

# CRÉER UNE APPLICATION WEB EN NODE JS

---

**P/2** Contexte

**P/4** 1. Installation de NPM (Node Package Manager) et Node JS

**P/5** 2. Créer un gabarit d'application grâce à Express Generator

**P/8** 3. Création de la base de données MongoDB

**P/10** 4. Le codage des interactions avec la base de données

**P/20** 5. Le code source de l'exemple

**P/21** 6. Pour aller plus loin

**P/22** Crédits

---

## Contexte



### Objectif de l'application

L'application que nous allons créer a pour objectif de gérer des utilisateurs, en ajouter ou en supprimer.

Voici le résultat attendu dans un navigateur :

### Express

Welcome to our test

#### User Info

**Name:** John DOE  
**Age:** 35  
**Gender:** Male  
**Location:** US

#### User List

UserName	Email	Delete?
<a href="#">jdoe</a>	jdoe@none.com	<a href="#">delete</a>

#### Add User

Username	Email
Full Name	Age
Location	gender
<input type="button" value="Add User"/>	

## ● Un rappel sur Node JS

Node.js est une plateforme logicielle libre et événementielle en JavaScript. Elle permet de réaliser une application complète en Javascript que ce soit du côté du client ou du serveur.

Un bon nombre de plateformes web connues à fort trafic tournent sur Node JS, on peut citer LinkedIn, Yahoo !, Rakuten,...

De nombreux frameworks Javascript tournent sur Node JS, on peut citer notamment **Express** et **Meteor**

Avec Node JS, on peut construire des applications mobiles cross-platform (avec **ionic**) ou des applications desktop cross-platform (avec **electron**)

Les meilleurs exemples dans ce domaine sont **Discord** et **Slack**, des outils de collaboration intégrant la discussion instantanée notamment utilisés par les gamers et les développeurs.



### Au préalable

Pour acquérir cette compétence, vous avez besoin de connaître le HTML et CSS et d'être à l'aise avec la programmation en Javascript.



### De l'aide

La documentation est essentielle pour bien comprendre le fonctionnement et le développement. Comme tout bon développeur, ayez le réflexe d'avoir la documentation de Node JS ou encore le site des communautés à disposition.

[Site officiel](#)

[Une documentation de développeur](#)

# 1. Installation de NPM (Node Package Manager) et Node JS

---

## ● NPM (Node Package Manager)

**NPM** est un outil en ligne de commande permettant de gérer les dépendances en Javascript (Node JS ici).

Il est à télécharger et à installer. L'installateur Node JS installe également NPM : <https://nodejs.org/en/download/>

**NPM** permet de :

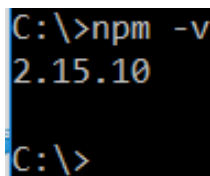
- Garder à jour les différentes bibliothèques dont un projet a besoin pour fonctionner.
- Gérer les dépendances entre bibliothèques (installation & mises à jour).
- Générer un fichier de chargement automatique commun à toutes les bibliothèques générées.

Centralisation des informations de chaque bibliothèque :

- Sur le site [www.npmjs.com](http://www.npmjs.com)
- Accès à Internet requis (configurer pour le proxy).

Pour vérifier si NPM est bien installé, on ouvre un terminal et on exécute la ligne de commande :

**npm -v**



```
C:\>npm -v
2.15.10
C:\>
```

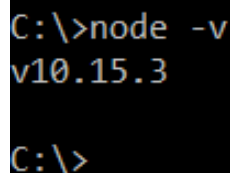
## ● Node JS

Pour installer la plateforme Node JS,

Elle est à télécharger et à installer. L'installateur Node JS installe également NPM : <https://nodejs.org/en/download/>

Pour vérifier si Node JS est bien installé, on ouvre un terminal et on exécute la ligne de commande :

**node -v**



```
C:\>node -v
v10.15.3
C:\>
```

## 2. Créer un gabarit d'application grâce à Express Generator

Nous allons demander à Express Generator (inclus dans Express) de nous créer tous les fichiers initiaux d'une application standard.

Nous allons spécifier que nous allons utiliser le moteur de templates **EJS**.

Ligne de commande : **npx express-generator --view="ejs" nom\_du\_projet**

Cette ligne de commande va créer le répertoire du projet et créer à l'intérieur l'arborescence et les fichiers de lancement, de routes, de vues par défaut.

```
C:\Users\1603344\Documents\nodejs>npx express-generator --view="ejs" nodetest1
npx: installed 10 in 2.722s

  create : nodetest1\
  create : nodetest1\public\
  create : nodetest1\public\javascripts\
  create : nodetest1\public\images\
  create : nodetest1\public\stylesheets\
  create : nodetest1\public\stylesheets\style.css
  create : nodetest1\routes\
  create : nodetest1\routes\index.js
  create : nodetest1\routes\users.js
  create : nodetest1\views\
  create : nodetest1\views\error.ejs
  create : nodetest1\views\index.ejs
  create : nodetest1\app.js
  create : nodetest1\package.json
  create : nodetest1\bin\
  create : nodetest1\bin\www

  change directory:
    > cd nodetest1

  install dependencies:
    > npm install

  run the app:
    > SET DEBUG=nodetest1:* & npm start

C:\Users\1603344\Documents\nodejs>
```

Quand notre projet est créé, il faut installer tous les modules nécessaires au lancement.

Ligne de commande : **npm install**

```
C:\Users\1603344\Documents\nodejs>cd nodetest1

C:\Users\1603344\Documents\nodejs\nodetest1>npm install
ejs@2.6.2 node_modules\ejs

cookie-parser@1.4.4 node_modules\cookie-parser
├── cookie-signature@1.0.6
└── cookie@0.3.1

debug@2.6.9 node_modules\debug
└── ms@2.0.0

http-errors@1.6.3 node_modules\http-errors
├── setprototypeof@1.1.0
├── inherits@2.0.3
├── statuses@1.5.0
└── depd@1.1.2

morgan@1.9.1 node_modules\morgan
├── on-headers@1.0.2
├── depd@1.1.2
├── basic-auth@2.0.1 (safe-buffer@5.1.2)
└── on-finished@2.3.0 (ee-first@1.1.1)

express@4.16.4 node_modules\express
├── escape-html@1.0.3
├── array-flatten@1.1.1
├── setprototypeof@1.1.0
├── cookie-signature@1.0.6
├── utils-merge@1.0.1
├── encodeurl@1.0.2
├── merge-descriptors@1.0.1
├── methods@1.1.2
├── vary@1.1.2
├── content-type@1.0.4
├── parseurl@1.3.3
├── cookie@0.3.1
├── fresh@0.5.2
├── path-to-regexp@0.1.7
├── range-parser@1.2.1
├── content-disposition@0.5.2
├── safe-buffer@5.1.2
├── etag@1.8.1
├── serve-static@1.13.2
├── statuses@1.4.0
├── depd@1.1.2
├── on-finished@2.3.0 (ee-first@1.1.1)
└── qs@6.5.2
```

Enfin, on va installer les modules nous permettant d'interagir avec une base de données MongoDB

On va exécuter notre application pour voir le rendu initial.

Ligne de commande : **npm start**

```
C:\Users\1603344\Documents\nodejs\nodetest1>npm start

> nodetest1@0.0.0 start C:\Users\1603344\Documents\nodejs\nodetest1
> node ./bin/www

GET / 200 12.738 ms - 207
GET /stylesheets/style.css 200 4.226 ms - 111
```

On va voir le résultat dans un navigateur. Le port de l'application standard est 3000.

A screenshot of a web browser's address bar. It features navigation icons (back, forward, refresh) and a lock icon. The address 'localhost:3000' is entered and highlighted in yellow.

# Express

Welcome to Express

### 3. Création de la base de données MongoDB

Avec Compass, nous allons créer la base de données qui contiendra nos utilisateurs.

### Create Database

**Database Name**

**Collection Name**

☐ Capped Collection ⓘ  
☐ Use Custom Collation ⓘ

Before MongoDB can save your new database, a collection name must also be specified at the time of creation. [More Information](#)

CANCEL

CREATE DATABASE

Ensuite nous allons créer manuellement un utilisateur.



The screenshot shows the MongoDB Compass web interface. On the left is a sidebar with a tree view of the database structure. The main area on the right displays the 'userlist' collection in the 'nodetest1' database. The document shown is:

```
{
  "_id": ObjectId("5d305d6fef0d414124e9add2"),
  "username": "jdoe",
  "email": "jdoe@none.com",
  "fullname": "John DOE",
  "age": "35",
  "location": "US",
  "gender": "Male"
}
```

## 4. Le codage des interactions avec la base de données

---

### ● La connection à la base de données

On va mettre en place la connection à la base de données au lancement de l'application.

On va écrire dans le fichier **app.js** se situant à la racine du projet.

Dans ce fichier, on va mettre après les différents **require**, les 3 lignes pour intégrer les bibliothèques et spécifier sur quelle base de données on souhaite se connecter

```
var mongo = require('mongodb');
var monk = require('monk');
var db = monk('localhost:27017/nodetest1');

1 var createError = require('http-errors');
2 var express = require('express');
3 var path = require('path');
4 var cookieParser = require('cookie-parser');
5 var logger = require('morgan');
6
7 // Database
8 var mongo = require('mongodb');
9 var monk = require('monk');
10 var db = monk('localhost:27017/nodetest1');
11
12 var indexRouter = require('./routes/index');
13 var usersRouter = require('./routes/users');
14
15 var app = express();
16
```

### ● Rendre accessible la connexion à la base de données par le Router

On va mettre en place le fait que le Router puisse avoir accès à la connection à la base de données.

On va écrire dans le fichier **app.js** se situant à la racine du projet.

// Make our db accessible to our router

```
app.use(function(req,res,next){
  req.db = db;
  next();
});
```

```

21 app.use(logger('dev'));
22 app.use(express.json());
23 app.use(express.urlencoded({ extended: false }));
24 app.use(cookieParser());
25 app.use(express.static(path.join(__dirname, 'public')));
26
27 // Make our db accessible to our router
28 app.use(function(req, res, next) {
29     req.db = db;
30     next();
31 });
32

```

Cela permettra qu'à chaque action effectuée sur une route on puisse récupérer la connexion à la base de données et agir dessus.

Par exemple, sur la route "AddUser", on va pouvoir ajouter un utilisateur en base de données.

Toutes les routes qui vont être mises pour les actions à réaliser sont à inscrire avant **module.exports = router** ;

Cela va inscrire les actions et les routes dans le système.

```

var express = require('express');
var router = express.Router();

/* Routes et actions à mettre ici */

module.exports = router;

```

## ● Afficher la liste des utilisateurs enregistrés en base de données

On va aller écrire notre route et notre première action dans **users.js** (routes).

```

/* GET Userlist page. */
router.get('/userlist', function(req, res) {
  var db = req.db;
  var collection = db.get('userlist');
  collection.find({}, {}, function(e, docs) {
    res.render('userlist', {
      "userlist": docs
    });
  });
});

```

```
1 var express = require('express');
2 var router = express.Router();
3
4 /* GET Userlist page. */
5 router.get('/userlist', function(req, res) {
6     var db = req.db;
7     var collection = db.get('userlist');
8     collection.find({}, {}, function(e, docs) {
9         res.render('userlist', {
10             "userlist" : docs
11         });
12     });
13 });
14
15 module.exports = router;
16
```

Pour la route "userlist" on va chercher tous les documents de la collection "users" en base de données et on renvoie les documents en paramètre à un template "userlist"


On va maintenant aller créer le template nécessaire à l'affichage de notre liste d'utilisateurs

Le fichier est créé dans le répertoire **views** et porte le nom **userlist.ejs**

On va l'éditer avec un éditeur de texte et mettre le code

```
<head>
  <title>User List</title>
</head>
<body>
  <h1>User List</h1>
  <ul>
    <%
      var list = '';
      for (i = 0; i < userlist.length; i++) {
        list += '<li><a href="mailto:' + userlist[i].email + '">
      }
    %>
    <%- list %>
  </ul>
</body>
</html>
```

Voici le résultat dans un navigateur (adresse : localhost:3000/users/userlist)

 localhost:3000/users/userlist/

# User List

- [jdoe](#)

Nous n'avons qu'un utilisateur dans la base pour le moment et nous ne sommes pas occupés de la mise en forme (CSS)

## ● Ajout d'un utilisateur

Pour ajouter un utilisateur, nous allons créer une nouvelle route, un formulaire html pour compléter les informations de ce nouvel utilisateur.

Tout d'abord la nouvelle route.

```
/* POST to Add User Service */
router.post('/adduser', function(req, res) {

  // Set our internal DB variable
  var db = req.db;

  // Get our form values. These rely on the "name" attributes
  var userName = req.body.username;
  var userEmail = req.body.useremail;
  var userFullName = req.body.userfullname;
  var userAge = req.body.userage;
  var userLocation = req.body.userlocation;
  var userGender = req.body.usergender;

  // Set our collection
  var collection = db.get('userlist');

  // Submit to the DB
  collection.insert({
    "username" : userName,
    "email" : userEmail,
    "fullname" : userFullName,
    "age" : userAge,
    "location" : userLocation,
    "gender" : userGender
  }, function (err, doc) {
    if (err) {
      // If it failed, return error
      res.send("There was a problem adding the information to the database.");
    }
    else {
      // And forward to success page
      res.redirect("userlist");
    }
  });
});
```

On récupère les informations de la requête envoyés d'un formulaire en post et on insère un nouveau document dans la base de données. Si réussite, on redirige vers la liste des utilisateurs sinon on affiche un message d'erreur.

Ensuite on va modifier la vue de la liste d'utilisateurs en intégrant un formulaire d'ajout utilisateur (**userlist.ejs**)

```

1 <!DOCTYPE html>
2 <html>
3   <head>
4     <title>User List</title>
5   </head>
6   <body>
7     <h1>User List</h1>
8     <ul>
9       <%
10         var list = '';
11         for (i = 0; i < userlist.length; i++) {
12           list += '<li><a href="mailto:' + userlist[i].email + '>' + userlist[i].username + '</a></li>';
13         }
14       <%- list %>
15     </ul>
16     <form id="formAddUser" name="adduser" method="post" action="/users/adduser">
17       <input id="inputUserName" type="text" placeholder="username" name="username" />
18       <input id="inputUserEmail" type="text" placeholder="email" name="useremail" />
19       <input id="inputUserFullName" type="text" placeholder="fullname" name="userfullname" />
20       <input id="inputUserAge" type="text" placeholder="age" name="userage" />
21       <input id="inputUserLocation" type="text" placeholder="location" name="userlocation" />
22       <input id="inputUserGender" type="text" placeholder="gender" name="usergender" />
23       <button id="btnSubmit" type="submit">Submit</button>
24     </form>
25   </body>
26 </html>

```

On va tester maintenant dans un navigateur.

← → ↻ localhost:3000/users/userlist

## User List

- [jdoe](#)

toto	toto@nono.fr	Toto	25	France
Unknown	Submit			

On remplit les informations du nouvel utilisateur, on soumet.

Il s'affiche désormais dans notre liste.

← → ↻ localhost:3000/users/userlist

## User List

- [jdoe](#)
- [toto](#)

username	email	fullname	age	location
gender	Submit			

On ajoute dans le design un bouton pour ajouter un nouvel utilisateur. Quand on clique sur ce bouton, on vide toutes les informations du formulaire et on met le curseur sur la zone "username". Nous allons nous aider de jquery pour cela en l'intégrant dans notre **userlist.ejs**.

```
<script
src="https://code.jquery.com/jquery-3.4.1.min.js"
integrity="sha256-CSXorXvZcTkaix6Yvo6HppcZGetbYMGWSFlBw8HfCJo="
crossorigin="anonymous"></script>
<script type="text/javascript">
```

```
function addUser() {
    $('#inputUserName').val('');
    $('#inputUserEmail').val('');
    $('#inputUserFullName').val('');
    $('#inputUserAge').val('');
    $('#inputUserLocation').val('');
    $('#inputUserGender').val('');
    $('#inputActionType').val('');
    $('#inputUpdateId').val('');
    $('#inputUserName').focus();
}
```

## ● Modifier les informations d'un utilisateur

Pour modifier un utilisateur, le principe que nous allons mettre en place est le suivant. L'utilisateur clique sur le bouton update de l'utilisateur qu'il souhaite modifier. Les informations de l'utilisateur choisi sont mises dans les zones du formulaire qui sert également pour l'ajout. En cliquant sur envoyer les informations sont mises à jour dans la base de données. Il y'a donc une gestion à faire entre l'action d'ajouter et celle de modifier.

On crée une nouvelle route dans notre fichier users.js pour récupérer toutes les informations de l'utilisateur selon son id (ObjectId mongo).

```
/* GET user by id. */
router.get('/:id', function(req, res) {
    var db = req.db;
    var userToFind = req.params.id;
    var collection = db.get('userlist');
    collection.findOne({"_id": userToFind }, {}, function(e, docs) {
        res.json(docs);
    });
});
```

On renvoie le résultat en json.

Pour gérer l'action d'ajouter ou de modifier et afin de pouvoir conserver l'id de l'utilisateur pour la modification en base de données, on crée dans notre formulaire deux champs hidden.

Le premier sera l'action soit "modifier" soit "ajouter" (par défaut) et le deuxième contiendra l'id.

```
<form id="formAddUser" name="adduser" method="post" action="/users/adduser">
    <input id="inputUserName" type="text" placeholder="username" name="username" />
    <input id="inputUserEmail" type="text" placeholder="email" name="useremail" />
    <input id="inputUserFullName" type="text" placeholder="fullname" name="userfullname" />
    <input id="inputUserAge" type="text" placeholder="age" name="userage" />
    <input id="inputUserLocation" type="text" placeholder="location" name="userlocation" />
    <input id="inputUserGender" type="text" placeholder="gender" name="usergender" />
    <input id="inputActionType" name="actiontype" type="hidden" value="add">
    <input id="inputUpdateId" name="updateid" type="hidden" value="">
    <button id="btnSubmit" type="submit">Submit</button>
</form>
```

On met en place le javascript sur le front (**userlist.ejs**) pour que quand on clique sur update les informations soient bien mises dans les zones du formulaire.

```
function updateUser(id) {

    $.ajax({
        url: "/users/"+id,
        cache: false
    })
    .done(function( json ) {
        if (json!=null) {
            $('#inputUserName').val(json.username);
            $('#inputUserEmail').val(json.email);
            $('#inputUserFullName').val(json.fullname);
            $('#inputUserAge').val(json.age);
            $('#inputUserLocation').val(json.location);
            $('#inputUserGender').val(json.gender);
            $('#inputActionType').val('edit');
            $('#inputUpdateId').val(id);
            $('#inputUserName').focus();
        }

    });

}
```

On met en place le lien dans le html pour exécuter cette fonction javascript.

```
<%
var list = '';
for (i = 0; i < userlist.length; i++) {
    list += '<li><a href="#" onclick="showUser(\''+userlist[i]._id + '\')">' +
    userlist[i].username + '</a>';
    list += '<span style="margin-left:20px"><a href="#"
    onclick="updateUser(\''+userlist[i]._id + '\')">Update</a></span>';
    list += '<span style="margin-left:20px"><a href="/users/deleteuser/' +
    userlist[i]._id + '">delete</a></span>';
    list += '</li>';
}
%>
<%- list %>
```

Maintenant, il va falloir modifier notre route "adduser" pour gérer l'enregistrement en base de données que ce soit pour l'ajout ou la modification.



```
/* POST to Add User Service */
router.post('/adduser', function(req, res) {

    // Set our internal DB variable
    var db = req.db;

    // Get our form values. These rely on the "name" attributes
    var userName = req.body.username;
    var userEmail = req.body.useremail;
    var userFullName = req.body.userfullname;
    var userAge = req.body.userage;
    var userLocation = req.body.userlocation;
    var userGender = req.body.usergender;
    var actionType = req.body.actiontype;
    var updateId = req.body.updateid;

    // Set our collection
    var collection = db.get('userlist');

    // Submit to the DB
    switch (actionType) {
        case 'edit':

            collection.update({ '_id' : updateId }, {
                "username" : userName,
                "email" : userEmail,
                "fullname" : userFullName,
                "age" : userAge,
                "location" : userLocation,
                "gender" : userGender
            }, function (err) {
                if (err) {
                    // If it failed, return error
                    res.send("There was a problem updating the information to the database.");
                }
                else {
                    // And forward to success page
                    res.redirect("userlist");
                }
            });
            break;
        case 'add':
        default:
```

```

        collection.insert({
            "username" : userName,
            "email" : userEmail,
            "fullname" : userFullName,
            "age" : userAge,
            "location" : userLocation,
            "gender" : userGender
        }, function (err, doc) {
            if (err) {
                // If it failed, return error
                res.send("There was a problem adding the information to the database.");
            }
            else {
                // And forward to success page
                res.redirect("userlist");
            }
        });
        break;
    }
});

```

## ● Supprimer un utilisateur

On crée une nouvelle route avec comme paramètre l'id utilisateur pour réaliser la suppression de l'utilisateur en base de données.

```

/* DELETE to deleteuser. */
router.get('/deleteuser/:id', function(req, res) {
    var db = req.db;
    var collection = db.get('userlist');
    var userToDelete = req.params.id;
    collection.remove({ '_id' : userToDelete }, function(err) {
        if (err) {
            // If it failed, return error
            res.send("There was a problem deleting the information to the database.");
        }
        else {
            // And forward to success page
            res.redirect("/users/userlist");
        }
    });
});

```

On met en place le lien pour la suppression dans le html

```

<%
var list = '';
for (i = 0; i < userlist.length; i++) {
    list += '<li><a href="#" onclick="showUser(\''+userlist[i]._id + '\')">' +
        userlist[i].username + '</a>';
    list += '<span style="margin-left:20px"><a href="#"
        onclick="updateUser(\''+userlist[i]._id + '\')">Update</a></span>';
    list += '<span style="margin-left:20px"><a href="/users/deleteuser/' +
        userlist[i]._id + '">delete</a></span>';
    list += '</li>';
}
%>
<%- list %>

```



## Les tests sont essentiels au bon fonctionnement de l'application

Toutes ces fonctionnalités sont à tester et à vérifier y compris dans la base de données.

## ● Afficher les informations d'un utilisateur quand on clique sur celui-ci

On va créer un encart pour afficher les informations de l'utilisateur quand on clique sur celui-ci.

```
<div id="showUser" style="width:20%;float:left">
  <strong>Name : </strong><span id='userInfoName'></span>
  <br>
  <strong>Age : </strong><span id='userInfoAge'></span>
  <br>
  <strong>Gender : </strong><span id='userInfoGender'></span>
  <br>
  <strong>Location : </strong><span id='userInfoLocation'></span>
</div>
```

On met place le javascript nécessaire qui, sur l'évènement clic, va chercher les informations et met à jour les valeurs dans les éléments html dans le template **userlist.ejs**.

```
<%
var list = '';
for (i = 0; i < userlist.length; i++) {
  list += '<li><a href="#" onclick="showUser(\''+userlist[i]._id + '\')">' +
    userlist[i].username + '</a>';
  list += '<span style="margin-left:20px"><a href="#"
    onclick="updateUser(\''+userlist[i]._id + '\')">Update</a></span>';
  list += '<span style="margin-left:20px"><a href="/users/deleteuser/' +
    userlist[i]._id + '">delete</a></span>';
  list += '</li>';
}
%>
<%- list %>
```

```
function showUser(id) {
  $.ajax({
    url: "/users/"+id,
    cache: false
  })
  .done(function( json ) {
    if (json!=null) {
      $('#userInfoName').html(json.fullname);
      $('#userInfoAge').html(json.age);
      $('#userInfoGender').html(json.location);
      $('#userInfoLocation').html(json.gender);
    }
  });
}
```

## 5. Le code source de l'exemple

---

Le code est dans ce fichier zip (cf. *code\_source.zip*)

## 6. Pour aller plus loin

---

Nous avons bien remarqué que les échanges entre le front et le back peuvent se faire avec des requêtes AJAX, que les routes peuvent retourner du JSON. Il suffit de changer le format de la réponse.

Au lieu de renvoyer vers une vue

```
res.render('userlist', {  
  "userlist" : docs  
});
```

On envoie du json.

```
res.json(docs);
```

Cela veut dire que la création d'un webservices serait facilité et aisé à mettre en place dans l'univers Javascript, sur Node JS et express.

Il faut néanmoins se pencher également sur la sécurité du webservices en Javascript pour verrouiller un peu les choses.

C'est un autre sujet.

Enjoy !

## Crédits

---

© AFPA

### **Reproduction interdite**

*Article L 122-4 du code de la propriété intellectuelle.*

« Toute représentation ou reproduction intégrale ou partielle faite sans le consentement de l'auteur ou de ses ayants droit ou ayants cause est illicite. Il en est de même pour la traduction, l'adaptation ou la reproduction par un art ou un procédé quelconques. »