

# Prérequis **Atelier n°6**

Node.js, Express.js et MongoDB


# Pré-requis

- Node.js : ^6.9.0
- Express.js: ^4.12.0
- Mongoose: ^4.4.0



Objectifs:  
**Objectifs:**


- Se connecter à la base des données MongoDB via le module mongoose
- Ajouter des données via un formulaire (POST) dans la base des données
- Afficher des données (GET)



# Configuration du projet(I/4)

## Configuration du projet(I/4)

- Commencez par installer le générateur Express, qui sera utilisé pour créer un projet de base pour nous: ***\$ npm install -g express-generator***
- ➔ le flag **-g** signifie que nous allons l'installer sur l'ensemble de notre système.



# Configuration du projet(2/4)

## Configuration du projet(2/4)


- Accédez au répertoire du travail, comme votre "bureau" ou "Documents", puis créer votre application:

**\$ *express node-mongoose-form***

- Vérifiez la structure du projet:

```
├─ app.js
├─ bin
│   └─ www
├─ package.json
├─ public
│   ├── images
│   ├── javascripts
│   └── stylesheets
│       └─ style.css
├─ routes
│   ├── index.js
│   └─ users.js
└─ views
    ├── error.jade
    ├── index.jade
    └─ layout.jade
```





# Configuration du projet(3/4)

## Configuration du projet(3/4)

- Le fichier **package.json** stocke les dépendances de notre projet, que nous avons encore besoin d'installer:


```
$ cd node-mongoose-form
```

```
$ npm install
```

Maintenant, nous allons installer une dernière dépendance:

```
$ npm install mongoose --save
```

➔ Le flag **--save** ajoute les dépendances et leurs versions au fichier **package.json**.



# Configuration du projet(4/4)

## Configuration du projet(4/4)

- Testons notre configuration en exécutant l'application:  
***\$ npm start***
- Naviguer vers [http: //localhost:3000/](http://localhost:3000/) dans votre navigateur et vous devriez voir le texte «Welcome to Express».

## Supervisor(1/4)

- Je recommande fortement la mise en place Supervisor de sorte que vous

pouvez exécuter votre application et surveiller les changements de code.

```
$ npm install supervisor -g
```

## Supervisor(2/4)

- Tuer le serveur en appuyant sur CTRL-C.
- Une fois installé, nous allons mettre à jour le fichier de **package.json** afin d'utiliser superviseur pour exécuter votre programme.

# Supervisor(3/4)

- Il suffit de changer ceci: **"scripts": {  
"start": "node ./bin/www"  
},**
- À celle-ci:  
**"scripts": {  
"start": "supervisor ./bin/www"  
},**


# Supervisor(4/4)

- Testons à nouveau:  
***\$ npm start***
- Dans votre console, vous devriez voir:

```
Watching directory 'node-mongoose-form' for changes.
```

Si vous voyez cela, vous savez que ça fonctionne bien.





# Routes(1/6)

## Routes(1/6)



- Ouvrez le fichier principal, **app.js**, qui abrite « the business logic »:

```
app.use('/', routes);  
app.use('/users', users);
```



# Routes(2/6)

## Routes(2/6)

## Voyons ensemble la route:

```
app.use('/users', users)
```

- Regardons comment Node.js gère cette logique de « handling routes ».
- On associe avec cette route, une variable **users** qui fait référence au fichier users.js

```
var users = require('./routes/users');
```



# Routes(3/6)

## Routes(3/6)

- Ouvrons ce fichier users.js:

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

/* GET users listing. */
router.get('/', function(req, res) {
  res.send('respond with a resource');
});

module.exports = router;
```

Quand la route **/users** est appelée, le message « respond with a resource » s'affiche!



# Routes(4/6)

## Routes(4/6)

## I. Ajout d'une route:

- Ajoutons maintenant une nouvelle route qui rend un formulaire HTML à l'utilisateur final.
- Commençons par ajouter le gestionnaire de route(the route handler) dans le fichier app.js:

```
app.use('/form', form);
```

Cela signifie: ***app.use('/ENDPOINT', VARIABLE\_NAME);***



# Routes(5/6)

## Routes(5/6)



- Définir la variable **form** pour renseigner le fichier JS au sein de notre dossier routes.

```
var form = require('./routes/form');
```

Regardons votre console, vous devriez voir une erreur, disant « Node can't find that './routes/form' module. Nous avons besoin de le créer.

- Créer ce fichier appelé from.js sous le répertoire "routes". Ajoutez le code suivant:

# Routes(6/6)

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

/* GET form. */
router.get('/', function(req, res) {
  res.send('My form');
});

module.exports = router;
```

- Allez vers <http://localhost:3000/form>: le message « My form » est affiché

# Twig: le moteur de template(1/3)

- Installer twig: **npm install twig --save**
- Créer un nouveau fichier « form.twig » sous le répertoire views:

```
{% extends 'layout.twig' %}  
{% block content %}  
  
    <h1>{{title}}</h1>  
  
{% endblock %}
```

{{title}} est une variable renvoyée à partir de ./routes/form.js

# Twig: le moteur de template(2/3)

- Modifions le fichier form.js de :  
`res.send('My form');`
- À:
- `res.render('form.twig', { title: 'My form' });`
  - Cela signifie qu'en tapant /form, retourner le fichier form.twig et afficher "My form" comme titre
- Actualisez <http://localhost:3000/form>.

# Twig: le moteur de template(3/3)

```
{% extends 'layout.twig' %}

{% block content %}
    <h1>{{title}}</h1>
    <form action="/create" method="post">
        <label for="comment" >Add a comment</label>
        <input type="text" name="comment" placeholder="Add a comment" />
        <input type="submit" value="Save" />
    </form>
```

- Modifions le fichier form.twig

Actualisez <http://localhost:3000/form> et vous allez avoir un formulaire

## Ajout de route handler pour /create

- Ajoutez une nouvelle route dans app.js:

```
app.use('/create', form);
```

Ouvrez form.js pour ajouter la logique de cette nouvelle route:

```
/* POST form. */  
router.post('/', function(req, res) {  
  console.log(req.body.comment);  
  res.redirect('form');  
});
```



Testez la page, lorsque vous soumettez le formulaire la valeur de `req.body.comment` s'affiche dans la console

## Configuration de Mongoose(1/3)

- Mongoose est un logiciel de modélisation d'objet pour Node.js qui fonctionne essentiellement comme un ORM que vous verriez dans d'autres langues (comme Eloquent pour Laravel).
- Commençons par définir le schéma, qui correspond ensuite à une collection à Mongo.

# Configuration de Mongoose(2/3)

- Créez le fichier *database.js* sous le répertoire *database* dans le répertoire racine de projet et ajoutez le code suivant:

```
var mongoose = require('mongoose');
var Schema   = mongoose.Schema;

var Comment = new Schema({
  title : String,
});

mongoose.model('comments', Comment);

mongoose.connect('mongodb://localhost/node-comment');
```

Nous avons inclus le module Mongoose avec une référence de Schema. On commence toujours par définir le schéma puis on le relie avec une collection « comments ».



Finalement, on ouvre une connection d'une instance de notre base des données locale.

## Configuration de Mongoose(3/3)

- Ouvrir le serveur base des données mongodb dans une autre console
- Puis, ouvrez le fichier *app.js* et ajoutez cette ligne:

```
// mongoose config  
var db = require('./database/database.js');
```

On a besoin de mettre à jour le fichier *form.js* pour ajouter(via POST) et de lire(via

GET) les données(data) à partir de la collection Mongo.

## Handling form GET requests(1/2)

- Ouvrez form.js et ajoutez le code suivant:

```
var mongoose = require('mongoose');  
var Comment = mongoose.model('comments');
```

Puis modifiez la fonction qui traite la requête GET:

```
/* GET form. */
router.get('/', function(req, res) {
  Comment.find(function(err, comments){
    console.log(comments)
    res.render(
      'form.twig',
      {title : 'My form', comments : comments}
    );
  });
});
```

**comment.find()** saisit tous les commentaires de la collection Mongo et les affecter à la variable **comments**. Cette variable est envoyée à la vue form.twig

## Handling form GET requests(2/2)

- Ajoutons une boucle pour parcourir les commentaires, puis afficher la clé « titre » de la collection:

```
{% extends 'layout.twig' %}
{% block content %}

<h1>{{title}}</h1>
<form action="/create" method="post">
    <label for="comment"> Add a comment</label>
    <input type="text", name="comment", value="comment" />
    <input type="submit", value="Save" />
</form>
{% for comment in comments %}
    <h3>{{comment.title}}</h3>

{% endfor%}
{% endblock %}
```

## Handling form POST requests

- Modifier le fichier *form.js* et mettre à jour la fonction traitant la requête POST:

```
/* POST form. */
router.post('/', function(req, res) {
  new Comment({title : req.body.comment})
  .save(function(err, comment) {
    console.log(comment)
    res.redirect('form');
  });
});
```

Nous enregistrons un nouveau commentaire en récupérant la valeur via ***req.body.comment***



Travail à faire:  
**Travail à faire:**



- 
- Faire les modifications nécessaires pour :
    - supprimer un commentaire
    - Modifier un commentaire