NODEJS

WEB & Express

Web

- Création d'application Web avec NODEJS
- Framework Express : http://expressjs.com
- Minimaliste, simple et puissant

Express Generator

- Générateur de squelette d'application basé sur express

```
$ npm install --global express-generator
```

- Utilisation simple. Se placer dans le répertoire ou l'on veut créer le projet et taper la commande **express :**

```
$ cd projet/
$ express
   create : .
   create : ./package.json
   create : ./app.js
```

Express Generator

Une fois l'installation des dépendances effectuée (`npm install`), il est possible de lancer le projet généré avec la commande spécifiée dans la trace précédente :

```
run the app:
  $ DEBUG=projet:* ./bin/www
```

```
$ DEBUG=projet:* ./bin/www
  projet:server Listening on port 3000 +0ms
GET / 200 305.513 ms - 170
GET /stylesheets/style.css 200 16.825 ms - 110
GET /favicon.ico 404 41.690 ms - 1046
GET /favicon.ico 404 14.939 ms - 1046
```

L'ouverture d'un navigateur sur le port 3000 (http://localhost:3000) affiche une page d'accueil avec le titre « Welcome to Express ».

Express

- Utiliser express pour tout construire

```
$ npm install --save express
```

- Module **express**

```
// Import du module express.
var express = require('express');

// Création de l'instance de l'application.
var app = express();
```

Express

Une fois l'application instanciée, elle peut être associée à un serveur HTTP afin de gérer les requêtes entrantes :

```
// Utilisation du module standard http pour créer le serveur HTTP.
var http = require('http');

// Création du serveur et association avec l'application Express.
var server = http.createServer(app);

// Association du port 80 au serveur HTTP.
server.listen(80);
```

Express fournit la méthode `listen()` pour simplifier encore un peu plus la vie du développeur :

```
// Instancie un serveur HTTP sur le port 80 et l'associe à
// l'application.
app.listen(80);
```

Express

Architecture

L'architecture d'Express est basée sur l'exécution en cascade de middlewares à chaque requête entrante.

Un middleware, pour Express, est une fonction qui reçoit en paramètres la requête et la réponse courantes ainsi que le middleware suivant.

```
function foo(request, response, next) {
  // Affiche la requête courante.
  console.log('Requête', request.method, 'sur', request.path);

  // Passe la main au middleware suivant.
  next();
}
```

```
// Associe le middleware foo à l'application.
app.use(foo);
```

Express

- Code complet

```
2  var express = require('express');
3  var http = require('http');
4
5  var app = express();
6
7  function foo(request, response, next) {
8
9    console.log('Requête', request.method, 'sur', request.path);
10
11   next();
12 }
13
14 app.use(foo);
15
16 var server = http.createServer(app);
17
18 app.listen(8000);
```

Si l'on effectue une requête sur le serveur grâce à un navigateur (http://localhost:80), on peut voir le message d'erreur « Cannot GET / » s'afficher car le middleware `foo()` n'a pas répondu à la requête. Dans la console du serveur, par contre, on voit la trace de `console.log()`

Express

Middlewares à ajouter à express

- Template engine pour express (Jade, Esj, etc…)
- **body-parser** pour traiter le contenu des requêtes
- **cookie-parser** pour traiter les cookies
- Indiquer à express ou se trouvent les fichiers statiques (images, css, polices, etc…)

Express

Templates Engines : Jade, ESJ, etc…

```
doctype html
html(lang="en")
  head
    title= pageTitle
    script(type='text/javascript').
      if (foo) {
        bar(1 + 5)
      }
  body
    h1 Jade - node template engine
    #container.col
      if youAreUsingJade
        p You are amazing
      else
        p Get on it!
      p.
        Jade is a terse and simple
        templating language with a
        strong focus on performance
        and powerful features.
```

```
app.set('view engine', 'ejs');
app.set('views', __dirname + '/views');
```

```
<!DOCTYPE html>
<html>
  <head>
    <title>Qui ? : <%=who%></title>
    <%-scripts%>
    <%-stylesheets%>
  </head>
  <body>
    <header>
      <%-blocks.header%>
    </header>
    <section>
      <%-body -%>
    </section>
    <footer>
      <%-blocks.footer%>
    </footer>
  </body>
</html>
```

Express

- **body-parser** pour traiter le contenu des requêtes

```
npm install --save body-parser

app.use(bodyParser.json());
app.use(bodyParser.urlencoded());
```

- **cookie-parser** pour traiter les cookies

```
npm install --save cookie-parser

app.use(cookieParser());
```

Express

- Indiquer à express ou se trouvent les fichiers statiques (images, css, polices, etc…)

```
app.use(express.static(path.join(__dirname, '')));
```

Dans cet exemple, on indique a express que les fichiers statiques se trouvent à la racine du serveur.

Express

- Paramètre requête
  - Permet d’accéder à la requête faite à l’application.
    - Méthode HTTP employée (GET, POST, etc…)
    - Chemin complet de la requête.

Vous trouverez toutes les informations sur l’objet Request à la page suivante : http://expressjs.com/4x/api.html#request.

Express

- Paramètre requête
  - Contenu de la requête (corps).
  - Cette propriété n'est pas remplie par défaut, il faut utiliser un middleware pour cela. Par exemple body-parser (http://npmjs.com/package/body-parser)

```
var bodyParser = require('body-parser');

// Prise en charge du JSON.
app.use(bodyParser.json());

// Prise en charge des formulaires HTML.
app.use(bodyParser.urlencoded());

// Création d'un middleware qui va afficher le corps de la requête.
app.use(function (req, res, next) {
  console.log(req.body);

  next();
});
```

Express

Le paramètre ***Response***

- Permet d'interagir sur la réponse
- Envoyer du texte grâce à la méthode ***send()***

```
app.use(function (req, res, next) {
  res.send('Hello world!');
});
```

Dans cet exemple, on peut constater qu'il ne faut pas appeler le middleware suivant (`next()`) dans le cas où la requête a fini d'être traitée.

Express

Le paramètre ***Response***

- Express permet aussi de renvoyer un objet JSON directement

```
app.use(function (req, res, next) {
  res.json({
    foo: 'bar'
  });
});
```

Express

Le paramètre ***Response***

- Il existe même une méthode pour proposer un fichier au téléchargement.

```
app.use(function (req, res, next) {
  res.download('/report.pdf');
});
```

Express

Routage : app.all()

```
// Sert la page d'accueil.
app.all('/', function (req, res) {
  res.send('Accueil');
});

// Sert la page À propos.
app.all('/about', function (req, res) {
  res.send('À propos');
});
```

La méthode `all()` permet d'associer un middleware à un chemin, et ce, quelle que soit la méthode HTTP utilisée. Mais il est aussi possible de préciser la méthode avec les méthodes spécialisées (`get()`, `post()`, `put`, `delete`, etc.)

Express

- Récupération de paramètres GET

```
http://www.monsite.com/folder?id=100022

app.get('/folder',function(req,res){

        var folder_id = req.query.id;

});
```

Express

- Récupération des paramètres POST

```
http://www.monsite.com/connect

app.post('/connect'.function(req,res){
    var email    =    req.body.email;
    var password  = req.body.password;

});
```

Express

- Retourner du JSON

```
app.get('/disonnect',function(req,res){

        res.json({"disconnect":"yes"});

});
```

Lorsque l'on effectue des appels ajax, on ne désire recevoir qu'un code de retour, voir un objet JSON. La fonction .json() permet de retourner directement du JSON.