INTRODUCTION

Démarrage d'un serveur basique :

- fichier server.js :

```
const http = require('http');
const server = http.createServer((req, res) => {
   res.end('Voilà la réponse du serveur !');
  });
  server.listen(process.env.PORT || 3000);
```

Démarrage du server :
- Exécuter : **`node server`**
- Accéder : http://localhost:3000

Installation nodemon :
- Exécuter : **`npm install -g nodemon`**
- Maintenant, au lieu d'utiliser node server, on peut utiliser nodemon server
- Il surveillera les modifications des fichiers et redémarrera le serveur lorsqu'il aura besoin d'être mis à jour. Cela garantit d'avoir toujours la dernière version du serveur dès qu'on sauvegarde, sans devoir relancer manuellement le serveur.

CREATION D'UNE APPLICATION EXPRESS

Coder des serveurs web en Node pur est possible mais long. Cela entraine l'utilisation du Framework Express.

Express est un module node JS permettant de développer un serveur web.

Il permet de coder les traitements à faire par votre serveur quand il reçoit des requêtes HTTP

Il est léger, rapide et robuste !

Rappels sur le http – URL

HTTP est un protocole requête / réponse

Le client envoie une requête (à une url), le serveur lui envoie une réponse

L'URL est constituée de plusieurs parties:
- Protocol
- Localisation
  - nom_du_serveur:port
  - /chemin/de/la/ressource
  - ?argument=valeur&arg=val
- #Données qui restent pour le client

http://www.monserver.com:8080/perso/data?lg=fr&doc=json#contenu

Express vous permet de démarrer un serveur et qui écoute les requêtes HTTP et qui y répond

Pour la réception

- Il décode les requêtes HTTP
- Il vous permet d'associer du code en fonction de la requête
- Dans le code il vous donne accès aux paramètres, aux headers et au body

Pour la réponse

- Il vous permet de construire une réponse (status, header et body)
- Et de l'envoyer vers le client

Et surtout il permet de structurer votre code sous forme de chaîne de traitements (middleware)

Installation :

- A partir de backend : **`npm install express`**

Une fois installé il faut l'exploiter comme un module classique (3 étapes)

1. **var express = require('express')** //charger le module
2. **var app = express()** //construire une application Express, ce qui permettra d'agir sur elle
3. **var server = app.listen(8080)** //démarrer l'application, ici sur le port 8080

L'application est démarrée mais elle ne fait rien car elle n'a aucun code de traitement !

Exécution de l'application Express sur le serveur Node :

- Dans le fichier server.js :

```
const http = require('http');
const app = require('./app');

app.set('port', process.env.PORT || 3000);
const server = http.createServer(app);

server.listen(process.env.PORT || 3000);
```

- Effectuer une demande vers ce serveur générera une erreur 404. Configurons alors une réponse simple pour assurer que tout fonctionne correctement, dans le fichier app.js :

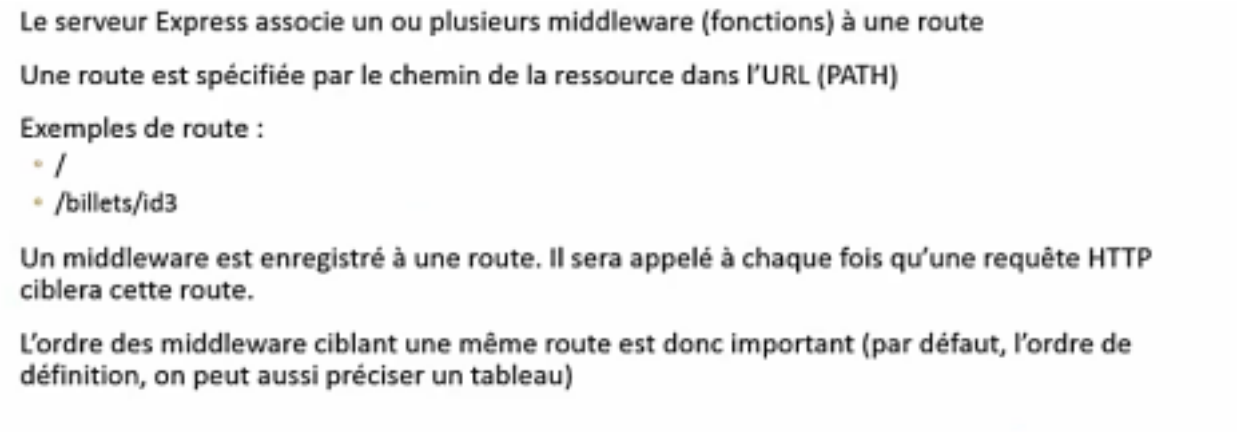
```
const express = require('express');

const app = express();

app.use((req, res) => {
 res.json({ message: 'Votre requête a bien été reçue !' });
});

module.exports = app;
```

LA ROUTE : LE CHEMIN VERS LES MIDDLEWARES



Des middlewares :

Une application Express est fondamentalement une série de fonctions appelées *middleware*. Chaque élément de *middleware* reçoit les objets request et response, peut les lire, les analyser et les manipuler, le cas échéant. Le *middleware* Express reçoit également la méthode next, qui permet à chaque *middleware* de passer l'exécution au *middleware* suivant.

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

const app = express();

app.use((req, res, next) => {
 console.log('Requête reçue !');
 next();
});


app.use((req, res, next) => {
 res.status(201);
 next();
});

app.use((req, res, next) => {
 res.json({ message: 'Votre requête a bien été reçue !' });
 next();
});

app.use((req, res, next) => {
 console.log('Réponse envoyée avec succès !');
});

module.exports = app;
```

Cette application Express contient quatre éléments de middleware :

- le premier enregistre « Requête reçue ! » dans la console et passe l'exécution ;
- le deuxième ajoute un code d'état 201 à la réponse et passe l'exécution ;
- le troisième envoie la réponse JSON et passe l'exécution ;
- le dernier élément de middleware enregistre « Réponse envoyée avec succès ! » dans la console.

Il s'agit d'un serveur très simple et qui ne fait pas grand-chose pour l'instant, mais il illustre comment le middleware fonctionne dans une application Express.

La route et le Path :

Pour associer des middlewares aux routes :

`use([PATH], middleware[,middleware])`

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

app.use('/' , function (req, res, next) {
    res.send('HELLO')
})
```

La fonction use() prend comme premier argument le chemin de la route (PATH)

Il est possible de donner le chemin static ou de donner des patterns de chemin

- app.get(**path**, function(req, rep) {})

Avec path=

- '/'
- '/tmp/test.txt'
- '/tmp/*.txt'
- '/tmp/.fly$/'
- '/tmp/:id'
  - req.params.id

express.static :

C'est un middleware permettant de renvoyer des fichiers correspondant au chemin de l'URL
Ex : http://www.server.com:3000/tmp/test.txt ➔ retournera le fichier ./tmp/text.txt

Mettez express.static(root) comme middleware dont root est le répertoire racine où sont stockés les fichiers.
Ex : **`app.use(express.static('tmp'))`**

Requete et réponse :

Express vous permet d'obtenir pleins d'information sur la requête (req)

- req.ip
- req.baseURL
- req.body
- req.params
- req.query
- req.method

Et de préciser la réponse (res)

- res.send()
- res.end()
- res.render()
- res.status()

Gérer les méthodes HTTP :

Il est souvent utile de filtrer la méthode HTTP (GET, POST, etc.) au delà de la route. D'où l'exploitation de req.method

Pour faciliter ce traitement, express propose des fonctions pour chaque méthode HTTP

- app.get(), app.post(), app.put(), etc.

Ces méthodes ont deux paramètres : la route (PATH) et une fonction callback qui sera appelée lors d'une réception d'une requête (function(req,res[,next]))

Ex:

- **app.get('/', function(req, res) { res.send('OK') }**

SYNTHESE :

```
// app.js
const express = require('express');

const app = express();

app.use('/publics',express.static("./publics"));

app.get('/',(req, res) => {
  res.send('Essayer l url publics');
});

module.exports = app;

/******** server.js ********/
const http = require('http');
const app = require('./app');

app.set('port', process.env.PORT || 3003);
const server = http.createServer(app);

server.listen(process.env.PORT || 3003, function() {
    console.log('Serveur démarré')
});
```

Modification :

```
// app.js
const express = require('express');

const app = express();
app.set("view engine", "ejs");
app.set("views", "./views");

app.use('/publics',express.static("./publics"));

app.get('/',(req, res) => {
  res.render('index', {test : 'bonjour'})
});

module.exports = app;
```

```
> backend > views > <> index.ejs > ...
<html>
    <head></head>
    <body>
        Bonjour Publics
        <p> <%= test %> </p>
    </body>
</html>
```

Installation body-parser :

- Exécuter : **`npm install body-parser`**

EXEMPLE :

```
// app.js
let express = require('express');
let bodyparser = require('body-parser');

let app = express();
app.set("view engine", "ejs");
app.set("views", "./views");

app.use('/publics',express.static("./publics"));
app.use(bodyparser.urlencoded({extended : false}))
app.use(bodyparser.json())

app.get('/',(req, res) => {
  res.render('index', {test : 'TEST PAGE', text: "" })
});

app.post('/',(req,res) => {
  console.log(req.body.message);
  if(req.body.message === undefined || req.body.message === ''){
    res.render('index', {test : 'TEST PAGE', text: "pas de message" })
  }
  else{
    res.render('index', {test : 'TEST PAGE', text : req.body.message })
  }
});
```

```
> backend > views > <> index.ejs > ...
<html>
    <head>
        <title><%= test %></title>
        <link rel="stylesheet" href="/publics/styles.css"></Link>
    </head>
    <body>
        <a href="/">Accueil</a> <br>
        <%= text %>
        <form action="/" method="post">
            <label for="message">message</label> <br>
            <textarea name="message" id="message" cols="30" rows="10"></textarea> <br>
            <button type="submit"> Envoyer </button>
        </form>
    </body>

</html>
```

TP FORMULAIRE D'INSCRIPTION.

Créer en HTML un formulaire d'inscription comportant plusieurs champs dont au moins un nom, login, mdp et

confirmation mdp. Le but de cet exercice est de faire en sorte que le serveur vérifie que le mot de passe est correct, et qu'il réponde un message du type "Bonjour Prenom Nom, ton compte est bien créé".

Faire en sorte que la soumission du formulaire provoque l'envoi (submit) des réponses à la page http://localhost:3003/ a l'aide de la méthode POST.

a. Créer un serveur express et y ajouter une route POST qui récupère les champs du formulaire. b. Faire en sorte que cette route renvoie un message de confirmation d'inscription ou d'erreur selon que le mot de passe soit bien duplique ou non. En cas d'erreur l'utilisateur est invite à retaper son mot de passe dans le formulaire (sans que les autres champs soient effacés).