place notre système de layout

- Créer un dossier *views* qui contiendra tous nos fichiers ejs (html/css)

- Dans ce dossier, créer un fichier layout.ejs qui sera le layout utilisé par notre app

Il faudra modifier votre fichier *index.js*
Les fichiers *layout.ejs et index.js* incomplet sont mis à votre disposition.
Vos balises html, head, body ainsi que tous vos scripts et vos références doivent être présent dans le layout.
Voici les scripts que nous utiliserons :

```
<script src="https://code.jquery.com/jquery-3.2.1.slim.min.js"
        integrity="sha384-
KJ3o2DKtIkvYIK3UENzmM7KCkRr/rE9/Qpg6aAZGJwFDMVNA/GpGFF93hXpG5KkN"
        crossorigin="anonymous"></script>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.12.9/umd/popp
er.min.js"
        integrity="sha384-
ApNbgh9B+Y1QKtv3Rn7W3mgPxhU9K/ScQsAP7hUibX39j7fakFPskvXusvfa0b4Q"
        crossorigin="anonymous"></script>
    <script src="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/js/bootstrap.min
.js"
        integrity="sha384-
JZR6Spejh4U02d8jOt6vLEHfe/JQGiRRSQQxSfFWpi1MquVdAyjUar5+76PVCmYl"
        crossorigin="anonymous"></script>
```

Et voici les références :

```
<link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/css/
bootstrap.min.css"
    integrity="sha384-
Gn5384xqQ1aoWXA+058RXPxPg6fy4IWvTNh0E263XmFcJlSAwiGgFAW/dAiS6JXm" crossorigin="an
onymous">
    <link href="https://stackpath.bootstrapcdn.com/font-awesome/4.7.0/css/font-
awesome.min.css" rel="stylesheet"
    integrity="sha384-
wvfXpqpZZVQGK6TAh5PVlGOfQNHSoD2xbE+QkPxCAFlNEevoEH3Sl0sibVcOQVnN" crossorigin="an
onymous">
```

Cet exercice sera aussi présenté en live coding

- Maintenant que le layout est fini, il est temps de l'utiliser dans nos routes.

Prenons la vue basique (/).
Au lieu de send le message « welcome » nous voulons afficher une vue personnalisée.
Changeons donc la ligne

```
router.get('/', (req, res) => res.send('Welcome'));
```

en

```
router.get('/', (req, res) => res.render('home'));
```

Cela veut dire que lorsque la route / est appelée, l'application affichera le contenu du fichier home.ejs.
Il ne reste plus qu'à créer le fichier home.ejs et le remplir.
Une simple ligne suffit pour tester. Essayez celle-ci :

```
<h1>Welcome</h1>
```

Maintenant, si vous lancez l'application et vous vous rendez sur localhost:8080/ , le message Welcome apparaitra en titre.

Exercice 2 : Faire le front de votre page d'accueil

Pour vous familiariser avec le HTML/Css, rien de mieux que de créer votre page d'accueil par vous-même.

https://getbootstrap.com/docs/4.0/components/buttons/

Voici le liens de la doc de bootstrap, le framework que nous utilisons.

Pour la page d'acceuil on vous demande de mettre en place :

- Une barre de navigation (navbar) comportant : un titre, deux boutons et un menu déroulant.
- Un titre pour votre page avec un petit texte de description bien centrée en dessous.
- Deux rangées de photos
- Deux boutons qui seront utilisé pour se rendre vers le système de login/register (exercice 4)
- Un bouton qui sera utilisé pour se rendre vers notre page d'api (exercice 3)
- Un footer

Si vous avez des questions n'hésitez pas. Pensez à bien regarder les docs de boostrap et d'html/css.

Si vous êtes déjà habitués au html/css, tentez de nouvelles choses comme des animations de transition, des sliders ou des éléments cachés.

Exercice 3 : L'utilisation d'API basiques

Dans cet exercice, nous vous demanderons de mettre en place des appels vers des APIs basiques. Nous avons sélectionné 3 APIs plutôt simples d'utilisation. Mais vous pouvez utilisez n'importante quelle API qui existe.

API n°1 : Météo

Comme API météo nous utiliserons Openweathermap. Cette API est gratuite, il faut cependant créer un compte et valider une clé unique. Cela pouvant prendre du temps, voici ma clé que vous pouvez utiliser (pas trop, elle a une limite et il en faut pour tout le monde) :

e4da0c54723866781051130c9ee01896

Pour commencer, dans votre fichier ejs, créez deux zones de texte, une zone d'input ainsi qu'un bouton qui appellera le script *weather()* quand il est pressé.
Pensez à donner des id uniques aux zones de texte et d'input car nous les modifierons dans le script.
L'input devra contenir le nom précis d'une Ville.

Maintenant que le front est en place, passons au script Javascript.
Commencez par importer le module *http.*
Ensuite, déclarez la fonction *weather()*, elle ne prend rien en paramètre.

Maintenant, grâce aux id unique que vous avez utilisé, récupérez vos deux zones de texte et votre input dans le script.
Il ne reste plus qu'à faire la requête http. Pour cela nous utiliserons XMLHttpRequest.
Pour cela, rien de compliquer. Il faut commencer par créer une variable comme ceci.

```
var call = new XMLHttpRequest();
```

Maintenant, nous devons définir les propriétés de notre requête. Nous utiliserons les propriétés suivantes :

```
req.responseType = '';
req.open(type, url, bool);
req.onload  = function() {};
```

responseType correspond au format de réponse que nous recevrons (text ou json par exemple).
open() est la fonction qui permet d'instancier la requête. Dans notre cas, elle prend 3 paramètre :

- Type -> le type de requête que nous effectuons (GET, POST, ect …)
- url -> l'url de la requête http
- bool -> true si la requête est asynchrone (dans notre cas true) .

onload correspond à la fonction qui sera exécutée lorsque nous recevrons une réponse.

Enfin, on envoie la requête avec :

```
req.send(null);
```

Pour finir, il ne reste plus qu'a faire la fonction qui correspond à onload. La réponse de la requête se situe dans

```
req.response
```

Il suffit de modifier nos deux textes. Dans le premier, mettons le nom de la ville. Dans le deuxième, mettons la température obtenue avec la requête. Attention, la réponse étant au format json, il faudra être précis pour trouver la température.
Je vous recommande de lancer la requête dans le navigateur pour pouvoir visualiser facilement la réponse.

Un fichier de correction weather.ejs est disponible. Il contient le script et le front.

API n°2 : Youtube

Cette api nécessite un compte google cloud platform avec une clé API active

Pour notre subscriber count, nous utiliserons l'api officielle youtube (*www.googleapis.com/youtube/v3*). Cette API est gratuite tant que le nombre de requête ne dépasse pas un certain nombre. Pour cela, elle nécessite une clé API google active. Vous pouvez créer une clé API sur google cloud platform.

Pour commencer, dans votre fichier ejs, créez deux zones de texte, une zone d'input ainsi qu'un bouton qui appellera le script *youtube()* quand il est pressé.
Pensez à donner des id uniques aux zones de texte et d'input car nous les modifierons dans le script.
L'input devra contenir le nom précis d'une chaine youtube.

Maintenant que le front est en place, passons au script Javascript.
Commencez par importer le module *http.*
Ensuite, déclarez la fonction *youtube()*, elle ne prend rien en paramètre.

Maintenant, grâce aux id unique que vous avez utilisé, récupérez vos deux zones de texte et votre input dans le script.

Il ne reste plus qu'à faire la requête http. Pour cela nous utiliserons XMLHttpRequest.
Pour cela, rien de compliquer. Il faut commencer par créer une variable comme ceci.

```
var call = new XMLHttpRequest();
```

Maintenant, nous devons définir les propriétés de notre requête. Nous utiliserons les propriétés suivantes :

```
req.responseType = '';
req.open(type, url, bool);
req.onload  = function() {};
```

responseType correspond au format de réponse que nous recevrons (text ou json par exemple).
open() est la fonction qui permet d'instancier la requête. Dans notre cas, elle prend 3 paramètre :

- Type -> le type de requête que nous effectuons (GET, POST, ect …)
- url -> l'url de la requête http
- bool -> true si la requête est asynchrone (dans notre cas true) .

onload correspond à la fonction qui sera exécutée lorsque nous recevrons une réponse.

Enfin, on envoie la requête avec :

```
req.send(null);
```

Pour finir, il ne reste plus qu'à faire la fonction qui correspond à onload. La réponse de la requête se situe dans

```
req.response
```

Il suffit de modifier nos deux textes. Dans le premier, mettons le nom du youtubeur. Dans le deuxième, mettons son nombre d'abonnés et le total de ses vues. Attention, la réponse étant au format json, il faudra être précis pour trouver les bons chiffres.
Je vous recommande de lancer la requête dans le navigateur pour pouvoir visualiser facilement la réponse.

Un fichier de correction youtube.ejs est disponible. Il contient le script et le front.

API n°3 : Blagues

Pour notre api de blague nous utiliserons JokeAPI v2 by Sv443. Elle ne nécessite aucune clé ou token.

Pour commencer, dans votre fichier ejs, créez deux zones de texte, un sélecteur déroulant ainsi qu'un bouton qui appellera le script *joke()* quand il est pressé.
Pensez à donner des id uniques aux zones de texte et d'input car nous les modifierons dans le script.
Le sélecteur déroulant devra contenir le type de la blague à afficher (Dark, Programming, Miscellaneous).

Maintenant que le front est en place, passons au script Javascript.
Commencez par importer le module *http.*
Ensuite, déclarez la fonction *joke()*, elle ne prend rien en paramètre.

Maintenant, grâce aux id unique que vous avez utilisé, récupérez vos deux zones de texte et votre sélecteur déroulant dans le script.

Il ne reste plus qu'à faire la requête http. Pour cela nous utiliserons XMLHttpRequest.
Pour cela, rien de compliquer. Il faut commencer par créer une variable comme ceci.

```
var call = new XMLHttpRequest();
```

Maintenant, nous devons définir les propriétés de notre requête. Nous utiliserons les propriétés suivantes :

```
req.responseType = '';
req.open(type, url, bool);
req.onload  = function() {};
```

responseType correspond au format de réponse que nous recevrons (text ou json par exemple).
open() est la fonction qui permet d'instancier la requête. Dans notre cas, elle prend 3 paramètre :

- Type -> le type de requête que nous effectuons (GET, POST, ect …)
- url -> l'url de la requête http
- bool -> true si la requête est asynchrone (dans notre cas true) .

onload correspond à la fonction qui sera exécutée lorsque nous recevrons une réponse.

Enfin, on envoie la requête avec :

```
req.send(null);
```

Pour finir, il ne reste plus qu'a faire la fonction qui correspond à onload. La réponse de la requête se situe dans

```
req.response
```

Il suffit de modifier nos deux textes.
La réponse nous donne le type de blague : single ou twopart.
Si la blague est de type single, une seule zone de texte est nécessaire.
Sinon, il faudra mettre la question dans la première zone texte et la chute dans la deuxième.
Je vous recommande de lancer la requête dans le navigateur pour pouvoir visualiser facilement la réponse.

Un fichier de correction joke.ejs est disponible. Il contient le script et le front.

Exercice 4 : Mettre en place un système de login

À la suite de l'exercice 2, vous avez, sur votre page d'accueil, deux boutons login et register.
Commencez par rediriger ces deux boutons vers les pages respectives de login et register.
Pour cela, ajouter

```
<a href="/user/login" et <a href="/user/login"
```

dans vos boutons dans votre page home.ejs
Pensez à bien modifier les routes dans user.js pour qu'elles render login et register