Node JS - Activités N01

▶Activité N01

◇Application pratique

```
const express = require("express"); // import du framework
const app = express(); // creation d'une instance

// methode get() pour envoyer une requette http de type `GET`
// definir une route '/' (root) et envoi de message
app.get("/", (req, resp) => {
resp.send("Bienvenu sur mon premier serveur Express !");
});

// Le serveur écoute sur le port 3000
app.listen(3000, () =>
  console.log("Serveur en ecoute sur http://localhost:3000")
);
```

◇Réponses aux questions

1. *Quelle différence avec ton serveur `Node.js` natif (Semaine 1) ?*
   - `NodeJS` est un environnement d'execution de javascript
   - `Express JS` est un framework back-end qui facilite la creation et le développement des projets backend, il fournir des fonctionnalité prêt a être utilises ...
2. *Qu'est-ce qu'Express gère automatiquement pour toi ?*
   - La Gestion des requêtes et des réponses HTTP, l'exécution des fonctions intermédiaires (middlewares)

▶Activité N02

◇Application pratique

```
// Création des routes
// Méthode `GET` pour renvoyer une liste de produits
app.get('/api/products',(req,res) => {
  res.json([{id:1,name:'Laptop'},{id:2,name:'Phone'}])
})

// Méthode `GET` pour renvoyer un produit spécifique selon id
app.get('/api/products/:id',(req,res)=>{
  res.json({message:`Produit ${req.params.id}`})
})
```

```
curl http://localhost:3000/api/products
curl http://localhost:3000/api/products/2
```

◇Réponses aux questions

1. *Qu'apporte le format `JSON` ?*
   - JavaScript Object Notation est un format léger et lisible pour échanger les données d'une manière structurées.
2. *Pourquoi chaque ressource a-t-elle sa propre route (/api/products/:id) ?*
   - chaque ressource possède un endpoint unique, pour qu'on puisse l'accéder, on doit spécifier le endpoint du API (ex : soit tous les produits ou just un produit spécifique)

▶Activité N03

◇Application pratique

```
// Middleware pour loguer des infos sur les requêtes
app.use((req,res,next)=>{
  console.log(`[${new Date().toISOString()}] ${req.method} ${req.url}`);
  next();
})

// Middleware pour mesurer le temps d'exécution d'une requête
app.use((req,res,next)=>{
  re.startTime=Date.now();
  next();
})

// Middleware pour afficher la durée de la requête
app.use('/ping',(req,res) =>{
  const duration=Date.now() - req.startTime;
  res.json({message: 'pong', duration: `${duration}ms`})
})
```

◇Réponses aux questions

1. *Que fait next() ?*
   - utilisé pour faire passer au middleware suivant.
2. *Quelle est la différence entre un* `middleware` *global et spécifique à une route ?*
   - Global : s'exécute pour toutes les routes de l'application.
   - Spécifique à une route : ne s'exécute que pour une route donnée.
3. *Pourquoi les* `middlewares` *sont essentiels dans Express ?*
   - Pour faciliter la gestion des logs, l'authentification, validation des données, gestion des erreurs ...

▶Activité N04

◇Application pratique

```
// Route pour simuler une erreur
app.get('/api/crash',(req,res,next)=>{
  const err=new Error('Erreur simulee ');
  next(err); // pour passer l'erreur au middleware d'erreur
})

// Middleware d'erreur pour intercepter et gérer les erreurs
app.use((err,req,res,next)=>{
  console.log('Erreur detectee :', err.message);
  res.status(500).json({error: err.message})
})
```

```
curl http://localhost:3000/api/crash
```

◇Réponses aux questions

1. Pourquoi le `middleware` d'erreur doit-il être placé en dernier ?
   - Parce qu'il s'exécute selon l'ordre où les middlewares sont déclarés.
2. Quelle différence entre `throw new Error()` et `next(err)` ?
   - throw new Error() : pour une erreur JavaScript classique.
   - next(err) : envoie l'erreur au middleware d'erreur d'Express.

▶Activité N05

◇Application pratique

`public/index.html`

```
<!DOCTYPE html>
 <html lang="en">
 <head>
   <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width, initial scale=1.0">
   <title>Document</title>
 </head>
 <body>
  <h1>Bienvenue sur le serveur Express</h1>
 </body>
</html>
```

`server.js`

```
app.use(express.static('public'))
```

`data/products.json`

```
[
  {"id":1,"name":"Laptop"},
  {"id":2,"name":"Headphones"}
]
```

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

app.get('/api/products',(req,res)=>{
  const data=fs.readFileSync('./data/products.json')
  const products = JSON.parse(data);
  res.json(products);
})
```

◇ Réponses aux questions

1. Quelle est la différence entre servir un fichier statique et lire un fichier JSON ?
   - fichier static :
   - lire un fichier json :
2. Pourquoi readFileSync n'est pas recommandé en production ?
   - parce qu'il bloque le thread principal de Node.js pendant la lecture du fichier, ce qui empêche le serveur de répondre à d'autres requêtes pendant ce temps.
3. Comment pourrait-on utiliser fs.promises pour une version asynchrone ?
   - en utilisant `async` / `await`