| Construire une API de démonstration en Node.js
| TD

Éléments de base

- ***Définition simple d'une API***

Une **API** (*Application Programming Interface*) est un moyen de communication entre deux terminaux, classiquement un client et un serveur. Les données sont spécifiquement demandées au serveur suivant une norme (**RESTFull** – *Representational State Transfert*).

https://fr.wikipedia.org/wiki/Representational_state_transfer

- ***Utilisation simple d'une API***

Les demandes sont effectuées par des URI via des verbes HTTP (GET, POST, …) et le serveur retournera les données suivant son type de format, à partir d'une base de données ou équivalent.

- ***Méthode de construction***

Les réponses étant les plus couramment retournées en JSon, NodeJS et son framework Express représente la meilleure solution (et la plus rapide) pour écrire une API.

Création de l'API

Pré-requis : NodeJS, Visual Studio Code et Chrome installés.

T01 | Dans votre répertoire utilisateur de vos projets NodeJS, créez un répertoire `api` et allez-y :

```
$> mkdir api
$> cd api
```

T02 | Initialisez le projet à l'aide de l'outil NPM :

```
$api> npm init -y
```

Cette commande initialise le projet par la création d'un fichier `package.json` instaurant le contexte des futurs paquets nécessaires à l'application. Techniquement, on voit que notre projet doit démarrer par un script nommé `index.js`.

On utilisera le framework Express pour la construction de l'API.

T03 | Installez Express :

```
$api> npm install express
```

La dépendance est ajoutée automatiquement (entre autres ajouts) dans le fichier `package.json`.

T04 | Créez le fichier `index.js`, point de départ de l'application :

```
const app = require('express')() ;
const PORT = 8080 ;

app.listen(
   PORT,
   () => console.log('API sur http://localhost:'+PORT)
);
```

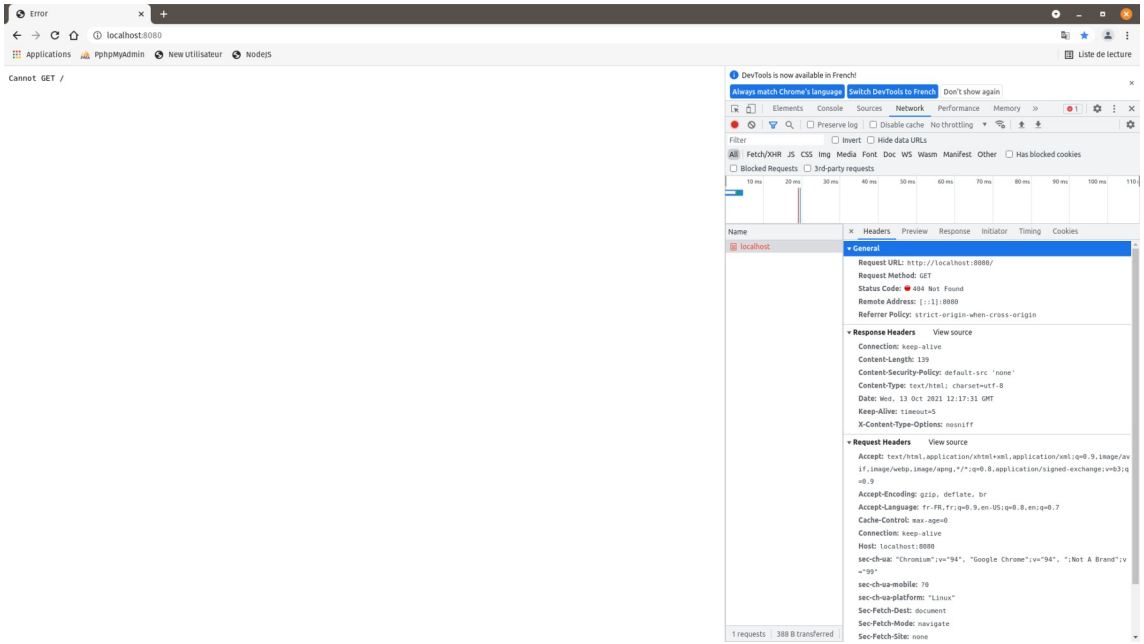
Explications :

→ on définit la constante représentant notre application via le midleware `express()`
→ on indique le port d'écoute pour HTTP
→ on lance l'écoute avec un retour sur la console

T05 | Lancez le serveur HTTP par la commande :

```
$api> node .
```

Le lancement de la page montre une erreur 404 (`network → headers`) ; faire aussi la démarche avec **Insomnia**. Car il n'y a pour l'instant aucune réponse de l'API (ou point de terminaison).



Création des deux méthodes usuelles

Note : *à chaque changement dans le code, vous devez relancer le serveur NodeJS.*

- ***GET***

T06 | Ajoutez dans le fichier `index.js`, (**AVANT le listen**) la méthode :

```
app.get('/',(req,res)=>{
   res.status(200).json({
      song : 'Une chanson',
      title : 'Un titre'
   })
});
```

Lancez à nouveau : cette fois-ci cela fonctionne avec comme retour un fichier JSon.

Supposons maintenant que, puisque l'on parle de JSon, on veuille traiter avec un fichier…

T07 | Créez un répertoire à la racine du projet nommé `raw` et placez-y le fichier intitulé `songs.json` au contenu suivant :

```
[{
   "titre": "The Musical Box",
   "auteur": "Genesis",
   "genre": "Rock"
 },{
   "titre": "Epitaph",
   "auteur": "King Crimson",
```

```
   "genre": "Rock"
 },{
   "titre": "Extreme Ways",
   "auteur": "Moby",
   "genre": "Pop"
}]
```

T08 | Modifiez et ajoutez les constantes en tête du fichier `index.js` :

```
const express = require('express');
const path = require('path');
const fs = require('fs');
const app = express();
const PORT = 8080;
```

On note que l'on fait appel à deux modules supplémentaires : l'un pour la gestion des répertoires, l'autre pour la gestion des fichiers.

T09 | Ajoutez dans le fichier `index.js`, (**AVANT le listen**) la méthode :

```
app.get('/api',(req,res)=>{
   fs.readFile(path.join(__dirname,'raw/songs.json'),'utf8',function(err,songs) {
      if (err) { throw err; }
      let liste = JSON.parse(songs);
      res.send(liste);
   });
});
```

Cette fois-ci, on répond sur l'URL `http://localhost:8080/songs`, où la lecture du fichier se fait avant son passage en valeur JSon → JavaScript pour une réponse à la demande.

Pour répondre à la norme REST, on peut apporter dans le GET une valeur pour sortir un seul élément. On rajoute alors une autre méthode pour ce type de demande.

T10 | Ajoutez dans le fichier `index.js`, (**AVANT le listen**) la méthode :

```
app.get('/api/:id',(req,res)=>{
   let { id } = req.params;

   fs.readFile(path.join(__dirname,'songs/songs.json'),'utf8',function(err,songs) {
      if (err) { throw err; }

      let liste = JSON.parse(songs);
      let long = liste.length;
      id = id -1;

      if ((id>=0) && (id<long)) {
         res.send(liste[id]);
      } else {
           res.status(418).send({ message: 'ID non valide !' });
        }
   });
});
```

Le code est assez simple et ne nécessite pas d'explications supplémentaires…

- ***POST***

Pour cette méthode, nous passerons par le logiciel **Insomnia** afin de pouvoir envoyer des POST en simulant l'envoi d'un formulaire.

T11 | Ajoutez dans le fichier `index.js`, (**AVANT le listen**) la méthode :

```
app.post('/songs',(req,res)=>{
   const { titre } = req.body;
   const { auteur } = req.body;
   const { genre } = req.body;
```

```
    res.send({
        donnees:`${titre} ${auteur} ${genre}`
    });
});
```

Dans cette méthode, il se pose un problème : nous allons envoyer des données sous forme JSon. Il faut donc indiquer (et donner) la possibilité au middleware **Express** de traiter du JSon dans le corps d'une requête.

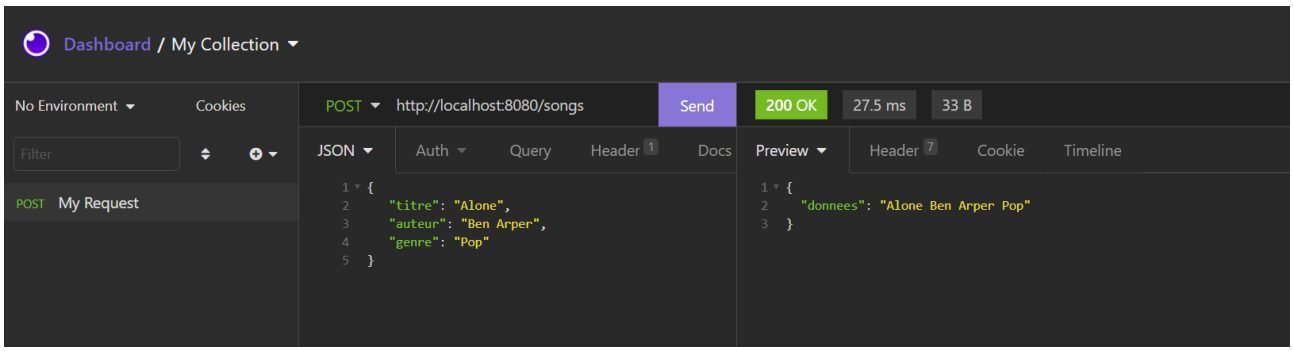
T12 | Ajoutez après les constantes du début de fichier ceci :

```
app.use(express.json());
```

La chanson supplémentaire :

```
{
    "titre": "Alone",
    "auteur": "Ben Arper",
    "genre": "Pop"
}
```

Le reste se passe avec Insomnia :



On voit bien le passage de la chanson supplémentaire. Il suffira dans la méthode POST de l'ajouter au fichier…