Activité d'apprentissage 01

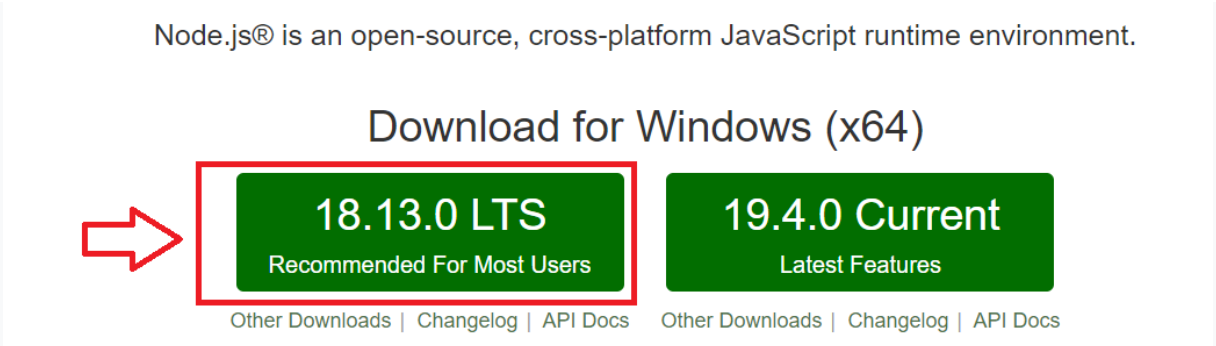
Objectifs attendus :

- Introduction au NodeJS
- Installation de NodeJS et NPM
- Node Package Module (NPM)
- Package.json
- Exemple 1

1- Installation NodeJS et NPM

L'installation de NodeJS et NPM est simple à l'aide du package d'installation disponible sur le site Web officiel de NodeJS.

node.js	https://nodejs.org/fr/



Maintenant, testez NodeJS en affichant sa version à l'aide de la commande suivante dans l'invite de commande :

node -v	npm -v

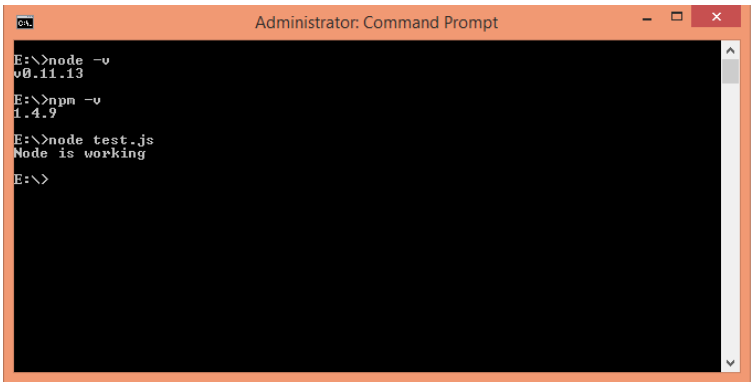
Exemple 1 :

Un moyen simple de tester le travail de nodeJS dans votre système consiste à créer un fichier javascript qui imprime un message.

Créons le fichier **test.js**

```
console.log("Node is working");
```

Exécutez le fichier test.js à l'aide de la commande Node **> node test.js** dans l'invite de commande.



Exemple 2 : Node Package Module

NPM est le module de package qui aide les développeurs javascript à charger efficacement les dépendances. Pour charger les dépendances, il suffit d'exécuter une commande dans l'invite de commande :

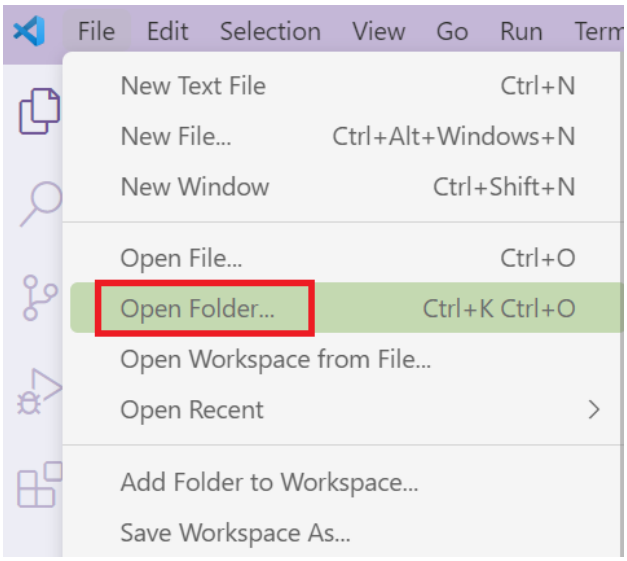
C:\Users\said_\Desktop\2022-2023\M206\demo01> **npm init -y**

NPM fournit de nombreux scripts utiles comme

`npm install`, `npm start`, `npm stop`

Exemple de base

1- Créer un projet nommé « demo01 »



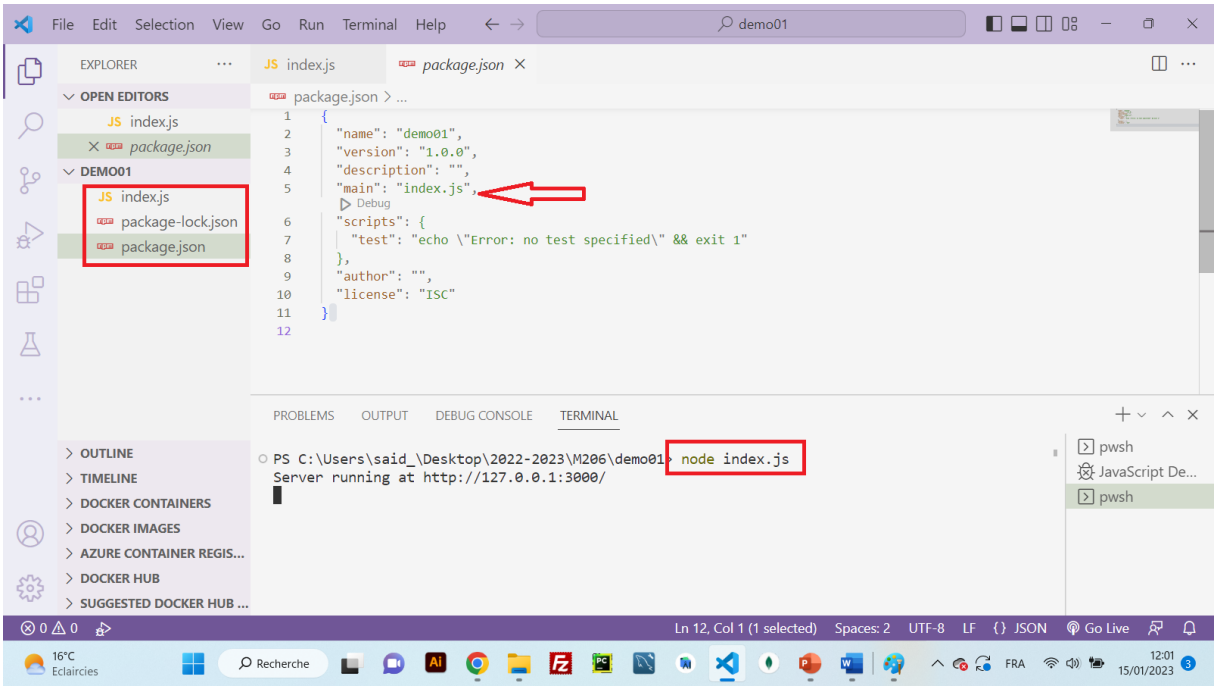
2- Initialiser le projet grâce à la commande :

C:\Users\said_\Desktop\2022-2023\M206\demo01> npm init -y

3- Ajouter un fichier index.js pour créer un serveur http :

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

const hostname = '127.0.0.1';
const port = 3000;

const server = http.createServer(function(req, res) {
  res.statusCode = 200;
  res.setHeader('Content-Type', 'text/plain');
  res.end('Hello World\n');
});

server.listen(port, hostname, function() {
  console.log('Server running at http://'+ hostname + ':' + port + '/');
});
```

http : est un module nodeJS qui permet de créer un serveur



4- Exécuter la commande suivante pour démarrer le serveur :

C:\Users\said_\Desktop\2022-2023\M206\demo01> **node index.js**

Activité d'apprentissage 02

Objectifs :

- **Installer Framework expressJS**
- **Créer API qui permet de récupérer la liste des produits**

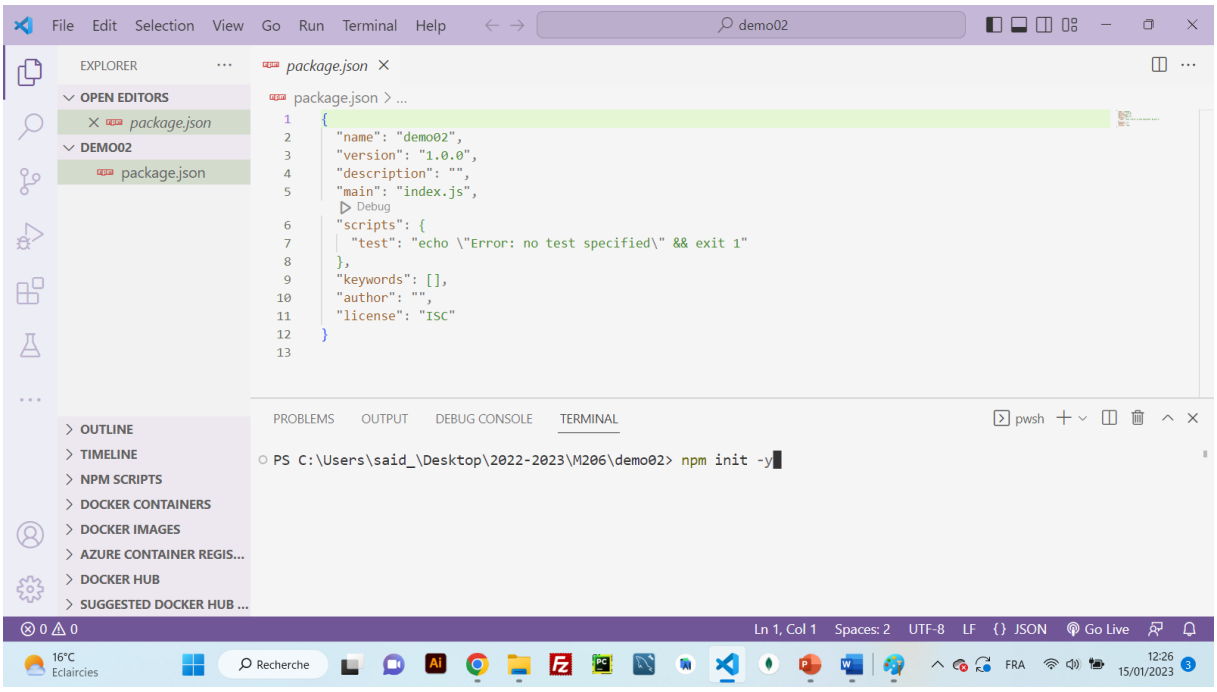
PS C:\Users\said_\Desktop\2022-2023\M206> mkdir demo02

PS C:\Users\said_\Desktop\2022-2023\M206> cd demo02

PS C:\Users\said_\Desktop\2022-2023\M206\demo02> npm init -y

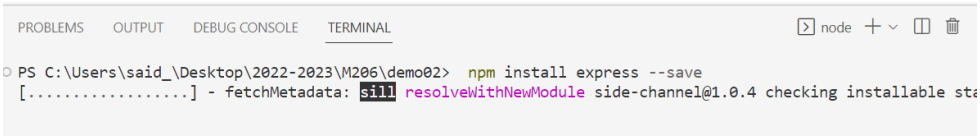
Cette commande vous invite à fournir un certain nombre d'informations, telles que le nom et la version de votre application. Pour le moment, vous pouvez simplement appuyer sur la touche RETURN pour accepter les valeurs par défaut

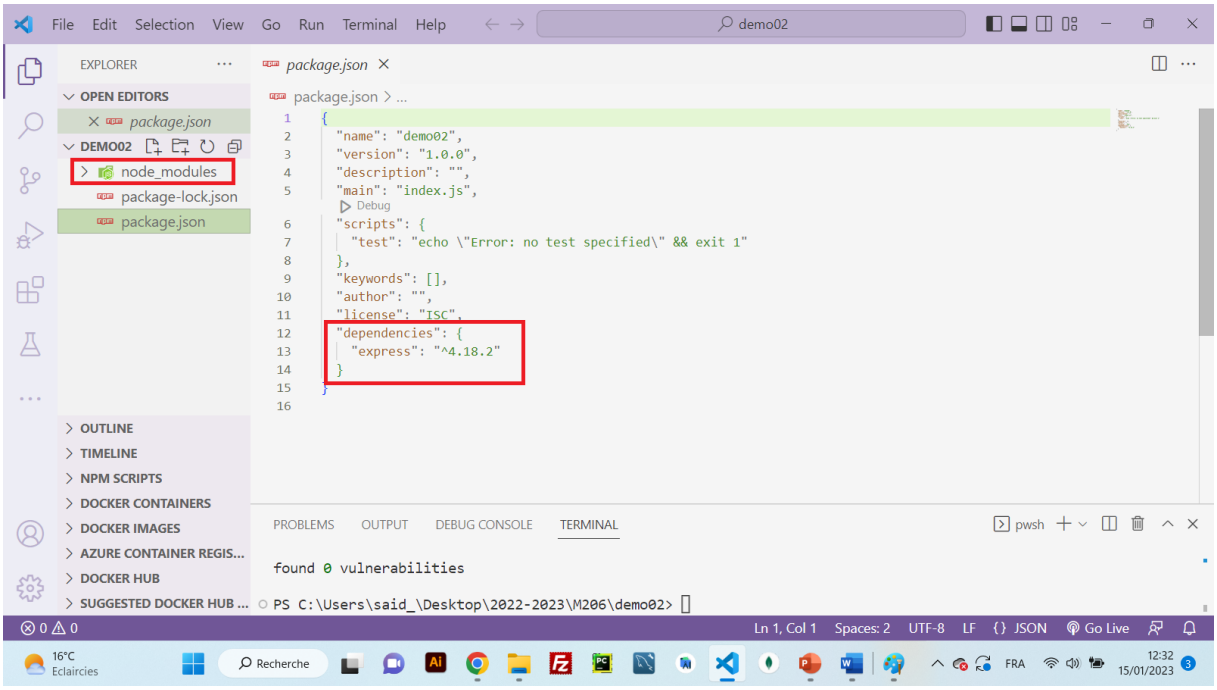
Un fichier **package.json** sera créer à la racine de votre projet



Maintenant installer Framework expressJS

PS C:\Users\said_\Desktop\2022-2023\M206\demo02> **npm install express**





- Ajouter un fichier **index.js** qui permet de créer un serveur http :

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

app.get('/',(req,res) => {
    res.send("Welcome to simplilearn");
});

app.listen(4000, ()=>{
    console.log("listening to port 4000");
});
```

```
const express = require('express');
const app = express();
const port = process.env.PORT || 8000;

app.get('/', (req, res) => {
    res.set('Content-Type', 'text/html');
    res.send('Hello world !!');
});

app.listen(port, () => {
    console.log('Server app listening on port ' + port);
});
```

- La commande **res.set** permet de fixer la valeur d'un champ de l'entête HTTP transmis au client.
- La commande **res.send** permet de transmettre une réponse au client.

Lancez le serveur : `node index.js`. Entrez l'URL suivante dans votre navigateur web :

```
http://localhost:8000/
```

```
{
  "name": "web",
  "version": "1.0.0",
  "description": "",
  "main": "server/server.js",
  "scripts": {
    "start": "node server/server.js",
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "ISC",
  "dependencies": {
    "express": "^4.16.3"
  }
}
```

PS C:\Users\said_\Desktop\2022-2023\M206\demo02> npm start

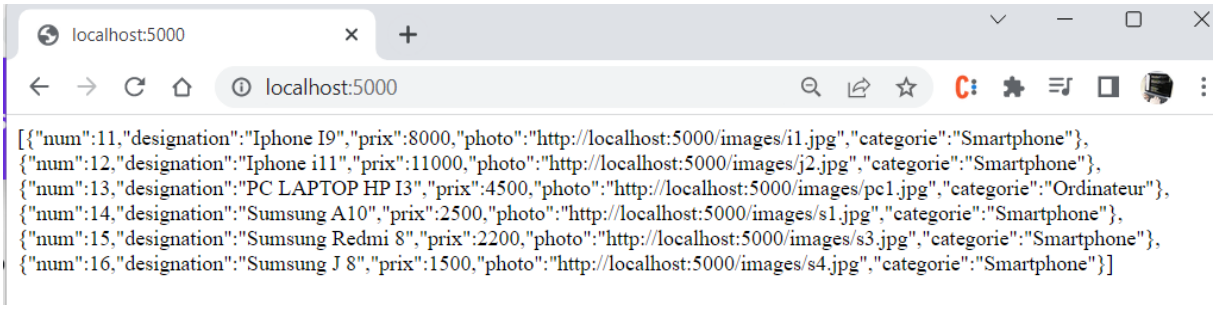
Maintenant modifier le fichier index.js en ajoutant un tableau d'objets produit

```
const express = require('express');
const app = express();
const port = process.env.PORT || 5000;

produits=[
    {
        num:11,
        designation:"Iphone I9",
        prix:8000,
        photo:'http://localhost:5000/images/i1.jpg',
        categorie:'Smartphone'
    },
    {
        num:12,
        designation:"Iphone i11",
        prix:11000,
        photo:'http://localhost:5000/images/j2.jpg',
        categorie:'Smartphone'
    },
    {
        num:13,
        designation:"PC LAPTOP HP I3",
        prix:4500,
        photo:'http://localhost:5000/images/pc1.jpg',
        categorie:'Ordinateur'
    },
    {
        num:14,
        designation:"Sumsung A10",
        prix:2500,
        photo:'http://localhost:5000/images/s1.jpg',
        categorie:'Smartphone'
    },
    {
        num:15,
        designation:"Sumsung Redmi 8",
        prix:2200,
        photo:'http://localhost:5000/images/s3.jpg',
        categorie:'Smartphone'
    },
    {
        num:16,
        designation:"Sumsung J 8",
        prix:1500,
        photo:'http://localhost:5000/images/s4.jpg',
        categorie:'Smartphone'
    }
];

app.get('/', (req, res) => {
    res.set('Content-Type', 'text/html');
    res.send(JSON.stringify(produits));
});
```

```
app.listen(port, () => {
    console.log('Server app listening on port ' + port);
});
```

Lancer à nouveau le serveur :

http://localhost:5000/



Installation des packages :

Le protocole HTTP définit le mode de communication entre le client (votre navigateur) et le serveur (votre application *Express*). Il définit le format des requêtes pouvant être émises par le client. Elles peuvent être décomposées en deux éléments :

- les *méthodse*. Ici notre client n'utilisera que les verbes suivants :
  - **POST** : pour envoyer des données au serveur.
  - **PUT/PATCH** : pour modifier ou remplacer des données stockées sur le serveur.
  - **GET** : pour obtenir des données du serveur.
  - **DELETE** : pour supprimer des données stockées sur le serveur.

Outils important :

Nodemon	BodyParser

nodemon

- Redémarrer automatiquement le serveur lorsqu'un changement a été effectué sur un des fichiers du projet

```
npm install -g nodemon
```

PS C:\Users\said_\Desktop\2022-2023\M206\demo02> nodemon demo02

BodyParser

C'est une bibliothèque vous permettant de directement parser le corps d'une requête. Le résultat sera directement disponible dans l'objet request. BodyParser est middleware

npm install body-parser

```
const bodyParser = require("body-parser")
// Content-type: application/json
app.use(bodyParser.json())
// Content-type: application/x-www-form-urlencoded
app.use(bodyParser.urlencoded({ extended: false }))
```

Exemple 1 :

```
app.post("/products", (req, res) => {
    product = {
    name: req.body.name,
    price: req.body.price
    }
res.json(product)
})
```

Définir les routes :

Une route est définie comme suit:

app.method(path, handler)

- method: permet de définir la méthode HTTP de la requête.
- path: permet de définir le chemin de la ressource demandée.
- handler: représente la fonction qui va gérer la requête lors de sa réception.

Handler

Un handler reçoit toujours deux objets en paramètres. Ces objets sont créés par express est sont spécifiques à chaque requête reçue.

```
app.get('/', (req, res) => {
res.send('hello world');
});
```

Chaîner les Handler

Il est également possible de chaîner les Handlers, pour ce faire il suffit de spécifier le paramètre “next” et d’y faire appel.

```
app.get('/example', (req, res, next) => {
console.log('La réponse sera envoyée par la fonction suivante...');
next();
        }, (req, res) => {
res.send('Hello from B!');
});
```

Ordre de déclaration des routes

L'ordre de déclaration des routes est important. Toujours mettre le **chemin racine en dernier**.

```
app.get('/products', (req, res) => {
res.send('products list');
});
app.get('/', (req, res) => {
res.send('hello world');
});
```

Méthodes de l'objet réponse

```
res.send('hello world');
res.status(404).end();
res.status(404).send('product not found.');
res.json(json_object);
res.redirect(301, 'http://example.com');
```

Paramètres d'une requête http

Il existe plusieurs méthodes pour récupérer les paramètres d'une requête HTTP:

```
// http://localhost:3000/?prenom=gahi&nom=said
app.get('/', (req, res) => {
res.send(req.query.prenom);
});
```

```
// http://localhost:3000/gahi/said
app.get('/:prenom/:nom', (req, res) => {
var prenom = req.params.prenom
res.send('Salut ' + prenom + ' !');
});
```

Body d'une requête http

Body d'une requête HTTP

Pour récupérer le body de la requête entrante, il vous suffit d'utiliser l'attribut **body** de l'objet **req**.

```
<form action="login" method="post">
  <input type="text" id="email" name="email">
  <input type="password" name="password">
  <input type="submit" value="Submit">
</form>
```

```
app.post('/login', function (req, res) {
res.json(req.body);
});
```

Etude de cas : Gestion Park

Définir une ressource et ses routes

Maintenant que le serveur est fonctionnel, il est temps de définir **le cœur de l'API:**

Pour notre exemple, nous prendrons le cas d'une société exploitant des parkings de longue durée et qui prend des réservations de la part de ses clients. Nous aurons besoin des fonctionnalités suivantes :

- Créer un parking
- Lister l'ensemble des parkings
- Récupérer les détails d'un parking en particulier
- Supprimer un parking
- Prendre une réservation d'une place dans un parking
- Lister l'ensemble des réservations
- Afficher les détails d'une réservation en particulier
- Supprimer une réservation

Ces opérations sont plus communément appelées CRUD, pour CREATE, READ, UPDATE, DESTROY. Dans notre exemple, **notre Node JS API dispose de deux ressources: le Parking et la Réservation**.

Création des routes

Le standard d'API REST impose que nos routes soient centrées autour de nos ressources et que la méthode HTTP utilisée reflète l'intention de l'action. Dans notre cas nous aurons besoin des routes suivantes :

- GET /parkings
- GET /parkings/:id
- POST /parkings
- PUT /parkings/:id
- DELETE /parkings/:id

Les réservations étant une sous-ressource de la ressource parking, nous aurons à créer les routes suivantes :

- GET /parkings/:id/reservations

- GET /parking/:id/reservations/:idReservation
- POST /parkings/:id/reservations
- PUT /parking/:id/reservations/:idReservation
- DELETE /parking/:id/reservations/:idReservation

Commençons par définir la route GET /parkings.
Cette route a pour but de récupérer l'ensemble des parkings dans nos données. Allons modifier notre fichier **index.js** :

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

const app = express()
app.get('/parkings', (req,res) => {
     res.send("Liste des parkings")

    })

app.listen(8080, () => {

        console.log("Serveur à l'écoute")

    })
```

la méthode `.get` d'express permet de définir une route GET. Elle prend en premier paramètre une *String* qui a défini la route à écouter et une *callback*, qui est la fonction à exécuter si cette route est appelée. Cette callback prend en paramètre l'objet `req`, qui reprend toutes les données fournies par la requête, et l'objet `res`, fourni par express, qui contient les méthodes pour répondre à la requête qui vient d'arriver.

Maintenant que notre route fonctionne et est capable de recevoir la requête entrante, nous allons pouvoir renvoyer la donnée des parkings au lieu d'avoir simplement une chaîne de caractères:

```
const express = require('express')
const app = express()
const parkings = require('./parkings.json')
app.get('/parkings', (req,res) => {

    res.status(200).json(parkings)

})
```

```
app.listen(8080, () => {
        console.log("Serveur à l'écoute")

})
```

La route GET /parkings/:id est la suivante. Nous avons besoin de récupérer l'id de la route depuis l'URL pour n'afficher que le JSON de ce parking dans la réponse. Cet id se trouve dans les *params*, dans l'objet `req`, envoyé par le navigateur. Reprenons notre fichier *index.js* :

```
const express = require('express')
const app = express()
const parkings = require('./parkings.json')
app.get('/parkings', (req,res) => {

    res.status(200).json(parkings)
})

app.get('/parkings/:id', (req,res) => {

    const id = parseInt(req.params.id)
    const parking = parkings.find(parking => parking.id === id)
    res.status(200).json(parking)
})

app.listen(8080, () => {
     console.log("Serveur à l'écoute")

})
```

Nous récupérons l'*id* demandé par le client dans les *params* de la requête. Comme ma route a défini '/:id', la valeur passée dans le *param* sera sous forme d'objet contenant la clé « *id*« . La valeur de req.params.id contient ce qui est envoyé dans l'URL, sous forme de String. Comme l'id de chaque parking est sous forme de Number, il faut d'abord transformer le *params* de String en Number. Ensuite, il faut rechercher dans les parkings pour trouver celui qui a l'*id* correspondant à celui passé dans l'URL.

Passons à la route POST /parkings pour pouvoir créer un nouveau parking.

Pour créer un nouveau parking via votre Node JS API, il va falloir envoyer au serveur les données relatives à ce nouvel élément, telles que son nom, son type etc. **Dès qu'il s'agit d'envoyer de la donnée, il faut utiliser une requête POST**.

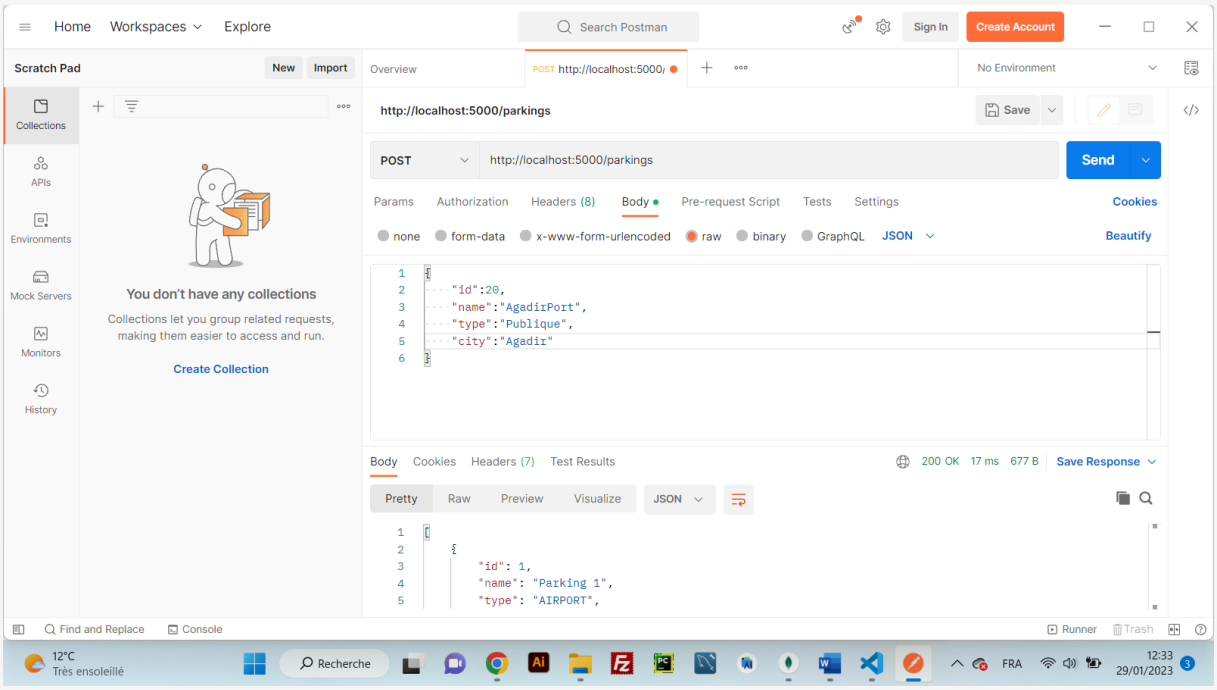
Pour récupérer les données passées dans la requête POST, nous devons ajouter un *middleware* **à notre Node JS API afin qu'elle soit capable d'interpréter le body de la requête.** Ce middleware va se placer à entre l'arrivée de la requête et nos routes et exécuter son code, rendant possible l'accès au body.

Il n'y a plus qu'à ajouter la route POST et à tester notre nouvelle route:

```
const express = require('express')
const app = express()
const parkings = require('./parkings.json')

const bodyParser = require("body-parser")
// Content-type: application/json
app.use(bodyParser.json())
app.use(bodyParser.urlencoded({ extended: false }))

// Middleware
app.use(express.json())
app.get('/parkings', (req,res) => {

     res.status(200).json(parkings)

})
app.get('/parkings/:id', (req,res) => {

    const id = parseInt(req.params.id)
    const parking = parkings.find(parking => parking.id === id)
    res.status(200).json(parking)

})

app.post('/parkings', (req,res) => {

    park = {
        id:req.body.id,
        name: req.body.name,
        type: req.body.type,
        city: req.body.city
        }
```

```
    parkings.push(park)
    res.status(200).json(parkings)
})
```

```
app.listen(8080, () => {

        console.log("Serveur à l'écoute")
})
```



Pour tester notre route POST, nous allons utiliser l'outil Postman qui nous permet de manipuler facilement des API.

Notre requête POST sur l'URL `localhost:8080/parkings` contient dans son body un objet JSON contenant l'id, le nom, le type et la ville de notre nouveau parking.

Dans un cas réel de Node JS API, votre base de données aurait généré l'id. Dans notre cas nous allons le passer à la main pour simplifier.

Passons à **la route PUT /parkings/:id** pour pouvoir modifier un parking.

```
app.put('/parkings/:id', (req,res) => {
    const id = parseInt(req.params.id)
    let parking = parkings.find(parking => parking.id === id)
    parking.name =req.body.name,
    parking.city =req.body.city,
    parking.type =req.body.type,
    res.status(200).json(parking)
```

```
})
```

Il reste maintenant à terminer cette ressource avec la route `**DELETE /parkings**`

```
app.delete('/parkings/:id', (req,res) => {
    const id = parseInt(req.params.id)
    let parking = parkings.find(parking => parking.id === id)
    parkings.splice(parkings.indexOf(parking),1)
    res.status(200).json(parkings)
})
```