



Express



NodeJS – Express JS

FRÉDÉRIC NADARADJANE

11/05/2022

NodeJS qu'est-ce que c'est ?

Node.JS est une plateforme de développement et d'exécution d'applications client serveur en Javascript Créé en 2009, le noyau de Node.JS est basé sur V8 le moteur Javascript de chrome et la librairie libuv (E/S asynchrones).

- Node.JS supporte le modèle de programmation asynchrone/événementielle.
- Node.JS se distingue par la réduction du temps de traitement et la capacité à traiter de nombreuses requêtes simultanément.
- Node.JS implémente le standard commonJS pour supporter le développement d'applications modulaires
- Installation
- Site officiel: <https://nodejs.org/fr>
- NVM (Node Version Manager) : outil de gestion des versions pour node.



NPM (Node Package Manager)

- Node Package Manager (NPM) est un outil en ligne de commande qui installe, met à jour ou désinstalle des paquets Node.js dans votre application. Il s'agit également d'un référentiel en ligne pour les paquets Node.js open-source. La communauté node du monde entier crée des modules utiles et les publie sous forme de paquets dans ce dépôt.
- Site officiel : <https://www.npmjs.com>

NPM (Node Package Manager)

- NPM est inclus dans l'installation de Node.js. Après avoir installé Node.js, vérifiez l'installation de NPM en écrivant la commande suivante dans un terminal ou une invite de commande.

```
C:\> npm -v  
8.11.0
```

- Pour accéder à l'aide de NPM, écrivez `npm help` dans l'invite de commande ou la fenêtre du terminal.

```
C:\> npm help
```

NPM (Node Package Manager)

NPM effectue l'opération en deux modes : global et local. Dans le mode global, NPM effectue des opérations qui affectent toutes les applications Node.js sur l'ordinateur, tandis que dans le mode local, NPM effectue des opérations pour le répertoire local particulier qui affecte une application dans ce répertoire uniquement.

Installer le package localement

Utilisez la commande suivante pour installer tout module tiers dans le dossier local de votre projet Node.js.

```
C:\>npm install <package name>
```

Par exemple, la commande suivante installera ExpressJS dans votre dossier.

```
C:\MyNodeProj> npm install express
```

Tous les modules installés à l'aide de NPM sont installés dans le dossier `node_modules`. La commande ci-dessus créera un dossier ExpressJS sous le dossier `node_modules` dans le dossier racine de votre projet et y installera Express.js.

Ajouter la dépendance dans package.json

Utilisez la commande `--save` à la fin de la commande `install` pour ajouter une entrée de dépendance dans le fichier `package.json` de votre application.

Par exemple, la commande suivante installera ExpressJS dans votre application et ajoutera également une entrée de dépendance dans le fichier `package.json`.

```
C:\MyNodeProj> npm install express --save
```

Ajouter la dépendance dans package.json

Le package.json du projet NodejsConsoleApp ressemblera à quelque chose comme ci-dessous.

package.json

```
{
  "name": "NodejsConsoleApp",
  "version": "0.0.0",
  "description": "NodejsConsoleApp",
  "main": "app.js",
  "author": {
    "name": "Dev",
    "email": ""
  },
  "dependencies": {
    "express": "^4.13.3"
  }
}
```


Installer les paquets globalement

NPM peut également installer des paquets de manière globale afin que toutes les applications node.js sur cet ordinateur puissent importer et utiliser les paquets installés. NPM installe les paquets globaux dans le dossier `/<User>/local/lib/node_modules`. Appliquez `-g` dans la commande `install` pour installer le paquet de manière globale. Par exemple, la commande suivante installera ExpressJS de manière globale.

```
C:\MyNodeProj> npm install -g express
```

Mettre à jour le package

NPM peut également installer des paquets de manière globale afin que toutes les applications node.js sur cet ordinateur puissent importer et utiliser les paquets installés. NPM installe les paquets globaux dans le dossier `/<User>/local/lib/node_modules`. Appliquez `-g` dans la commande `install` pour installer le paquet de manière globale. Par exemple, la commande suivante installera ExpressJS de manière globale.

```
C:\MyNodeProj> npm install -g express
```

Installer les packages globalement

Pour mettre à jour le paquet installé localement dans votre projet Node.js, naviguez dans l'invite de commande ou dans la fenêtre du terminal jusqu'au dossier du projet et écrivez la commande de mise à jour suivante.

```
C:\MyNodeProj> npm update <package name>
```

La commande suivante mettra à jour le module ExpressJS existant à la dernière version.

```
C:\MyNodeProj> npm update express
```

Désinstaller les packages

Utilisez la commande suivante pour supprimer un paquet local de votre projet.

```
C:\>npm uninstall <package name>
```

La commande suivante désinstallera ExpressJS de l'application.

```
C:\MyNodeProj> npm uninstall express
```

Node.js web server

DANS CETTE SECTION, NOUS ALLONS APPRENDRE À CRÉER UN SIMPLE SERVEUR WEB NODE.JS ET À GÉRER LES REQUÊTES HTTP.

Node.js web server

Pour accéder aux pages web d'une application web, vous avez besoin d'un serveur web. Le serveur web traitera toutes les demandes http pour l'application web, par exemple IIS est un serveur web pour les applications web ASP.NET et Apache est un serveur web pour les applications web PHP ou Java, fonctionnant sous Windows Server.


Node.js offre la possibilité de créer votre propre serveur web qui traitera les requêtes HTTP de manière asynchrone. Vous pouvez utiliser IIS ou Apache pour exécuter une application web Node.js mais il est recommandé d'utiliser le serveur web Node.js.

Crée un serveur Node.js

Node.js permet de créer facilement un serveur web simple qui traite les demandes entrantes de manière asynchrone.

L'exemple suivant est un serveur web Node.js simple contenu dans le fichier server.js.

server.js

 Copy

```
var http = require('http'); // 1 - Import Node.js core module

var server = http.createServer(function (req, res) { // 2 - creating server

    //handle incomming requests here..

});

server.listen(5000); //3 - listen for any incoming requests

console.log('Node.js web server at port 5000 is running..')
```

Node.js web server

Dans l'exemple ci-dessus, nous importons le module http en utilisant la fonction `require()`. Le module http est un module de base de Node.js, il n'est donc pas nécessaire de l'installer en utilisant NPM. L'étape suivante consiste à appeler la méthode `createServer()` de http et à spécifier la fonction callback avec les paramètres de demande et de réponse. Enfin, appelez la méthode `listen()` de l'objet serveur qui a été renvoyé par la méthode `createServer()` avec le numéro de port, pour commencer à écouter les demandes entrantes sur le port 5000. Vous pouvez spécifier n'importe quel port non utilisé ici. Exécutez le serveur web ci-dessus en écrivant la commande `node server.js` dans l'invite de commande ou dans la fenêtre du terminal et il affichera le message comme indiqué ci-dessous.

```
C:\> node server.js  
Node.js web server at port 5000 is running..
```


Traiter les requêtes HTTP

C'est ainsi que vous créez un serveur web Node.js en suivant des étapes simples. Maintenant, voyons comment gérer une requête HTTP et envoyer une réponse dans un serveur web Node.js.

La méthode `http.createServer()` comprend des paramètres de demande et de réponse qui sont fournis par Node.js.

L'objet `request` peut être utilisé pour obtenir des informations sur la requête HTTP en cours, par exemple l'url, l'en-tête de la requête et les données.

L'objet `response` peut être utilisé pour envoyer une réponse à une demande HTTP en cours. L'exemple suivant démontre la manipulation de la demande et de la réponse HTTP dans Node.js.

Traiter les requêtes HTTP

server.js

Copy

```
var http = require('http'); // Import Node.js core module

var server = http.createServer(function (req, res) { //create web server
  if (req.url == '/') { //check the URL of the current request

    // set response header
    res.writeHead(200, { 'Content-Type': 'text/html' });

    // set response content
    res.write('<html><body><p>This is home Page.</p></body></html>');
    res.end();

  }
  else if (req.url == "/student") {

    res.writeHead(200, { 'Content-Type': 'text/html' });
    res.write('<html><body><p>This is student Page.</p></body></html>');
    res.end();

  }
  else if (req.url == "/admin") {

    res.writeHead(200, { 'Content-Type': 'text/html' });
    res.write('<html><body><p>This is admin Page.</p></body></html>');
    res.end();

  }
  else
    res.end('Invalid Request!');

});

server.listen(5000); //6 - listen for any incoming requests

console.log('Node.js web server at port 5000 is running..')
```

Traiter les requêtes HTTP

Maintenant, exécuter le serveur web ci-dessus comme indiqué ci-dessous.

```
C:\> node server.js  
Node.js web server at port 5000 is running..
```

Pour le tester, vous pouvez utiliser le programme de ligne de commande curl, qui est préinstallé sur la plupart des ordinateurs Mac et Linux.

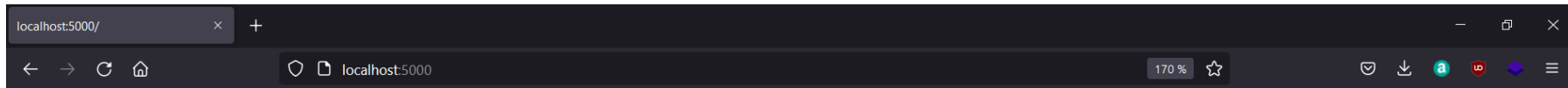
```
curl -i http://localhost:5000
```

Vous devriez voir la réponse suivante

```
HTTP/1.1 200 OK  
Content-Type: text/plain  
Date: Tue, 8 Sep 2015 03:05:08 GMT  
Connection: keep-alive  
This is home page.
```

Traiter les requêtes HTTP

Pour les utilisateurs de Windows, pointez votre navigateur sur <http://localhost:5000> et voyez le résultat suivant.

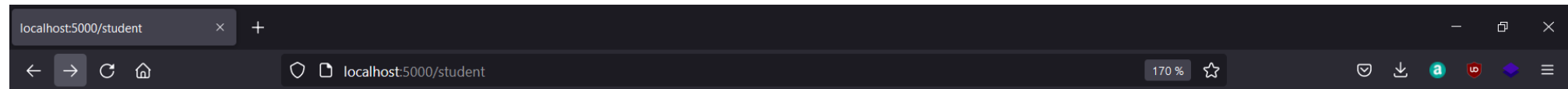


This is home Page.

De la même manière, pointez votre navigateur sur <http://localhost:5000/student> et voyez le résultat suivant.

Traiter les requêtes HTTP

De la même manière, pointez votre navigateur sur <http://localhost:5000/student> et voyez le résultat suivant.

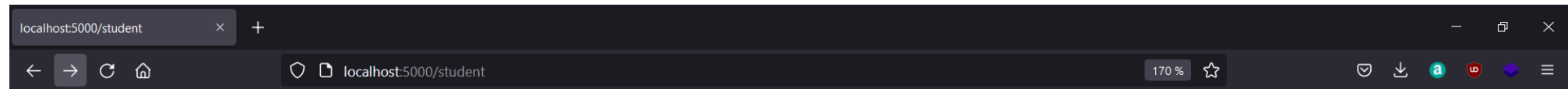


This is student Page.

Il affichera "Invalid Request" pour toutes les requêtes autres que les URLs ci-dessus.

Traiter les requêtes HTTP

De la même manière, pointez votre navigateur sur <http://localhost:5000/student> et voyez le résultat suivant.



This is student Page.

Il affichera "Invalid Request" pour toutes les requêtes autres que les URLs ci-dessus.

Sending JSON Response

L'exemple suivant montre comment servir une réponse JSON à partir du serveur web Node.js.

server.js

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

var server = http.createServer(function (req, res) {

    if (req.url == '/data') { //check the URL of the current request
        res.writeHead(200, { 'Content-Type': 'application/json' });
        res.write(JSON.stringify({ message: "Hello World" }));
        res.end();
    }
});

server.listen(5000);

console.log('Node.js web server at port 5000 is running..')
```

Node.js file system

Node.js inclut le module **fs** pour accéder au système de fichiers physiques. Le module fs est responsable de toutes les opérations d'E/S de fichiers asynchrones ou synchrones.

Voyons quelques exemples d'opérations d'E/S courantes utilisant le module fs.

Reading File

Use `fs.readFile()` method to read the physical file asynchronously.

Reading file

Utilisez la méthode `fs.readFile()` pour lire le fichier physique de manière asynchrone.

Signature:

```
fs.readFile(fileName [,options], callback)
```

Paramètre Description :


`filename` : Chemin complet et nom du fichier sous forme de chaîne.

`options` : Le paramètre `options` peut être un objet ou une chaîne qui peut inclure l'encodage et l'indicateur. L'encodage par défaut est `utf8` et l'indicateur par défaut est `"r"`.

`callback` : Une fonction avec deux paramètres `err` et `fd`. Elle sera appelée lorsque l'opération `readFile` sera terminée.

Reading file

L'exemple suivant démontre la lecture asynchrone du fichier TestFile.txt existant.

Example: Reading File	 Copy
<pre>var fs = require('fs'); fs.readFile('TestFile.txt', function (err, data) { if (err) throw err; console.log(data); });</pre>	

L'exemple ci-dessus lit TestFile.txt (sous Windows) de manière asynchrone et exécute la fonction de rappel lorsque l'opération de lecture est terminée. Cette opération de lecture déclenche une erreur ou se termine avec succès. Le paramètre err contient des informations sur l'erreur, le cas échéant. Le paramètre data contient le contenu du fichier spécifié

Reading file

Voici un exemple de fichier TextFile.txt

TextFile.txt

This is test file to test fs module of Node.js

Maintenant, exécutez l'exemple ci-dessus et voyez le résultat comme indiqué ci-dessous.

```
C:\> node server.js
```

```
This is test file to test fs module of Node.js
```

Node.js web server

Utilisez la méthode `fs.readFileSync()` pour lire le fichier de manière synchrone, comme indiqué ci-dessous.

Example: Reading File Synchronously

[Copy](#)

```
var fs = require('fs');  
  
var data = fs.readFileSync('dummyfile.txt', 'utf8');  
console.log(data);
```

Writing File

Utilisez la méthode `fs.writeFile()` pour écrire des données dans un fichier. Si le fichier existe déjà, il écrase le contenu existant, sinon il crée un nouveau fichier et y écrit les données.

Signature:

```
fs.writeFile(filename, data[, options], callback)
```

Paramètre Description :

- `filename` : chemin complet et nom du fichier sous forme de chaîne.
- `data` : Le contenu à écrire dans un fichier.
- `options` : Le paramètre `options` peut être un objet ou une chaîne de caractères qui peut inclure l'encodage, le mode et le drapeau. L'encodage par défaut est `utf8` et l'indicateur par défaut est `"r"`.
- `callback` : Une fonction avec deux paramètres `err` et `fd`. Elle sera appelée lorsque l'opération d'écriture sera terminée.

Writing File

L'exemple suivant crée un nouveau fichier appelé test.txt et y écrit "Hello World" de manière asynchrone.

Example: Creating & Writing File

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

fs.writeFile('test.txt', 'Hello World!', function (err) {
    if (err)
        console.log(err);
    else
        console.log('Write operation complete.');
```

```
});
```

Writing File

De la même manière, utilisez la méthode `fs.appendFile()` pour ajouter le contenu à un fichier existant.

Example: Append File Content

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

fs.appendFile('test.txt', 'Hello World!', function (err) {
    if (err)
        console.log(err);
    else
        console.log('Append operation complete.');
```

```
});
```

Framework pour Node

IL EXISTES DE NOMBREUX FRAMEWORKS POUR NODE.JS

Révision sur NodeJS

Framework pour Node.js

Vous avez appris que nous devons écrire nous-mêmes beaucoup de code de bas niveau pour créer une application Web à l'aide de Node.js dans la section consacrée au serveur Web Node.js.

Il existe divers frameworks tiers open-source disponibles dans le gestionnaire de paquets Node qui rendent le développement d'applications Node.js plus rapide et plus facile. Vous pouvez choisir un framework approprié en fonction des exigences de votre application.

Framework pour Node.js

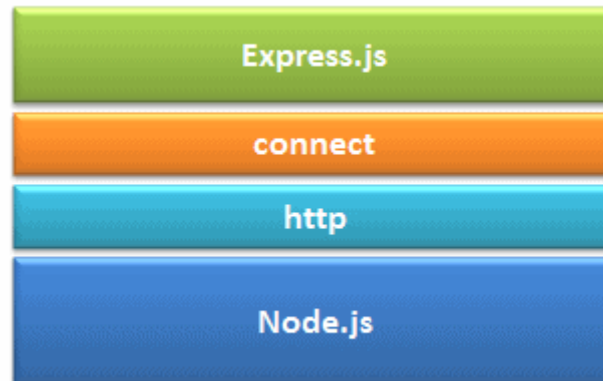
- [Express.js](#),
- [Geddy](#),
- [Locomotive](#),
- [Koa](#),
- [Total.js](#),
- [Hapi.js](#),
- [Keystone](#),
- [Derbyjs](#),
- [Sails.js](#),
- [Meteor](#),
- [Mojito](#),
- [Restify](#),
- [Loopback](#),
- [ActionHero](#),
- [Frisby](#),
- [Chocolate.js](#)

Framework pour Node.js

Parmi tout ces frameworks nous allons étudier ExpressJs qui est le framework le plus répandu.

Express.js est un framework web pour Node.js. Il fournit diverses fonctionnalités qui rendent le développement d'applications web rapide et facile, ce qui prendrait autrement plus de temps en utilisant uniquement Node.js.

Express.js est basé sur le module middleware de Node.js appelé connect qui, à son tour, utilise le module http. Ainsi, tout intergiciel basé sur connect fonctionnera également avec Express.js.



Express.JS

Avantages d'Express.js

1. Rend le développement d'applications web Node.js rapide et facile.
2. Facile à configurer et à personnaliser.
3. Vous permet de définir les routes de votre application en fonction des méthodes HTTP et des URL.
4. Comprend divers modules middleware que vous pouvez utiliser pour effectuer des tâches supplémentaires sur la demande et la réponse.
5. Facile à intégrer avec différents moteurs de modèles comme Jade, Vash, EJS, etc.
6. Permet de définir un intergiciel de gestion des erreurs.
7. Permet de servir facilement les fichiers statiques et les ressources de votre application.
8. Permet de créer un serveur API REST.
9. Facilité de connexion avec des bases de données telles que MongoDB, Redis, MySQL.

Express.JS

Vous pouvez installer express.js en utilisant npm. La commande suivante installera la dernière version d'express.js globalement sur votre machine afin que chaque application Node.js sur votre machine puisse l'utiliser.

```
npm install -g express
```

La commande suivante installera la dernière version d'express.js dans le dossier de votre projet.

```
C:\MyNodeJSApp> npm install express --save
```

Comme vous le savez, --save mettra à jour le fichier package.json en spécifiant la dépendance d'express.js.

Express.js Web Application

Dans cette section, vous apprendrez à créer une application Web à l'aide d'Express.js.

Express.js offre un moyen simple de créer un serveur Web et de rendre des pages HTML pour différentes requêtes HTTP en configurant des routes pour votre application.

Express.js Web Application

Tout d'abord, importez le module Express.js et créez le serveur web comme indiqué ci-dessous.

app.js: Express.js Web Server

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

// define routes here..

var server = app.listen(5000, function () {
  console.log('Node server is running..');
});
```


Express.js Web Application

Dans l'exemple ci-dessus, nous avons importé le module Express.js en utilisant la fonction `require()`.

Le module `express` renvoie une fonction.

Cette fonction renvoie un objet qui peut être utilisé pour configurer l'application Express (`app` dans l'exemple ci-dessus).

L'objet `app` comprend des méthodes pour acheminer les demandes HTTP, configurer le middleware, rendre les vues HTML et enregistrer un moteur de modèle.

La fonction `app.listen()` crée le serveur web Node.js à l'hôte et au port spécifiés.

Elle est identique à la méthode `http.Server.listen()` de Node. Exécutez l'exemple ci-dessus en utilisant la commande `node app.js` et faites pointer votre navigateur sur `http://localhost:5000`. Il affichera `Cannot GET /` car nous n'avons pas encore configuré de routes.

Configure Routes

Utilisez l'objet `app` pour définir les différentes routes de votre application. L'objet `app` comprend les méthodes `get()`, `post()`, `put()` et `delete()` pour définir les routes pour les requêtes HTTP GET, POST, PUT et DELETE respectivement.

L'exemple suivant montre la configuration des routes pour les demandes HTTP.

Configure Routes

Utilisez l'objet `app` pour définir les différentes routes de votre application. L'objet `app` comprend les méthodes `get()`, `post()`, `put()` et `delete()` pour définir les routes pour les requêtes HTTP GET, POST, PUT et DELETE respectivement.

L'exemple suivant montre la configuration des routes pour les demandes HTTP.

Configure Routes

Example: Configure Routes in Express.js

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

app.get('/', function (req, res) {
  res.send('<html><body><h1>Hello World</h1></body></html>');
});

app.post('/submit-data', function (req, res) {
  res.send('POST Request');
});

app.put('/update-data', function (req, res) {
  res.send('PUT Request');
});

app.delete('/delete-data', function (req, res) {
  res.send('DELETE Request');
});

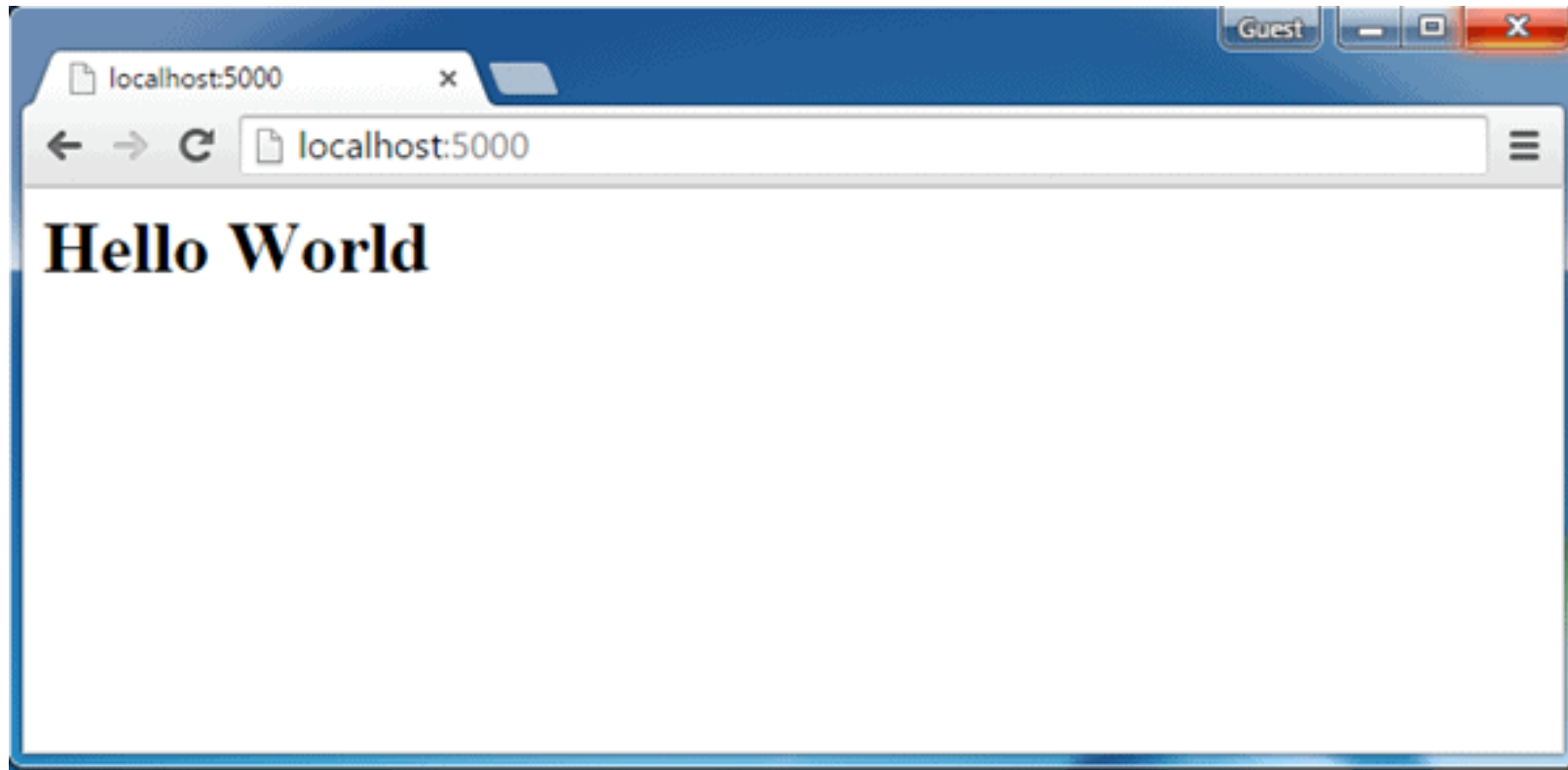
var server = app.listen(5000, function () {
  console.log('Node server is running..');
});
```

Configure Routes

Dans l'exemple ci-dessus, les méthodes `app.get()`, `app.post()`, `app.put()` et `app.delete()` définissent des routes pour HTTP GET, POST, PUT, DELETE respectivement. Le premier paramètre est le chemin d'une route qui commencera après l'URL de base. La fonction de rappel comprend la demande et l'objet de réponse qui sera exécuté sur chaque demande.

Exécutez l'exemple ci-dessus en utilisant la commande `node server.js`, et faites pointer votre navigateur sur `http://localhost:5000` et vous verrez le résultat suivant.

Configure Routes



Handle POST Request

Ici, vous apprendrez à gérer les requêtes HTTP POST et à récupérer les données du formulaire soumis.

Tout d'abord, créez le fichier `Index.html` dans le dossier racine de votre application et écrivez-y le code HTML suivant.

Handle POST Request

Example: Configure Routes in Express.js

```
<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <meta charset="utf-8" />
  <title></title>
</head>
<body>
  <form action="/submit-student-data" method="post">
    First Name: <input name="firstName" type="text" /> <br />
    Last Name: <input name="lastName" type="text" /> <br />
    <input type="submit" />
  </form>
</body>
</html>
```


Handle POST Request

Pour traiter les requêtes HTTP POST dans Express.js version 4 et supérieure, vous devez installer le module middleware appelé body-parser. L'intergiciel faisait auparavant partie d'Express.js, mais vous devez désormais l'installer séparément.

Ce module body-parser analyse les données codées JSON, buffer, string et url soumises à l'aide d'une requête HTTP POST. Installez body-parser en utilisant NPM comme indiqué ci-dessous.

```
npm install body-parser --save
```

Maintenant, importez body-parser et récupérez les données de la requête POST comme indiqué ci-dessous.

Handle POST Request

app.js: Handle POST Route in Express.js

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

var bodyParser = require("body-parser");
app.use(bodyParser.urlencoded({ extended: false }));

app.get('/', function (req, res) {
  res.sendFile('index.html');
});

app.post('/submit-student-data', function (req, res) {
  var name = req.body.firstName + ' ' + req.body.lastName;

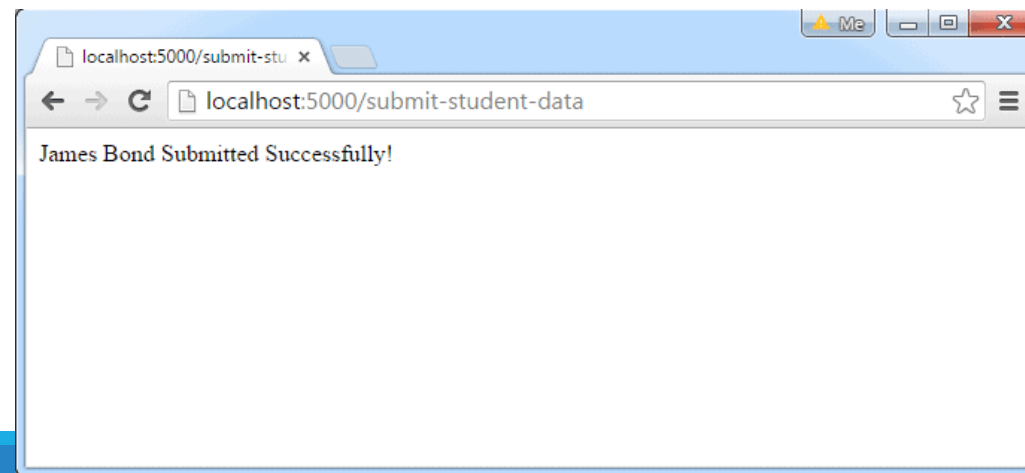
  res.send(name + ' Submitted Successfully!');
});

var server = app.listen(5000, function () {
  console.log('Node server is running..');
});
```

Handle POST Request

Dans l'exemple ci-dessus, les données POST sont accessibles à l'aide de `req.body`. Le `req.body` est un objet qui comprend des propriétés pour chaque formulaire soumis. `Index.html` contient les types de saisie `firstName` et `lastName`, vous pouvez donc y accéder en utilisant `req.body.firstName` et `req.body.lastName`.

Maintenant, exécutez l'exemple ci-dessus en utilisant la commande `node server.js`, faites pointer votre navigateur sur `http://localhost:5000` et voyez le résultat suivant.



Handle POST Request

Exercice

Créer un formulaire de contact :

Nom, prénom, email et message

La page suivante devra afficher le message suivant :

Bonjour [Nom] [prénom],

Merci de nous avoir contacter.

Nous reviendrons vers vous dans les plus bref délai à cette adresse : [email]



Mongodb

DÉCOUVREZ COMMENT ACCÉDER À LA BASE DE DONNÉES
DOCUMENTAIRE MONGODB À L'AIDE DE NODE.JS DANS CETTE
SECTION.

Installation de mongodb

Afin d'accéder à la base de données MongoDB, nous devons installer les pilotes MongoDB. Pour installer les pilotes MongoDB natifs à l'aide de NPM, ouvrez une invite de commande et écrivez la commande suivante pour installer le pilote MongoDB dans votre application.

```
npm install mongodb --save
```

Il faut ensuite créer une base de données mongo.

Deux solutions s'offrent à vous, l'installation d'une instance de mongodb sur votre machine, ou d'utiliser mongodb Atlas qui met à disposition des bases gratuitement.

<https://account.mongodb.com/account/login>

Installation de mongodb

```
const url =  
"mongodb+srv://frednad:123test@cluster0.4vjgd.mongodb.net/Formu?retryWrites=true&w=ma  
jority"
```

```
mongoose.connect(url, {  
  useNewUrlParser:true,  
  useUnifiedTopology: true  
}).then(console.log("MongoDB connected"))  
.catch(err => console.log(err))
```


Installation de mongodb

Créons nos models.

Les Models vont nous permettre de stocker les données de notre utilisateurs selon des « règles » qu'on appellera ici SCHEMA.

Les types avec mongoDB :

- String,
- Number,
- Boolean,
- Date,
- Map

Installation de mongodb

```
const mongoose = require('mongoose');  
const contactSchema = mongoose.Schema({  
  nom : { type: String, required: true},  
  prenom : { type: String},  
  email : { type: String, required: true},  
  age : { type: Number}  
});  
  
module.exports = mongoose.model('Contact', contactSchema);
```

Installation de mongodb

Après chaque donnée nous pouvons rajouter des « properties »

Pour que le champs soit requis : `required: true` ,

Pour que le champs est une valeur par default dans la base : `default : [value]`

Pour que le champs soit unique : `unique: true`

Plus d'infos sur les properties :

<https://mongoosejs.com/docs/schematypes.html>

Installation de mongodb

```
app.post("/submit-data-form", function(req, res){  
  const Data = new Form({  
    lastname: req.body.lastname,  
    firstname: req.body.firstname,  
    email: req.body.email,  
    message: req.body.message  
  });  
  Data.save().then(()=>{  
    res.redirect('/')  
  }).catch(err=>console.log(err))  
});
```

Installation de mongodb

```
app.get("/", function (req, res){  
  
  Form.find().then(data=>{  
    console.log(data);  
  }).catch(err=> console.log(err))  
  
})
```