

Node JS : Express Js



Express

Tables des matières : Partie 1

1. **Express Js ?**
2. **npm init**
 - 2.1 Package.json exp
3. **Installation de Express**
 - 2.2 Le -g et le npm uninstall
 - 2.3 Le fichier package-lock.json
4. **Utilisation**
 - 4.1 Astuce : Nodemon
5. **Les routes**
 - 5.1 Req, Res, Next
 - 5.1.1 : Req
 - 5.1.2 : Res
 - 5.1.3 : Next
6. **Body Parser**
7. **Les templates**

Tables des matières : Partie 2

8. Interaction avec MySql

8.1 Create

8.2 Read

8.3 Update

8.4 Delete

9. Découpage MVC

10. Astuces

11. Session

12. DataTable

13. Le Router

14. Un autre moteur de modèles : “PUG”

15. Traitement des erreurs

17. Exercises Global

Qu'est-ce que Express JS ?

Express JS?

Vous l'avez remarqué, Node JS est bas niveau et événementiel, il faut tout gérer.

C'est bien 5 minutes mais...n'y aurait-il pas quelque chose pour nous faciliter la vie?

Oui! Le framework Express de Node JS



Là où il faut créer des modules de modules pour gérer un tant soit peu proprement nos routes, Express va nous proposer un système intégré tellement plus pratique.

- Gestion simplifiée des routes
- Gestion aisées des routes dynamiques

Express JS?

Express est une infrastructure d'applications Web Node.js minimaliste et flexible qui fournit un ensemble de fonctionnalités robustes pour les applications Web et mobiles.

Grâce à une foule de méthodes utilitaires HTTP et de middleware mise à votre disposition, la création d'une API robuste est simple et rapide.

Express apporte une couche fine de fonctionnalités d'application Web fondamentales, sans masquer les fonctionnalités de Node.js que vous connaissez et appréciez.

Express JS?

Il vient se greffer sur Node.js pour nous aider à :

- Créer des applications webs SPA, multipage, mobile first, etc...
- Configurer des « middleware » servant à gérer les requêtes http.(API)
- Configurer les tables de routage pour effectuer les différentes actions demandées via http
- Manipuler/gérer des templates dynamiques
- gérer des fichiers statiques tel que les css, les js, image, etc...
- interagir avec une base de données sql ou no-sql ou encore real-time
- ...

En fait, tout comme node.js mais en plus facile et plus rapidement :)

Npm init

Npm init

Dans le cadre d'un développement propre et disponible au partage nous allons démarrer un projet **server.js**, MAIS ! en utilisant la commande **npm init**.

Cette commande va permettre de créer un package.json, qui sera, un résumé de notre serveur, avec ses infos, commandes et ses dépendance que nous aurons installées.

Avantage

Le package.json est utilisé par npm pour définir les dépendances de notre package.

Cela nous permet d'avoir une trace des packages (Nom, Version) utilisés et facilitera le partage de notre projet avec des personnes tierces ou simplement à restaurer notre projet en cas de perte ou de déplacement sans se soucier des versions actuelles des packages installés.

Npm init

Après avoir créé un fichier de base du serveur en .js, (index, server, app ...)

Lançons la commande “**npm init**”:

Node va nous poser une série de question, seul une est réellement importante et obligatoire.

Voyons ensemble les questions :

```
Press ^C at any time to quit.
package name: (test)
version: (1.0.0)
description:
entry point: (server.js)
test command:
git repository:
keywords:
author:
license: (ISC)
About to write to C:\Users\Evengyl\Desktop\Bstorm\2021\I3 Bx - Web9\test\package.json:

{
  "name": "test",
  "version": "1.0.0",
  "description": "",
  "main": "server.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1",
    "start": "node server.js"
  },
  "author": "",
  "license": "ISC"
}

Is this OK? (yes) ☐
```

Npm init

Package name : (test) → le nom de votre projet, package car oui, au final vous allez créer un package également disponible pour un bon vieux “**npm install**”.

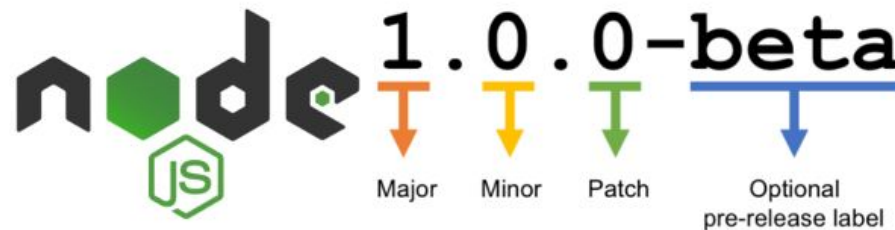
Attention !

veuillez donner un nom à votre package assez **explicite**, veiller également à ne pas le nommer comme une des dépendance que nous installerons plus tard, car il ne voudra simplement pas les installer... ah oui, notre projet étant un package, vous ne pouvez pas installer un package du même nom dedans... Logique imparable.

Exemple : **Express** ne marchera pas, ou tout du moins, pas pour après !

Npm init

Version : (1.0.0) → C'est la version de votre projet. Par défaut, voici comment fonctionne le système de version avec node.js :



Nous, nous n'allons pas nous attarder là dessus, la version 1.0.0 de base est très convenable pour le moment.

Npm init

description : → représente une brève description de votre projet (Facultatif)

entry point : (server.js) → c'est la qu'entre en jeux le nom de votre serveur, ou tout du moins le nom que vous lui avez donné quand vous avez créé votre fichier .js initial. Attention ce fichier doit exister ou alors vous devrez le créer, car c'est comme ça que la commande node plus tard fonctionnera avec les packages additionnels.

test command : → elle permet de configurer le nom de la commande de test, nous ne nous attarderons pas sur ça pour le moment. (Facultatif)

git repository : → vous pourrez entrer ici le nom de votre répo git, car npm init le permet, mais il vous faudra quand même une configuration **git** ici, c'est juste pour le partage de package que c'est intéressant pour les personnes consultant votre projet sur **npm** (Facultatif)

Npm init

keywords : → cette option vous permet de renseigner des mots clés sur votre serveur pour le retrouver plus facilement si vous le déposer sur **npm**, c'est un tableau qui sera généré sur base des mot-clés entré, attention, ils doivent être séparé par une virgule. (Facultatif)

author : → ??? vous ??? :), ou votre équipe / société / etc...

license : (ISC) → est une license de publication gratuite par défaut, elle signifie : **Internet Systems Consortium License**, retenez juste qu'elle est plus permissive que la license **MIT** classique.

Il vous propose ensuite un résumé de la configuration. Si celle-ci vous convient : vous pouvez accepter

Npm init

Cela à pour but de nous créer un fichier **package.json**.

Ce fichier contient sous format json et intelligible par **npm** et **Node**, la configuration que vous avez écrite sous la commande **npm init**.

Ce fichier recevra par la suite, les informations concernant les packages, dépendances qui viendront compléter notre package.

```
{
  "name": "test",
  "version": "1.0.0",
  "main": "index.js",
  ▶ Debug
  "scripts": {
    "test": "echo \"Error: no test\"
    "start": "node server.js"
  },
  "author": "",
  "license": "ISC",
  "keywords": [],
  "description": ""
}
```

```
{
  "name": "test",
  "version": "1.0.0",
  "main": "index.js",
  ▶ Debug
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1",
    "start": "node server.js"
  },
  "author": "",
  "license": "ISC",
  "keywords": [],
  "description": "",
  "dependencies": {
    "express": "^4.17.1"
  }
}
```

Exercices

Prenez le temps créer une configuration de package avec toutes les informations que vous pourrez.

Temps accordé : 15 min

Installation

Installation

L'installation se fait très simplement, nous avons brièvement vu que Node Js s'installait avec certains outils complémentaires tels que le NPM (Node Package Manager)

Nous allons donc enfin l'utiliser pour installer express:

npm install express

```
C:\Users\Evengyl\Desktop\Bstorm\2021\NodeJS\DemoETU\ExpressDemo>npm install express
npm notice created a lockfile as package-lock.json. You should commit this file.
npm WARN expressDemo@1.0.0 No description
npm WARN expressDemo@1.0.0 No repository field.

+ express@4.17.1
added 50 packages from 37 contributors and audited 50 packages in 2.1s
found 0 vulnerabilities

C:\Users\Evengyl\Desktop\Bstorm\2021\NodeJS\DemoETU\ExpressDemo>
```

Le -g et le npm uninstall

Vous pouvez également installer des dépendance en mode global, donc efficiente pour tous vos projet lancés depuis ce poste de travail, elles s'inscrivent dans le répertoire des dépendances native de **Node**.

Attention de cette manière, elles ne s'ajoutent pas au package.json, ce qui amènera forcément à des conflits si vous **déplacez / donnez** votre projet.

Dans tous les cas, si vous faite une fausses manoeuvre ne vous en faite pas, tout est toujours rattrapable.

Pour désinstaller une dépendance que vous auriez installée par mégarde ou dans une mauvaise version, utilisez la commande suivante pour la désinstaller.

npm uninstall depName

ou si elle à été installée en **-g** “global” (ce qui pourrait être une erreur dans certain cas), utilisez

npm uninstall -g depName

Le package-lock.json

Lorsque l'installation sera terminée vous verrez apparaître, si vous n'avez pas fait une installation en **-g**, un fichier nommé **package-lock.json**

Ce fichier contient tous les informations sur les packages requis pour faire tourner la dépendance installée.

Ici dans le fichier, nous n'aurons que les packages pour faire tourner **Express**, voyez comme le fichier est déjà grand...

Il s'avère également que l'installation nous à créer un dossier **node_modules**.

Ce dossier est la résultante de tout ce qu'a besoin en terme de fichier, express, pour fonctionner.

Express et autre.. toute les dépendance on besoin de un ou plusieurs package pour tourner.

Exercices

En reprenant l'exercice précédent, installer le package express et vérifier qu'il est bien installé.

Temps accordé : 15 min

Utilisations

Express JS

Utilisation

Express installé, il ne nous reste plus qu'à l'utiliser :

- Comme toutes librairies, nous allons effectuer un import au moyen de la commande *'require'*.

```
var express = require('express');
```

Remarque :

Les librairies installée nativement par Node ne sont pas présente dans le node_modules, les librairies que nous allons utiliser ne sont pas toute directement native, il ne sera donc pas étonnant de voir grossir le dossier des node_modules assez vite, car une librairies ne comporte pas toujours une seul dépendances !...

Un projet Angular par exemple comporte +- 60.000 fichiers dans le dossier node_modules... , prenez le cap de ne jamais transférer votre projet avec le dossier node_modules, par défaut même le .gitignore créé ne le compte pas dans le projet, vous devrez simplement faire un **npm install** plus tard...)

Utilisation

Et c'est tout? Presque!

Dans la version que je qualifierai de « traditionnelle » nous avons du créer un server au moyen de la librairie 'http'... les règles en vigueur ne changent pas. Seule la manière d'aborder la problématique évolue.

Plus question de faire ca:

```
var http = require('http');  
var server = http.createServer(function(req,res){ //... });  
  
server.listen(8888);
```


Utilisation

On va changer pour ca:

```
var express= require('express');  
var app = express();
```

```
app.listen(3000);
```

A noter que 'var express' et 'var app' sont des conventions à respecter dans votre intérêt, toute la documentation fonctionne sur base de ces déclarations.

```
var express = require('express');  
var app = express();  
  
//...  
  
const port = 3001  
app.listen(port, console.log(`Les serveur Express écoute sur le port ${port}`))
```

Utilisation

Concrètement c'est '**express()**' qui se charge de créer notre serveur.

Nous n'avons plus qu'à l'écouter au moyen de la méthode connue désormais:
listen(port);

```
app.listen(port, console.log(`Les serveur Express écoute sur le port ${port}`))
```

Jusque là nous n'avons encore rien fait d'inhabituel.

Attaquons ces fameuses routes 😊

Rapide astuce : Nodemon

Pour faciliter la vie de notre serveur et de notre cerveau nous allons utiliser une micro librairie presque aussi utilisée que express,

Il s'agit de **nodemon**, son but est de simplement capter tout changement fait sur nos **js,mjs,json** ! j'ai bien dis **.js...**, (les futures templates ne seront pas pris en compte eux)

Il nous permettra de relancer le serveur automatiquement en cas de sauvegarde de changements dans le projet serveur, plus besoin de couper et de relancer le serveur manuellement.

Nous allons l'installer dans le native **node**, donc avec :

```
npm install -g nodemon
```

Et voilà il sera effectif pour tout nos projet.

Si une erreur de type interdiction de lancer nodemon à cause de policy de lancement de script extérieur, il suffit de taper cette ligne pour régler le soucis. (console powershell en administrateur)

```
-> Set-ExecutionPolicy -Scope "CurrentUser" -ExecutionPolicy "Unrestricted"
```

ps : [nodemon - npm \(npmjs.com\)](https://nodemon.io)

Exercices

En partant de l'exercice précédent, déployer un serveur express et vérifier son lancement / rechargement auto avec nodemon.

Temps accordé : 15 min

Les Routes

Les Routes

Les frameworks Web fournissent des ressources telles que des pages HTML, des scripts, des images, etc. sur différentes routes (url).

ExpressJS nous facilite la gestion de ces routes via la fonction `app.method(path,handler)`

- App : L'application express en elle même
- Method : l'un des verbes http : GET, POST, PATCH, PUT, DELETE,...
- Path : Chemin de l'url qui doit être capturée
- handler : fonction anonyme ou fléchée :) qui doit être exécutée

Remarque :

Express propose également la méthode `all` qui permet de réagir à une requête url de la même façon indépendamment du verb utilisé -> exemple dans les slides suivant

Les Routes

Le routing de base suit un pattern très simple:

app.METHOD(PATH, HANDLER)

Exemple:

```
app.get('/', function(req, res) {  
    console.log('hello world');  
});
```

Ici nous disons à notre route express, intercepte le verb http get qui a pour url “/” donc rien, ou l’index, ou la home page, comme on veut, et exécute la fonction anonyme suivante, cette fonction nous donnera deux paramètres, même trois en fait :) mais nous le verrons plus loin.

Le **req**: contient toute les informations concernant la requête capturée.

Le **res**: est l’élément que le serveur express va renvoyer à son interlocuteur. (le navigateur par ex)

Les Routes

Jusque la rien de bien compliqué...

Petit rappel pour les verbes **http** connus (liste non exhaustive):

- **GET**
 - Méthode de requête le plus couramment utilisé pour la demande de ressources au server.
- **POST**
 - Méthode de requête utilisée pour envoyer des ressources au serveur en vue du traitement de données généralement issues d'un formulaire.
- **PUT**
 - Méthode de requête demandant la mise à jour de la ressource ciblée sur le serveur. + pour API
- **DELETE**
 - Méthode de requête demandant la suppression de la ressource ciblée sur le server + pour API

Les Routes

Express peut utiliser le package *path-to-regexp*

(<https://www.npmjs.com/package/path-to-regexp>) pour effectuer le match entre l'url et le path défini au niveau de notre route.

Nous pouvons donc définir une route avec un path simple */*, */about*, */home* etc...

Mais également utiliser des masques pour router :

- **'/ab+cd'** : capture les routes */abcd*, */abbcd*, */abbbcd*, etc... (le **+** signifie un ou plus)
- **'/ab*cd'** : capture les routes */abcd*, */abxcd*, */ab123cd*, */abZorrocd*, etc.... (le ***** signifie **toute autres caractères**)
- **'/ab(cd)?e'** : capture les routes */abe*, */abcde* (le **()?** signifie facultatif)

Les Routes

Les Routes Dynamiques: Routes variant en fonction de paramètres passés dans l'url.

Verb **http** utilisé: **GET**

Pattern utilisé :

`app.method(PATH/:PARAMName, HANDLER)`

Exemple :

```
app.get('/', function(req,res){  
  console.log( 'Hello world' )  
});    -> Sans Paramètres
```

```
router.get('/:id', function(req, res) {  
  console.dir(req.params);  
  res.send('The id you specified is ' + req.params.id);  
});
```

Les Routes

Construction des routes

Nous avons la possibilité de facilement construire nos routes pour permettre des segments statiques et dynamique

Exemple :

('/:?) : Permet d'avoir une route <http://domain/{?}> où ? est un paramètre passé à notre route

```
router.get('/Hello/:nom/:prenom', function(req, res) {  
  res.send('Bonjour ' + req.params.prenom + ' ' + req.params.nom);  
});
```

Les Routes

Le problème réside dans le fait qu'on ne peut imposer le type de paramètre à passer.

Il est donc de votre responsabilité de vérifier l'exactitude du paramètre dans votre fonction de callback **function(req, res) { ... }** ou **(req, res) => { ... }**.

Nous avons également la possibilité d'utiliser le routeur de Express mais il fera partie d'un autre chapitre de ce cours (MVC) quand nous aurons découpé le code en plusieurs dossier/fichier de travail pour l'organisation de notre code pour le rendre réutilisable / organisé.

Les Routes

Comme mentionné plus haut, Express propose également une alternative au choix d'un verb http, il propose le `.all`.

Il correspond d'office avec tous les verbs http (Get Post Put Delete etc...) utilisé mais gardera quand même sa forme de match avec l'url, pour ça, ça ne change rien.

Un petit cas pratique : Si on match avec tel url, peu importe le verbs, on crée un log en bsd par exemple.

```
app.all("/", function(req, res, next)
{
  //ici dans tout les verbs http possible, lors d'une url /,
  //Express utilisera se middleware et passera au suivant avec le .next
  console.log("Passe ici")
  next()
})
```

Les routes : la 404

Nous pourrions par la suite de ces raisonnements, en déduire une chose intéressante...

Comment gérer la fameuse 404, elle famoso 404..

En fait c'est très simple, en considérant la chose comme suit ; Toute les **routes** seront lue dans l'ordre d'écriture, les **middlewares** interceptent les **routes** correspondantes aux **patterns**. nous pouvons en déduire que si la **requête** passe à travers tous les **patterns**, nous nous retrouvons avec une **requête** non traitée en fin de **fichier**... ce qu'y veut dire que c'est là que nous placerons notre **404**.

Elle se traite avec un **pattern** connu maintenant. Le **app.all** et le **pattern match "*"**.

```
app.all("*", (req, res) => {  
  res.status(404).send("404")  
})
```

Les Routes : multipoints

Toujours en restant dans la bienveillance de **express** à notre égard, nous avons également la possibilité de lui dire (notre URI) d'être intercepté dans un middleware par plusieurs routes différentes.

```
app.get(['/', '/listUser'], (req, res, next) => {
```

Un simple **tableau d'URI** permettra à express de savoir quand il doit matcher tel ou tel route.

Dans l'exemple ici présent, nous constatons que quand l'url sera "/" ou "**listUser**", elle sera **matchée** et rentrera dans ce **middleware** de routage

Exercices

Sur base de l'exercice précédent, créer un système de routage complet qui portera sur le thème suivant : un cv complet multipage et multi-services.

Il doit s'y trouver au moins les pages suivantes :

Accueil, Mes infos personnelles, Mes données de contact, mon parcours professionnel, mon parcours scolaire, mon parcours de formations, mes passions / hobbies

Ces routes afficherons toutes leurs "titre" en html sur le site. à vous de trouvez le bon nom de route associée à cette thématique.

Temps accordé : 45 min

Req Res Next

Req, Res, Next

Depuis pas mal de slide vous devez vous demander une chose, à quoi sert et comment utiliser tout ces fameux paramètres de callback de notre fonction de route.. nous parlons donc de **req res next** que vous voyons apparaître avec ou sans le next etc...

Il est temps de rendre dynamique nos route et de gérer ce qu'il en sort et ce qu'il y rentre, que faire à tel endroit ou à tel moment etc...

Le **req** : Cette variable contient tout ce que la requête entrante nous apporte depuis le navigateur ou autre, lister tout ce qu'elle contient sera de la folie surtout vu la redondance d'informations qu'elle contient.

Mais en voici déjà un bon aperçu pour le développement futur.

Req


Req contient (de manière non-exhaustive) :

- l'encodage de la requête,
- l'événement déclencheur (Si il y en a un),
- les éléments de timeout de requête etc..
- le header de la requête au complet !
- l'url, le verb Http
- la route utilisée avec ses **paramètres** etc.. :)
- un bel objet json avec les params de route !

Ok... quelques exemples s'imposent pour certaines petites choses.

l'url :

```
//Routes//  
app.get("/:param", (req, res)=> {  
  console.log(req.url)  
  res.status(200).end();  
})
```



```
/YeahUr1  
[]
```

Req

Les Params (chemin de routage):

```
//Routes//  
app.get("/:param", (req, res)=> {  
  console.log(req.params)  
  res.status(200).end();  
})
```

```
{ param: 'YeahUrl' }  
[]
```

La méthode (verb http) :

```
//Routes//  
app.get("/:param", (req, res)=> {  
  console.log(req.method)  
  res.status(200).end();  
})
```

```
GET  
[]
```

Req

Le **header** (headers & rawHeaders) :

```
//Routes//  
app.get("/:param", (req, res)=> {  
  console.log(req.headers)  
  res.status(200).end();  
})
```

```
{  
  'user-agent': 'PostmanRuntime/7.26.8',  
  accept: '*/*',  
  'postman-token': '4f086129-747a-450c-975d-cc199a75ccc1',  
  host: 'localhost:3000',  
  'accept-encoding': 'gzip, deflate, br',  
  connection: 'keep-alive'  
}
```

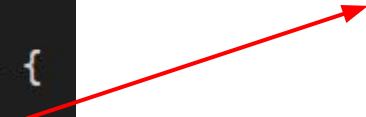
```
//Routes//  
app.get("/:param", (req, res)=> {  
  console.log(req.rawHeaders)  
  res.status(200).end();  
})
```

```
[  
  'User-Agent',  
  'PostmanRuntime/7.26.8',  
  'Accept',  
  '*/*',  
  'Postman-Token',  
  '45ab13d4-f0a6-4a21-8ee9-27d9c1c8801d',  
  'Host',  
  'localhost:3000',  
  'Accept-Encoding',  
  'gzip, deflate, br',  
  'Connection',  
  'keep-alive'  
]
```

Req

Les Paramètres nommés (Query) :

```
//Routes//  
app.get("/", (req, res)=> {  
  console.log(req.query)  
  res.status(200).end();  
})
```



```
{ tata: 'YeahUrl' }  
[]
```

Celui-ci est pour nous le plus important. il permet de récupérer les paramètres nommés passer dans l'url, un paramètre nommé ressemble à ceci pour cet exemple :

GET	▼	http://localhost:3000/?tata=YeahUrl
-----	---	-------------------------------------

Res

La variable **res** est la variable qui sera renvoyée au navigateurs ou autre système / service appelant, ayant envoyé la requête http en fait.

De base elle contient sensiblement les mêmes choses que **req**.

Nous pouvons dès lors, terminer une demande **URI** (url, requête etc...) venant de req et ayant été interceptée par express router. **app.VERB()** etc...

Suivant un raisonnement logique, toutes requêtes **URI**, doit obligatoirement répondre quelque chose, sinon le navigateur / services va tourner en boucle sans savoir ce qu'il doit attendre / faire... donc ? plantage !

Voyons ensembles une séries de points concernant le paramètre “**res**”.

Res

Les méthodes de réponses fournies par Node avec Express :

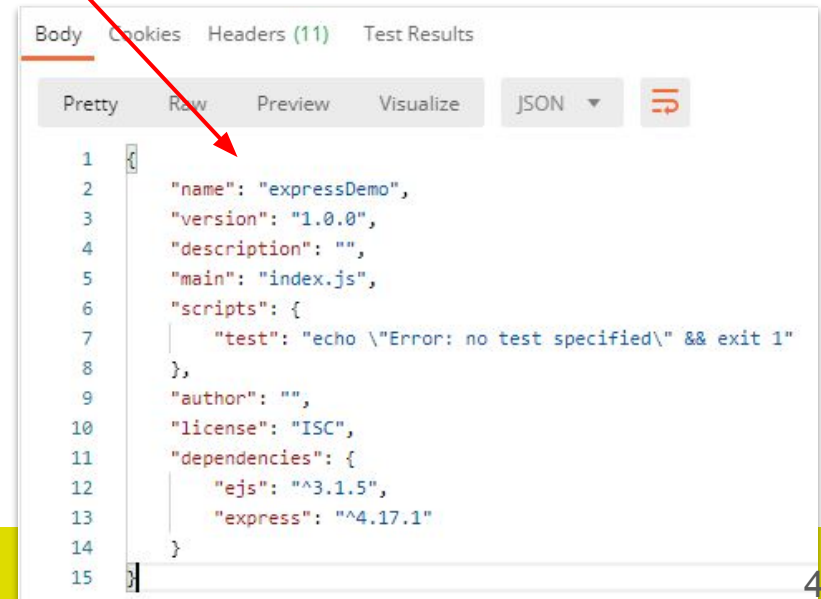
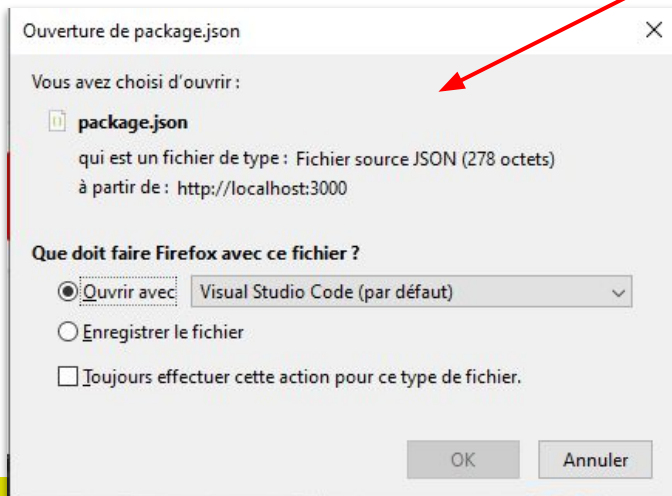
Méthode	Description
<code>res.download()</code>	Vous invite à télécharger un fichier.
<code>res.end()</code>	Met fin au processus de réponse.
<code>res.json()</code>	Envoie une réponse JSON.
<code>res.jsonp()</code>	Envoie une réponse JSON avec une prise en charge JSONP. Non vue dans ce cours
<code>res.redirect()</code>	Redirige une demande.
<code>res.render()</code>	Génère un modèle de vue.
<code>res.send()</code>	Envoie une réponse de divers types.
<code>res.sendFile()</code>	Envoie une réponse sous forme de flux d'octets. Non vue dans ce cours
<code>res.sendStatus()</code>	Définit le code de statut de réponse et envoie sa représentation sous forme de chaîne comme corps de réponse. Une autre version sera vue.

Res : download()

le **res.download("pathFileName")** → Méthode permettant de répondre à la requête par le lancement d'un téléchargement de fichier. Elle prendra comme paramètre le chemin relatif du fichier que le serveur doit renvoyer en téléchargement.

exemple :

```
//Routes//  
app.get("/", (req, res)=> {  
  res.download("package.json");  
})
```

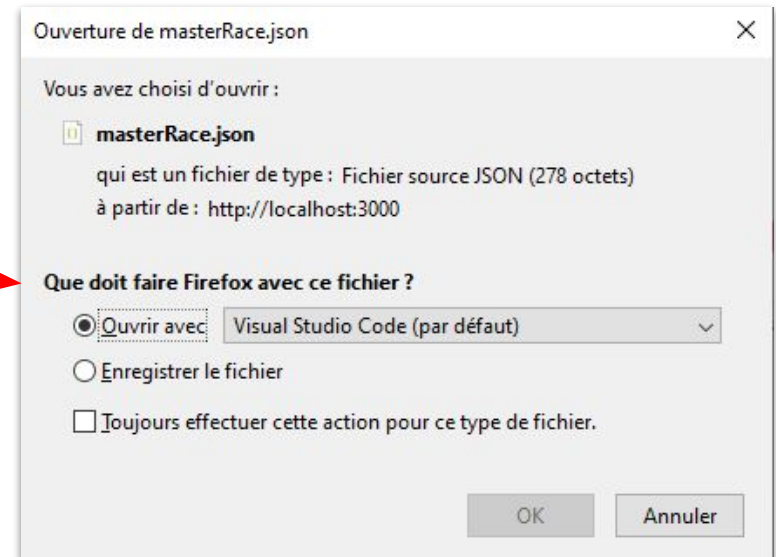


Res : download()

le `res.download("pathFileName", "nameFileFinal, function (err) {`
`if(err){ ... }`

`})` → elle existe aussi en version plus complète, sont deuxième paramètre sera le nom final pour le téléchargement, et ensuite une petite fonction anonyme pour gérer le cas d'erreur de téléchargement.

```
//Routes//  
app.get("/", (req, res)=> {  
  res.download("package.json", "masterRace.json", function (err){  
    if(err){  
      console.log("aie, erreur de dl")  
    }  
  });  
})
```



Res : end()

le **res.end()** : cette méthode met fin directement au processus de réponse, elle n'est pas propre à **Express** mais est utilisée aussi sur celui-ci.

Elle ne permet pas de transférer des données, elle est souvent alliée à une seule autre méthode que nous verrons par la suite. la méthode **.status()**.

La seule chose que nous pouvons constater pour cette méthode c'est qu'elle empêche tout bonnement le navigateur de tourner en rond alors que nous n'avons rien à renvoyer de spécifique.

```
//Routes//  
app.get("/", (req, res)=> {  
  res.end()  
})
```


Res : Json()

Le `res.json()` : permet de renvoyer un objet json, qu'il soit nommé, ou complexe, elle comprendra et renverra l'intégralité de la variable passée.

exemple :

```
var jsonVar = {
  "name": "expressDemo",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "ISC",
  "dependencies": {
    "ejs": "^3.1.5",
    "express": "^4.17.1"
  }
}

//Routes//
app.get("/", (req, res)=> {
  res.json(jsonVar)
})
```



JSON	Données brutes	En-têtes
Enregistrer	Copier	Tout réduire Tout développer 🔍 Filtrer le JSON
name:	"expressDemo"	
version:	"1.0.0"	
description:	""	
main:	"index.js"	
▼ scripts:		
test:	"echo \"Error: no test specified\" && exit 1"	
author:	""	
license:	"ISC"	
▼ dependencies:		
ejs:	"^3.1.5"	
express:	"^4.17.1"	

Res : Redirect()

le **res.redirect(URI)** : permet de signifier à **Express** que la **requête**, même si elle à été traitée ici mais que, par exemple n'est pas terminée ou doit aboutir à une autre page, de créer une redirection vers tel endroit.

exemple :

```
//Routes//
app.get("/", (req, res)=> {
  let exemple1 = "https://google.be"
  let exemple2 = "/redirectTest"
  let exemple3 = "/redirectTest/varParams"
  res.redirect(exemple1)
})

app.get("/redirectTest", (req, res) => {
  res.end()
})

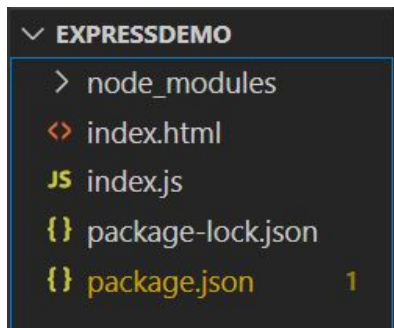
app.get("/redirectTest/:varParams", (req, res) => {
  console.log(req.params)
  res.end()
})
```

Res : Render()

Le **res.render("viewName")** : permet de rendre un template principalement, ou un fichier contenant x chose à rendre.

Attention, plusieurs choses à dire ici.

La première : Vous devez impérativement avoir créer un dossier views à la racine de votre serveur car sinon vous aurez une jolie erreur :

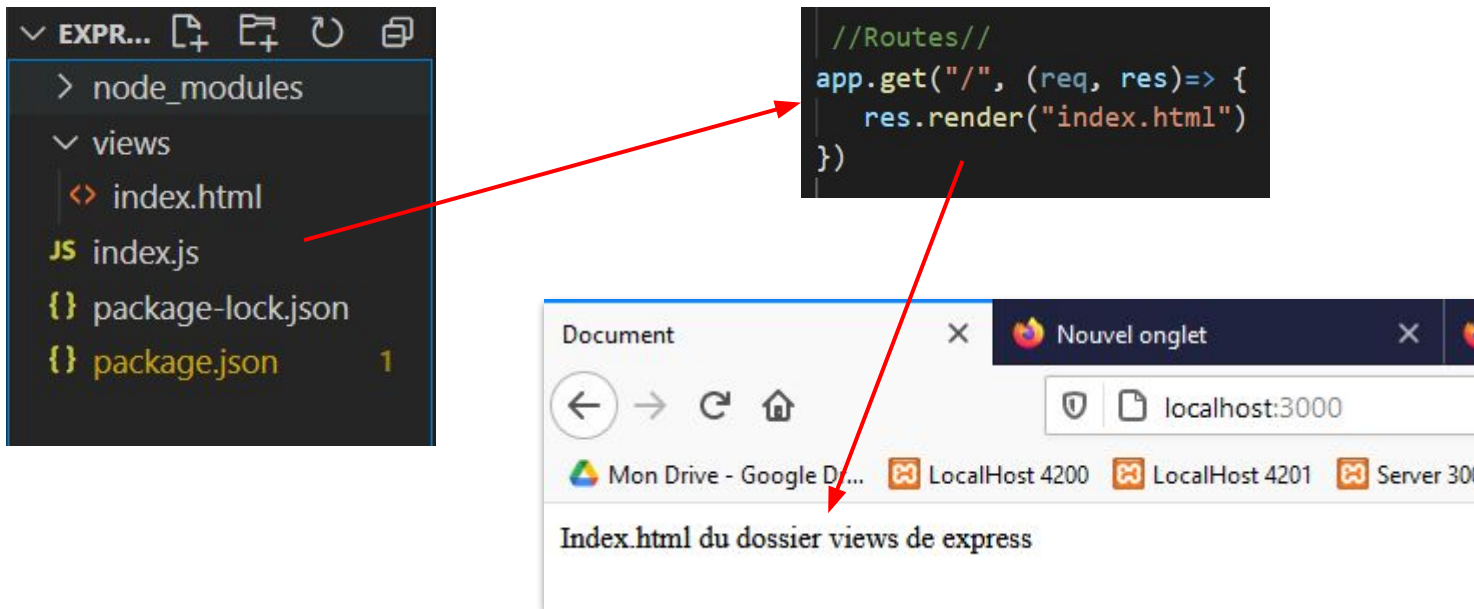


```
//Routes//
app.get("/", (req, res)=> {
  res.render("index.html")
})
```

```
Error: Failed to lookup view "index.html" in views directory "C:\Users\Evengyl\Desktop\Bstorm\2021\NodeJS\DemoETU\ExpressDemo\views"
    at Function.render (C:\Users\Evengyl\Desktop\Bstorm\2021\NodeJS\DemoETU\ExpressDemo\node_modules\express\lib\application.js:580:17)
    at ServerResponse.render (C:\Users\Evengyl\Desktop\Bstorm\2021\NodeJS\DemoETU\ExpressDemo\node_modules\express\lib\response.js:1012:7)
    at C:\Users\Evengyl\Desktop\Bstorm\2021\NodeJS\DemoETU\ExpressDemo\index.js:9:8
    at Layer.handle [as handle_request] (C:\Users\Evengyl\Desktop\Bstorm\2021\NodeJS\DemoETU\ExpressDemo\node_modules\express\lib\router\layer.js:95:5)
    at next (C:\Users\Evengyl\Desktop\Bstorm\2021\NodeJS\DemoETU\ExpressDemo\node_modules\express\lib\router\route.js:137:13)
    at Route.dispatch (C:\Users\Evengyl\Desktop\Bstorm\2021\NodeJS\DemoETU\ExpressDemo\node_modules\express\lib\router\route.js:112:3)
    at Layer.handle [as handle_request] (C:\Users\Evengyl\Desktop\Bstorm\2021\NodeJS\DemoETU\ExpressDemo\node_modules\express\lib\router\layer.js:95:5)
    at C:\Users\Evengyl\Desktop\Bstorm\2021\NodeJS\DemoETU\ExpressDemo\node_modules\express\lib\router\index.js:281:22
    at Function.process_params (C:\Users\Evengyl\Desktop\Bstorm\2021\NodeJS\DemoETU\ExpressDemo\node_modules\express\lib\router\index.js:335:12)
    at next (C:\Users\Evengyl\Desktop\Bstorm\2021\NodeJS\DemoETU\ExpressDemo\node_modules\express\lib\router\index.js:275:10)
```

Res : Render()

Voici comment vous devriez avoir votre organisation vues + rendu (tpl) + serveur



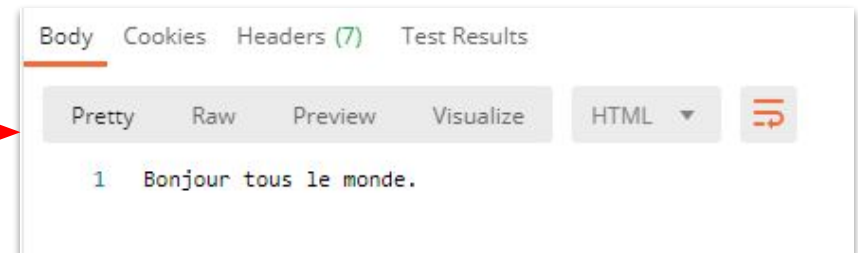
Nous verrons plus en détails les rendus de templates avec EJS plus tard dans ce cours.

Res : send()

le **res.send("string")** → Méthode permettant de répondre à la requête par divers type de donnée brute. aucune mise en forme ni rien si elle n'est pas elle même définie dans ce fameux type de réponse.

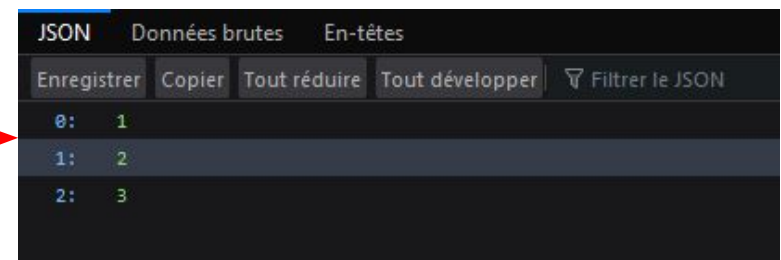
exemple :

```
//Routes//  
app.get("/", (req, res)=> {  
  res.send("Bonjour tous le monde.");  
})
```



Le **.send()** permet également d'envoyer un tableau par exemple, il sera lu comme une sorte de json par le navigateur... **n'adoptez pas cette méthode, utiliser la méthode .json pour faire du renvoi de données.**

```
//Routes//  
app.get("/", (req, res)=> {  
  res.send([1, 2, 3])  
})
```



Exercices

Améliorer le concept de vos routes précédente en y ajouter une liste de route supplémentaire,

Prévoir : Télécharger mon cv, aller voir mon profil linked-in, mon CV en json

Temps accordé : 30 min

Body-parser

Forçons nos requêtes d'entrées à être lisible en Json

Body parser

- Permet de parser les paramètres dans une variable 'req.body' au format clé/valeur
- Retourne null s'il n'y a rien à parser
- Extended: true : permet de récupérer n'importe quel type
- Extended: false : ne passe que des paramètres string ou array (conseillé)

Exemple et détails : prenons le cas d'une api classique qui reçoit un POST, elle est interceptée par app.post, d'accord. Vous ne pourrez **pas** retrouver ce formulaire dans req):

```
[nodemon] restarting due to changes...  
[nodemon] starting `node index.js`  
undefined
```

Nous allons passer par **body-parser**, qui va gérer après configuration la réception de données reçues par **POST**. Comme des formulaires par exemple ?

Aller hop, un petit coup de **npm install body-parser** s'impose...

Body parser

Procédons à l'installation, le **require** et la configuration de body-parser, dans la console node (après avoir coupé le serveur bien-sur, entrez la commande

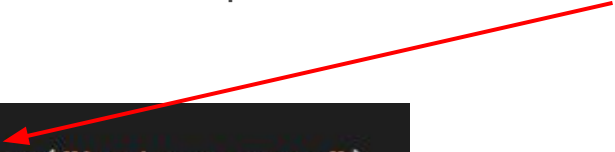
npm install body-parser

```
C:\Users\Evengyl\Desktop\Bstorm\2021\NodeJS\DemoETU\ExpressDemo>npm install body-parser
npm WARN expressDemo@1.0.0 No description
npm WARN expressDemo@1.0.0 No repository field.

+ body-parser@1.19.0
updated 1 package and audited 66 packages in 1.116s
found 0 vulnerabilities
```

Comme pour Express nous allons aller le chercher dans les dépendances installée avec require.

```
var bodyParser = require("body-parser")
```



ps : ici le package est appelé bodyParser par convention de nommage

Body parser

Après cette installation voyons les deux grande étapes de configuration ensemble.

```
app.use(bodyParser.urlencoded({ extended: false }))  
app.use(bodyParser.json())
```

Prenons la première ligne :

```
app.use(bodyParser.urlencoded({extended : false}))
```

Cette ligne signifie : on demande à **express** d'utiliser pour toute **requête (app.use)**, le **body parser (bodyParser.urlencoded)** , en lui spécifiant de ne traduire en **quelque chose** que les objet de type **string** ou **array** et pas le reste (**extended : false**)

Pour la seconde ligne :

```
app.user(bodyParser.json())
```

Cette ligne signifie simplement que l'on demande à **body-parser** de travailler avec le format **json**, bien plus facile pour travailler après dans notre **code**, car il se traduit très facilement en objet **JS**

Body parser

Notre configuration de **body-parser** est maintenant terminée.

ps : ces deux lignes doivent être écrites avant les premières routes, et après le require et l'instance de express...

```
var express = require('express')
var app = express()

var bodyParser = require("body-parser")

app.use(bodyParser.urlencoded({ extended: false }))
app.use(bodyParser.json())
```

ps : il n'est pas très courant de voir autant de lignes de configuration dans **express**... oui...

Aller ! en route pour le passage de données !

Maintenant que ceci est fait regardons de plus près ce que va faire notre **body-parser** sur un **POST** en requête.

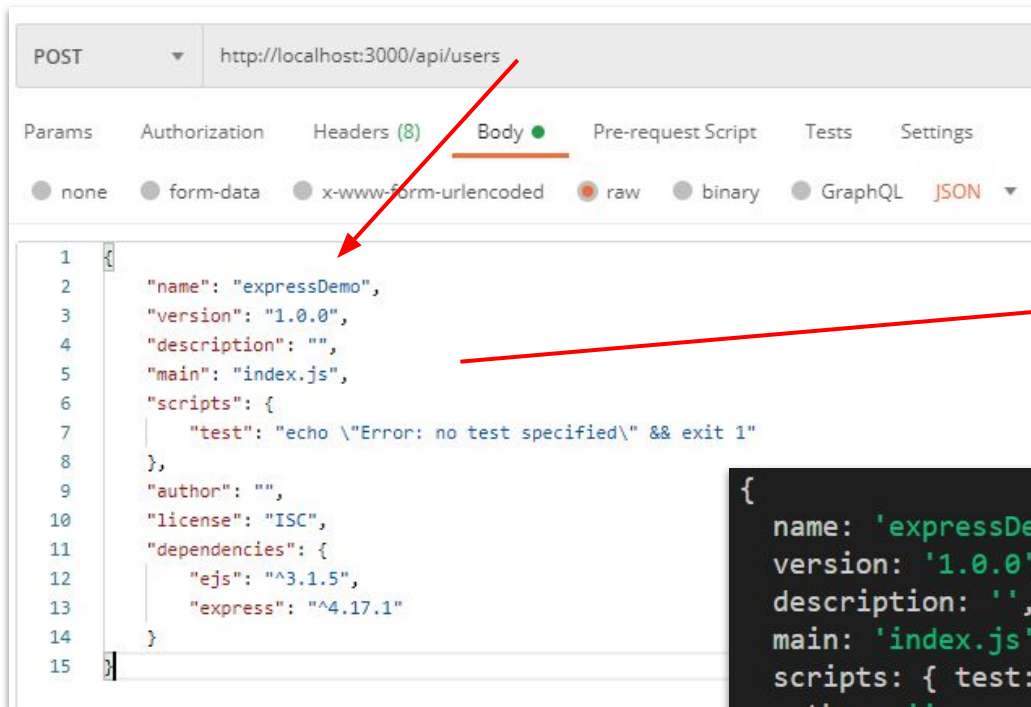
Exemple **sans** body-parser :

```
[nodemon] restarting due to changes...
[nodemon] starting `node index.js`
undefined
```

...déception...

Body parser

Maintenant analysons ce que l'on va avoir après activations de body-parser :



```
// POST /api/users gets JSON bodies
app.post('/api/users', function (req, res) {
  console.log(req.body)
})
```

```
{
  name: 'expressDemo',
  version: '1.0.0',
  description: '',
  main: 'index.js',
  scripts: { test: 'echo "Error: no test specified" && exit 1' },
  author: '',
  license: 'ISC',
  dependencies: { ejs: '^3.1.5', express: '^4.17.1' }
}
```

Exercices

Sur base de vos savoir faire actuelle, travaillez la route de **contact** avec une route de post et de get.

Et avec le body-parser, préparer déjà une réception de formulaire html, que vous afficherez en attendant dans la console du serveur.

Temps accordé : 30 min

Les Templates

Avec Ejs : Embedded JavaScript

[EJS -- Embedded JavaScript templates](#)

Les templates

Enfin! Après avoir injecté de manière horrible de l'html en réponse à nos requêtes nous allons nous tourner vers une solution plus propre.

Comme vu précédemment, nous allons utiliser **EJS** comme langage de template

Il nous faut installer **EJS** avec ***npm install ejs***

Une fois fait, une deuxième étape s'impose: créer un dossier '**views**' à la racine de votre projet.

C'est tout!

```
C:\Users\Evengyl\Desktop\Bstorm\2021\NodeJS\DemoETU\ExpressDemo>npm install ejs
npm WARN expressDemo@1.0.0 No description
npm WARN expressDemo@1.0.0 No repository field.

+ ejs@3.1.5
added 15 packages from 8 contributors and audited 65 packages in 1.833s
found 0 vulnerabilities
```

Les templates

La syntaxe d'EJS:

Comme pour beaucoup de moteurs de vue des tags existent, ils sont la base de ce moteur

Pour EJS, le tag principal est '`<% %>`'

On le trouve sous plusieurs déclinaisons

- `<%` ☐ Contrôle du flow d'instruction
- `<%= ... %>` ☐ Affichage d'une valeur de variable, avec échappement des tags HTML
- `<%-` ☐ Affichage d'une valeur sans échappement, injections de données
- `<%#` ☐ Commentaires
- `<%%` ☐ Permet de sortir '`<%`'

Les templates

Structures Conditionnelles et itératives :

- `If() { } else { }`
- `For(iterator; condition; increment) { }`
- `Variable.Foreach(function(iterator) { })`

L'intérêt des templates est de pouvoir retrouver l'utilisation des inclusions de fichiers au moyen de la méthode '**include()**'.

par exemple : l'**index.ejs**, que nous allons créer, sera le point d'entrée commun à toutes les vues de notre site : donnons lui ce code

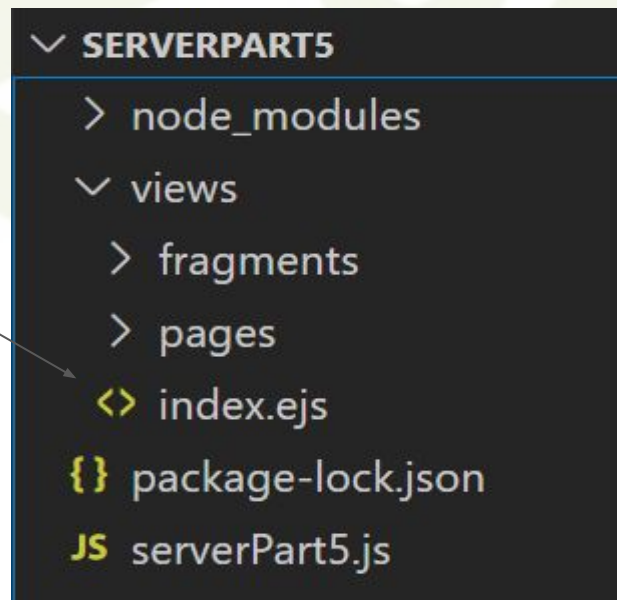
```
<%- include(page); -%>
```

Celui si va attendre de recevoir une variable nommée `page` pour inclure cette variable qui sera en réalité une page html dans nos fichier **.ejs** !

Les templates

Maintenant que nous avons vu les syntaxes et vu les opportunités s'offrant à nous.

Nous pouvons regarder du côté de notre serveur en lui-même pour savoir où et comment intégrer nos vues **“.ejs”**.



Les templates

Dans un premier temps il faut dire au serveur que nous utilisons le middleware ejs

```
app.engine('html', require('ejs').renderFile); // view engine for nodeJS
```

Ensuite dans chaque route ou nous voulons un retour de avec une vues nous ferons au minimum ceci : `res.render("index.ejs")`

Cette ligne nous permet de “rendre” un template **EJS**, la méthode render prendra au minimum un paramètre qui sera en réalité le nom du template à aller chercher par **EJS**, attention nous pouvons constater ici que le template n’est pas sur un chemin relatif, donc **/views/index.ejs** !

D’où l’importance d’avoir créer un dossier **/views**, EJS prendra par défaut ce dossier de templates !

Les templates

Attention que la ligne `res.render("index.ejs").`

Ne prends en charge que l'injection de l'**index.ejs** ! dans notre index .ejs nous lui avons également dit de gérer la variable "**page**", qui elle même sera une autre page **EJS**.

Il faut donc adapter notre cotre d'injection ejs pour lui donner page...

et c'est la que **EJS** est terriblement bien simple, nous n'avons qu'à lui donner un objet de params... comme ceci par exemple ? :)

```
res.render( "index.ejs", { page:"page/home.ejs"} )
```

Les Templates

Mais **EJS** ne s'arrête pas là dans un sens, car il faut bien se dire que l'objet passé en paramètre est un objet qui prends des variables et des valeurs, nous pouvons allègrement lui passé tout ce que nous voulons en terme de donnée, que ce soit un nom de template ou même un demi millions d'enregistrement de base de données :)

Nous pourrions très bien avoir ceci.

```
res.render( "index.ejs", { page:"page/home.ejs", title:"Home page by EJS", allData : DatasDB } )
```

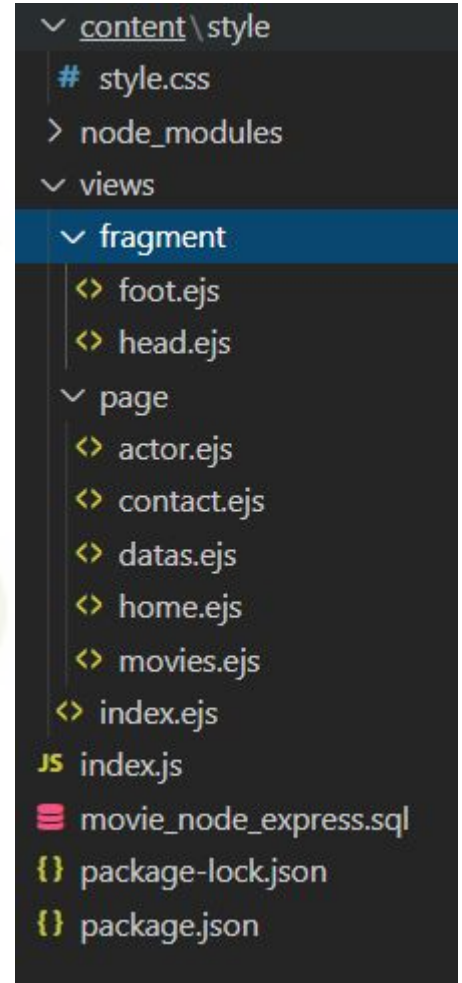

Les Templates

Pour conclure avec **EJS** voici un exemple d'architecture répandue dans les serveurs node particulièrement les web app (au niveau découpage des vue.).

Nous retrouvons un découplage complet des partie de vues.

On constate que l'on se rapproche d'une architecture web traditionnel.

```
<%- include("fragment/header.ejs"); -%>
<%- include(page); -%>
<%- include("fragment/footer.ejs"); -%>
```



Exercices

Maintenant que les templates sont acquis, prenez le temps de voir après un templates pour vos CV, parmi ceux fournis par le formateur.

Découper le template pour y implémenter vos routes. faites correspondre parfaitement votre serveur avec votre html.

Attention, les pages inexistantes de ces templates devront être créées.

Attention, laissez les données factices pour le moment.

Temps accordé : la journée...

Interaction avec MySql

Interaction avec MySql

A l'heure où ces lignes sont écrites, nous sommes en 2021, il n'est que peu probable à cette date de rencontrer un site, une app, un jeu, ne travaillant qu'avec ses données brut de code, par là , nous entendons, sans base de données, ni API (centrale de données directement accessible par une URL, dans les grandes lignes : un cours en est dédiées.).

à l'aube des toujours de plus grande technologies orientées web, on se doit de maîtriser l'appel aux base de données.

Dans ce cours nous verrons l'appel et la consommation de données venant de MySql, nous partons également du principe que vous savez mettre en place la connection à la base de données, le paramétrage et avoir un accès utilisateurs pouvant faire les opérations de CRUD (create, read, update, delete)

Interaction avec MySql

Pour utiliser l'accès à une base de données **MySql** nous allons installer un middleware en plus.

Nous allons utiliser le package **promise-mysql** :

```
C:\Users\Evengyl\Desktop\Bstorm\2021\NodeJS\[WEB] Node JS\02_Express\NJSExpressPlus>npm install promise-mysql
npm WARN njsexpplus@1.0.0 No description
npm WARN njsexpplus@1.0.0 No repository field.

+ promise-mysql@1.3.2
updated 1 package and audited 101 packages in 1.426s
found 1 moderate severity vulnerability
  run `npm audit fix` to fix them, or `npm audit` for details
```

Comme pour le reste des package à utiliser et à installer, il nous faudra le **"require"**

```
var mysql = require('promise-mysql');
```

Interaction avec MySql

Comme pour tout package bien fait il ne nous reste plus qu'une petite phase de configuration et un apprentissage des méthodes d'appel à la base de données.

La configuration se fait comme tel :

createConnection prendra au minimum

4 paramètres, **l'hôte**, **l'utilisateur** utilisé,

sont **mot de passe** et la **base de données** utilisée.

En retour, **MySql** nous fournit un objet de connection, c'est de là que nous pourrons travailler nos query etc...

```
mysql.createConnection({  
  host: 'localhost',  
  user: 'root',  
  password: '',  
  database: 'movie_node_express'  
}).then( (conn) => { connection = conn; } );
```

Promise-MySql ----- CREATE

Pour créer un élément dans la **base de données** nous devons procéder comme suit.

Appel à la méthode de **query**, qui prendra 3 paramètres, le premier est la **query sql** elle même en n'oubliant pas que les valeurs de **set**, seront quand à elle passées en second paramètre sous forme d'objet **JSON**, le troisième paramètre est une méthode de **callback** lorsque l'**insert** sera terminé.

Cette façon de faire sera utilisée partout dans le **process** de **promise-mysql**.

```
app.get("/AddMovies",function(req,res){  
  
    connection.query(  
        'INSERT INTO film SET ?',  
        {  
            title: "test insert",  
            description: "test insert",  
            annee_sortie: "1977-01-01"  
        },  
  
        function(error, results, fields)  
        {  
            // ...  
        }  
    );  
});
```

```
function(error, result, field) { }
```

Mettons un point d'arrêt sur cette fameuse méthode de **callback**.

Instinctivement maintenant vous aurez compris que, **error** contient les informations si il y eu une erreur et de la vous pourrez soulever une **exception** ou autre.

Elle vaut **null** si aucune erreur n'a été détectée.

```
code: 'ER_BAD_FIELD_ERROR',  
errno: 1054,  
sqlMessage: "Unknown column 'titrle' in 'field list'",  
sqlState: '42S22',  
index: 0,  
sql: "INSERT INTO film SET `titrle` = 'test insert', `c`
```

```
Les serveur Express écoute sur le port 3001  
null
```



```
function(error, result, field) { } (insert)
```

Le **callback result** quant-à lui contient une série d'informations intéressante en cas de réussite de la **requête**, ici par exemple pour un **insert** réussi nous avons ceci :


```
OkPacket {  
  fieldCount: 0,  
  affectedRows: 1,  
  insertId: 62,  
  serverStatus: 2,  
  warningCount: 0,  
  message: '',  
  protocol41: true,  
  changedRows: 0  
}
```

Attention que en cas d'erreur la variable **result** ne vaudra que **"undefined"**

function(error, **result**, field) { } (select)

Le **callback result** quant-à lui contient en cas de réussite de la **requête select**, toutes les informations qui ont été sélectionnées sous format JSON :

Attention que en cas d'erreur la variable **result** ne vaudra que **"undefined"**




```
[
  RowDataPacket {
    id: 17,
    title: 'Star wars 2',
    description: "L'attaque des clones",
    annee_sortie: 1979-12-31T23:00:00.000Z
  },
  RowDataPacket {
    id: 18,
    title: 'Star wars 3',
    description: 'La revanche des Siths',
    annee_sortie: 1982-12-31T23:00:00.000Z
  },
  RowDataPacket {
    id: 19,
    title: 'Star wars 4',
    description: 'Un nouvel espoir',
    annee_sortie: 1998-12-31T23:00:00.000Z
  },
]
```

```
function(error, result, field) { }
```

En cas de **SELECT**, il est très intéressant de voir le contenu de la variable **field**.

Elle contient en effet tous les définition en **JSON**, des **colonnes** sélectionnées ainsi que la **db**, la **table**, et tout un tas d'informations pratiques.



```
FieldPacket {  
  catalog: 'def',  
  db: 'movie_node_express',  
  table: 'film',  
  orgTable: 'film',  
  name: 'title',  
  orgName: 'title',  
  charsetNr: 33,  
  length: 150,  
  type: 253,  
  flags: 4097,  
  decimals: 0,  
  default: undefined,  
  zeroFill: false,  
  protocol41: true  
},
```

Promise-MySql ----- READ

Sur base des **informations** que nous avons vues avec le **callback** et l'**insert**, il est assez aisé de partir sur les autres formes du **CRUD**

Ici, le **SELECT** avec deux exemples, un où l'on ne donne pas de paramètres et l'autre où nous donnons deux **id** à récupérer, le "?" permet à **promise-MySql** d'**échapper** les valeurs reçues en paramètres.

Elle prendra un paramètre dans un tableau pour chaque "?" qu'elle rencontrera.

```
// [GET] MoviePage
app.get("/Movies",function(req,res){

  connection.query("SELECT * FROM film" ,function(error, result, fields)
  {
    res.render("index.ejs", { page:"page/movies.ejs", title:"Movies Page", datas : result } )
  })
});
```

```
// [GET] MoviePage
app.get("/Movies",function(req,res){

  connection.query("SELECT * FROM film WHERE id IN (?, ?)", [ 17, 18 ], function(error, result, fields)
  {
    console.log(result)
    res.render("index.ejs", { page:"page/movies.ejs", title:"Movies Page", datas : result } )
  })
});
```

Promise-MySql ----- UPDATE

Ici pour l'exemple du update nous conservons la même forme de **nommage**, sauf que par cette exemple nous montrons que la liste des paramètres donné pour le SET, sera toujours dans le **tableau** de paramètres mais sous forme **d'objet**, ils seront échappé et converti en string pour la **query**.

Attention que les paramètres nommés doivent correspondre au nommage de la base de données.

```
app.get("/UpdateMovies",function(req,res){  
  
  connection.query(  
    'UPDATE film SET ? WHERE id = ?',  
    [{  
      title: "test insert",  
      description: "test insert",  
      annee_sortie: "1977-01-01"},  
      1  
    ],  
  
    function(error, results, fields)  
    {  
      .....  
    }  
  );  
});
```

Promise-MySql ----- DELETE

Ici pour le **Delete** rien ne change de la forme de convention dictée plus haut

```
// [GET] MoviePage
app.get("/DeleteMovies",function(req,res){

  connection.query(
    'DELETE FROM film WHERE title = ?', ["test insert"],

    function(error, results, fields)
    {
      //
    }
  );
});
```

Rappel des règles de nommages

INSERT → **SET ?** → **connection.query("INSERT INTO blabla SET ?", { column : value, column2 : value2 etc...})**

SELECT → **WHERE ? AND ? OR ?** → **connection.query("SELECT blabla FROM bla WHERE id = ? AND title = ?", [value1, value2])**

UPDATE → **SET ? WHERE id = ?** → **connection.query("UPDATE blabla SET ? WHERE id = ?", [{column : value, column2 : value2 etc...}, value3])**

DELETE → **WHERE id = ?** → **connection.query("DELETE FROM blabla WHERE id = ?", [value1])**

Promise-MySql ----- READ multi table ?

Quid d'un read sur des tables liées entre elle par foreign key et par MTO/OTM
MTM par exemple ?

En réalité il n'y a pas de vrai contrainte de conception sur des tables multi foreign
key, il suffira de parfaire notre requête en fonction de.

```
// [GET] DatasPage
app.get("/Datas",function(req,res){
  let actorsList, filmsList;

  connection.query("SELECT * FROM film").then(
    (resultQuery) =>
    {
      filmsList = resultQuery;

      connection.query("SELECT * FROM actor").then(
        (resultQuery) =>
        {
          actorsList = resultQuery;

          res.render("index.ejs", { page : "page/datas.ejs",
                                   title : "Datas Collection",
                                   listFilms : filmsList,
                                   listActors : actorsList } )
        });
      });
  });
});
```


Interaction avec MySql : Astuce

Astuce : pour des raisons de lisibilités dans les slides, les erreurs connues en rouge dans les 3 opérations de crud, n'en sont pas, elle ont été volontairement remplacée par "..."

Ceux-ci qui pourrait être cela à la base :

```
function(error, results, fields)
{
  if(error)
    throw error;
  else
  {
    connection.query("SELECT * FROM film").then(
      (rows) => {
        res.render("index.ejs",
          { page:"page/movies.ejs",
            title:"Movies Page",
            datas : rows}
        )
      }
    );
  }
}
```

Aller plus loins avec Promise-MySql ---- 1 :

La librairies en tant que tel nous permet encore pas mal de choses intéressante, voyons-en quelques unes rapidement

```
connection.query('DELETE FROM film WHERE title = "test insert"', function (error, results, fields) {  
    if (error) throw error;  
    console.log('deleted ' + results.affectedRows + ' rows');  
})
```

-> nous permet par exemple de savoir combien de lignes ont été supprimées lors de notre delete.

Aller plus loin avec Promise-MySql ---- 2 :

```
connection.query('UPDATE film SET ...', function (error, results, fields) {  
    if (error) throw error;  
    console.log('changed ' + results.changedRows + ' rows');  
})
```

-> Nous permet de savoir combien de ligne ont été affectée par notre Update

MySQL®

Exercices

Partant de l'exercices précédent, incorporez toutes vos données dans une base de données.

le choix de la base/tables/colonnes sont à votre goût, ce n'est pas le but du cours.

Le site devra ensuite consommer toutes ces données pour les avoir dans le résultat final de votre HTML.

Temps accordé : 4 heures

Module et exports

Différentes techniques pour découper nos projets

Module et exports

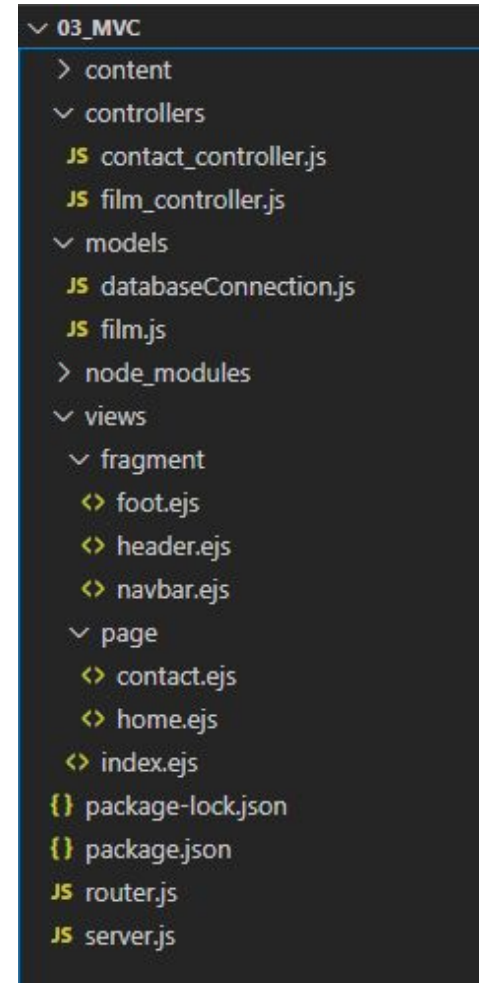
Dans Express, comme dans toute autre app qui se respecte ou tout framework, il est important de rester bien structuré dans nos créations.

Il va donc falloir penser à découper notre code pour qu'il corresponde à une architecture dite "propre" pour le déploiement ou le partage de travail, ou simplement pour ne pas s'y perdre quand l'on reviendra dedans plus tard, ou encore même tout simplement pour ne pas s'y perdre tout court....

Pour ce faire nous devons voir les formats de découpage en JS

Il existe plusieurs façons de séparer du code en plusieurs fichiers.

Nous allons en voir 3.



Module et exports

La première consiste à exporter une série de fonctions nommées, déclarées dans une **variable**.

Cette liste de **fonctions** sera ensuite exportée grâce à :

module.exports = NameVarFct

Elle sera ensuite récupérée dans le fichier voulu, avec un **require**.

Qui initialisera une variable avec le contenu du **exports**.

Cette variable déclarée fonctionnera avec appel des fonctions stockées dans le **fichier** précédemment créé.

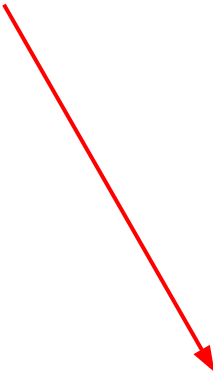
```
var listFct = {  
  info : function(){  
    console.log("Info !!")  
  },  
  
  warning : function(){  
    console.log("Warning !!")  
  },  
  
  error : function(){  
    console.log("Error !!")  
  }  
}  
  
module.exports = listFct
```

```
var listFct = require("./DemoExport.js")  
  
listFct.info();  
listFct.warning();  
listFct.error();
```

Module et exports

La méthode suivant consiste à exporter une fonction anonyme ou fonction fléchée.

Le simple fait de l'importer avec require, permettra à var variable initialisée de s'auto executée lors de son appel, nous avons exporter une méthode anonyme, elle se lancera donc sur son hôte directement en mettant les parenthèse d'appel de fonction.



```
module.exports = function (){  
  ...  
  console.log("Yeah anonymous !")  
}
```

```
var Anos = require("./DemoExport.js")  
Anos()
```


Module et exports

La troisième technique consiste à **exporter** uniquement une **fonction fléchée** mais en la nommant au préalable dans **l'exports** lui même.

```
exports.TestExp = () => {  
  ...  
  console.log("Other exports method")  
}
```

Elle pourra être appelée directement par son nom dans la **variable** initialisée lors du **require**.

```
var DemoExport = require("./DemoExport.js")  
DemoExport.TestExp()
```

Module et exports

La dernière façon de faire et par la même occasion une des plus utilisée sur le **web express**.

Vous devrez créer une **class js** et l'exporter directement.

le **require** en résultant sera l'**objet** en lui même, il faudra donc en créer une **instance** pour qu'il puisse fonctionner.

```
module.exports = class TestClass{  
  ...  
  constructor(test1, test2){  
    this.test1 = test1  
    this.test2 = test2  
  }  
  
  display(){  
    console.log(this.test1 + " " + this.test2)  
  }  
}
```

```
var testClass = require("../DemoExport.js")  
var TestClass = new testClass("tata", "tatat")  
TestClass.display()
```

ps : ce cours n'a pas pour but de vous montrer comment réaliser des class js..

Exercices

Sur base de ce que vous avez appris dans ce module ;

Il n'y a pas d'exercice, vous devrez terminer le module App.router d'abord

PS : Attention tout de même, si vous exporter des composants de votre application dans d'autres fichier, vous devrez impérativement faire attention à ce que les variable / middleware utilisé soit inclus également dans ce fichier peu importe le moyen, "require" ou par passage de paramètre.

App.router Express

Le router surpuissant de express

Exercices

Les variables d'environnement

Avec .env et npm dotenv

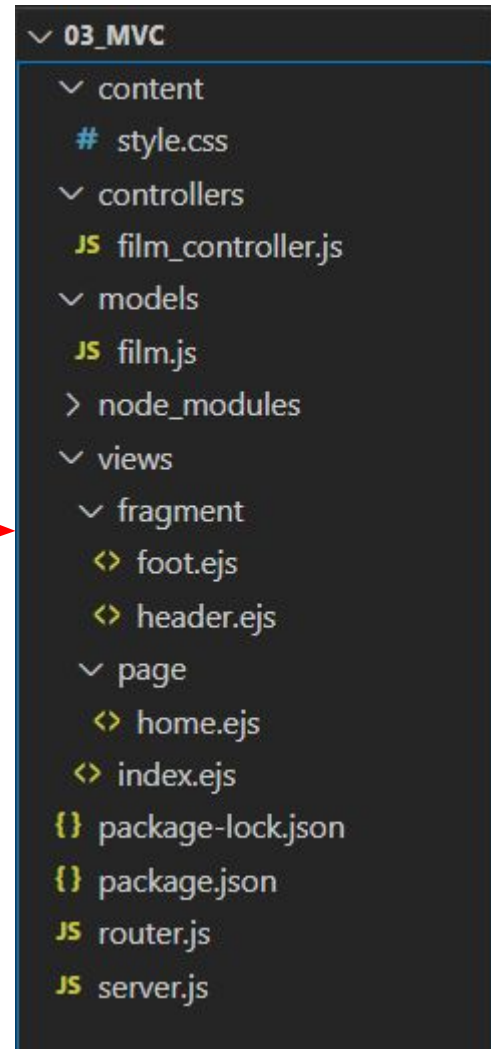
Découpage MVC

Pattern Modèle Vue Contrôleur

Découpage MVC

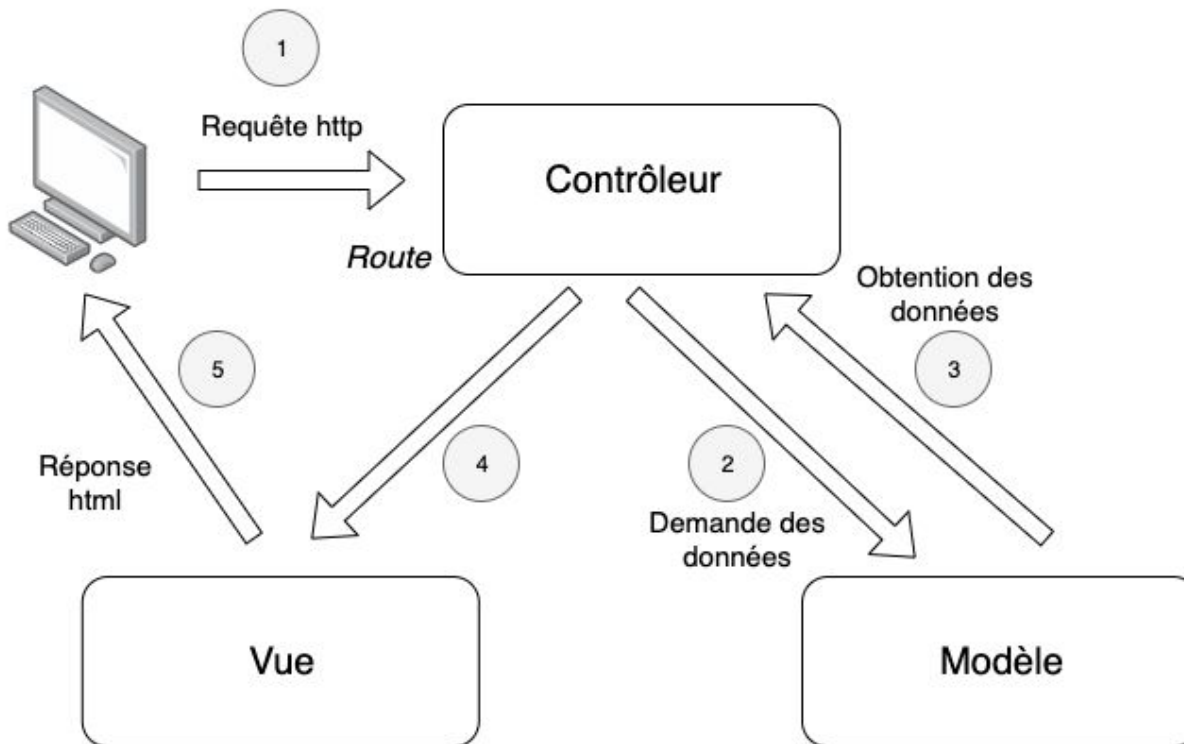
Pour continuer dans la découpes de notre fichier de serveur, nous allons maintenant découper nos **middleware** pour avoir un **routeur** à part, le **contrôle** de ces **routes** à part, nos **modèles**, et nos **vues** bien séparées, histoire de rendre l'architecture beaucoup plus propre, comme ceci.

Nous allons voir dans les slides suivants, le concept **MVC** et ce que cela implique.



Découpage MVC

Le **concept**, ou même le **pattern de conception**, ou encore le **patron de construction de structure MVC**, veux dire, **Modèle Vues Contrôleurs**.



Découpage MVC

Détaillons le **schéma** ensemble.

Pour un découpage propre et réutilisable, tout en restant stable dans le temps et d'une très grande réputation, nous allons procéder comme suit.

Nous allons répartir notre serveur entre plusieurs sources de travail.

- La première que nous avons déjà faite c'est le routage de Express avec une petite particularité.
- La seconde sera les modèles de nos données, avec les appel à MySql.
- La troisième sera plus communes, les contrôleurs, tout ce qui ne concerne plus les route, les datas et les vues, sera simplement la couche de contrôle, celle qui contiendra en théorie tous nos algorithmes.

Découpage MVC

Découpage MVC

Découpage MVC

Découpage MVC

Découpage MVC

Découpage MVC

Exercices

Axios

ex request ; Ou comment faire des requêtes http
dans des requêtes express

```
const axios = require("axios");//voir https://www.npmjs.com/package/axios
```

```
app.get("/", (req, res, next) => {  
  axios.get("http://restcountries.eu/rest/v2/all")  
    .then((response) => {  
      res.json(response.data)  
    })  
    .catch((error) => {  
      console.log(error)  
    })  
    .then(() => {  
    })  
})
```

Exercices

Session

Avec express-session

Session

- Vous aurez besoin du middleware '**express-session**'
- Vous aurez besoin du middleware '**body-parser**'
- Particularité : ici vous entrerez une clé secrète pour générer des **session ID** aléatoire sur base de **votre clé**.

```
app.use(session( { secret: "KJhklzab1&ckz@654654tartenpion",  
                  cookie: { maxAge : (3600 * 1000)} //en milli sec donc la 1 heure  
                  }));
```

```
var bodyParser = require('body-parser');  
app.use(bodyParser.json()); // support json encoded bodies  
app.use(bodyParser.urlencoded({ extended: true })); // support encoded bodies for form http post
```

Session

- Quand vous vous logerez, le **middleware** de **session** saura différencier les différents **clients**.
- Exemple sur deux navigateurs différents.
- Le **sessionID** sera généré sur base de votre **clé secrète**
- Vous récupérerez les valeurs au moyen de **req.session.mavARIABLE**

```
sessionID: 'ykMyo_3FQrAiUFgRuBKGTvVXKwFDVXJD',  
session: Session { cookie: [Object], username: 'Evengyl' },  
route: Route { path: '/Actor', stack: [Array], methods: [Object] },  
[Symbol(kCapture)]: false
```

```
sessionID: 'RyOuAekxOQK5oyX5HyZCjRiDgccFoIvW',  
session: Session { cookie: [Object], username: 'test' },  
route: Route { path: '/Actor', stack: [Array], methods: [Object] },  
[Symbol(kCapture)]: false
```

Exercices

Data-Table JQuery

Organisons nos listes de données

DataTable JQuery

Que préférez-vous ?

Ceci ?

Movies List

Show entries

Search:

Titre	Sous-titre	Date de sortie
Star wars	The clone wars	Tue Jan 01 2008 00:00:00 GMT+0100 (GMT+01:00)
Star wars	Rogue one	Fri Jan 01 2016 00:00:00 GMT+0100 (GMT+01:00)
Star wars	Solo	Mon Jan 01 2018 00:00:00 GMT+0100 (GMT+01:00)
Star wars	Rogue Squardon	Sun Jan 01 2023 00:00:00 GMT+0100 (GMT+01:00)
Star wars 1	La menace fantôme	Sat Jan 01 1977 00:00:00 GMT+0100 (GMT+01:00)
Star wars 2	L'attaque des clones	Tue Jan 01 1980 00:00:00 GMT+0100 (GMT+01:00)
Star wars 3	La revanche des Siths	Sat Jan 01 1983 00:00:00 GMT+0100 (GMT+01:00)
Star wars 4	Un nouvel espoir	Fri Jan 01 1999 00:00:00 GMT+0100 (GMT+01:00)
Star wars 5	L'empire contre attaque	Tue Jan 01 2002 00:00:00 GMT+0100 (GMT+01:00)
Star wars 6	Le retour du Jedi	Sat Jan 01 2005 00:00:00 GMT+0100 (GMT+01:00)

Showing 1 to 10 of 13 entries

Previous **1** 2 Next

DataTable JQuery

Ou ceci...

Movies List

Titre	Sous-titre	Date de sortie
Star wars 1	La menace fantôme	Sat Jan 01 1977 00:00:00 GMT+0100 (GMT+01:00)
Star wars 2	L'attaque des clones	Tue Jan 01 1980 00:00:00 GMT+0100 (GMT+01:00)
Star wars 3	La revanche des Siths	Sat Jan 01 1983 00:00:00 GMT+0100 (GMT+01:00)
Star wars 4	Un nouvel espoir	Fri Jan 01 1999 00:00:00 GMT+0100 (GMT+01:00)
Star wars 5	L'empire contre attaque	Tue Jan 01 2002 00:00:00 GMT+0100 (GMT+01:00)
Star wars 6	Le retour du Jedi	Sat Jan 01 2005 00:00:00 GMT+0100 (GMT+01:00)
Star wars 7	Le réveil de la force	Thu Jan 01 2015 00:00:00 GMT+0100 (GMT+01:00)
Star wars 8	Les derniers Jedi	Sun Jan 01 2017 00:00:00 GMT+0100 (GMT+01:00)
Star wars 9	L'ascension de Skywalker	Tue Jan 01 2019 00:00:00 GMT+0100 (GMT+01:00)
Star wars	The clone wars	Tue Jan 01 2008 00:00:00 GMT+0100 (GMT+01:00)
Star wars	Rogue one	Fri Jan 01 2016 00:00:00 GMT+0100 (GMT+01:00)
Star wars	Solo	Mon Jan 01 2018 00:00:00 GMT+0100 (GMT+01:00)
Star wars	Rogue Squardon	Sun Jan 01 2023 00:00:00 GMT+0100 (GMT+01:00)

DataTable JQuery

Je me doute de la réponse, pour avoir ces sort-list de datas, search-bar, etc etc, il vous suffit dans le head dans la partie **html** d'implémenter ceci:

```
<script src="https://cdn.datatables.net/1.10.11/js/jquery.dataTables.min.js"></script>
```

En plus de bootstrap et jquery de base bien sur..

Il vous faudra de base une table correctement constituée, en bas de ce Template vous pourrez ensuite ajouter ces lignes de JavaScript pour activer le data table de jquery-datatable, elle ne prends que l'id en paramètre de jQuery et la magie opère.

Attention : des dizaines d'autres options sont disponibles pour la configurations de dataTable. Voir la doc.



```
<script type="text/javascript">
  $(document).ready(function()
  {
    $('#table').DataTable();
  });
</script>
```

Exercices

Sequelize

L'Orm De Express

Object-Relational Mapping

Ou comment s'abstraire de la base de données MySql

Exercices

Socket IO

Le Web Socket préféré de Express
Ou comment faire du real-time avec Express

Exercices

FireBase

Le services websocket de google
Bsd real-time, Auth, Storage

Exercices

Sécurisons nos App

Exercices

NodeMailer

Envoyer des Emails avec Node Express

Si Gmail utilisé alors il faut activer l'envoi par application tierces

<https://myaccount.google.com/u/1/lesssecureapps>

Exercices

Déploiement

Le Déploiement

L'exemple de mise en production avec déploiement se fera avec l'exemple du formateur Loïc - Bstorm. Grâce à son hébergement o2switch.

Attention. les slides suivants vous permettront de comprendre certaine chose, mais tous les cas différent.

Vous ne verrez ici que des bonnes pratiques dans le déploiement.

Exerices

Il ne vous reste plus qu'à payer un hébergement et de mettre en production :)

Non je rigole...

Temps accordé : infinite

Merci pour votre attention