Express.js

- Express est un Framework Web Node.js minimaliste et flexible qui fournit un ensemble de fonctionnalités robuste pour développer des applications Web et mobiles.

- Il permet de configurer des middlewares pour répondre aux requêtes HTTP.

- Il définit une table de routage utilisée pour effectuer différentes actions basée sur HTTP et URL.

- Il permet de retourner dynamiquement des pages HTML.

Utilisation de middlewares

- Express est un Framework basé sur le concept de middlewares.
- Ce sont des petits morceaux d'application qui rendent chacun un service spécifique.
- Express est fourni avec une quinzaine de middlewares de base, et d'autres développeurs peuvent bien entendu en proposer d'autres via NPM.
- Ces middlewares sont interconnectés dans un pipe line et peuvent communiquer entre eux.
- Tous ces middlewares communiquent entre eux en se renvoyant jusqu'à 4 paramètres :
  - error : les erreurs
  - request : la requête du visiteur
  - response : la réponse à renvoyer (la page HTML et les informations d'en-tête)
  - next : un callback vers la prochaine fonction à appeler

Middleware niveau application

- Le middleware niveau application est lié à une instance de l'objet app.object en utilisant les fonctions app.use() et app.METHOD(), où METHOD est la méthode HTTP de la demande que gère la fonction middleware (par exemple GET, PUT ou POST) en minuscules.

app.get (chemin [, middleware], callback [, callback ...])

app.put (chemin [, middleware], callback [, callback ...])

app.post (chemin [, middleware], callback [, callback ...])

app.delete (chemin [, middleware], callback [, callback ...])

app.use (chemin [, middleware], callback [, callback ...])

app.use (rappel)

Paramètres

Paramètre	Détails
chemin (path)	Spécifie la portion de chemin ou l'URL que le rappel donné va gérer.
middleware	Une ou plusieurs fonctions qui seront appelées avant le rappel. Essentiellement un chaînage de plusieurs fonctions de callback. Utile pour une manipulation plus spécifique, par exemple une autorisation ou un traitement des erreurs.
callback	Une fonction qui sera utilisée pour gérer les demandes sur le path spécifié. Il sera appelé comme callback (request, response, next).
request	Un objet encapsulant des détails sur la requête HTTP que le rappel est appelé à gérer.
response	Un objet utilisé pour spécifier comment le serveur doit répondre à la demande.
next	Un rappel qui passe le contrôle au prochain itinéraire correspondant. Il accepte un objet d'erreur facultatif.

Installer Express

- Pour installer Express, utiliser la commande :

npm install express --save

- On commence par demander l'inclusion d'Express et on crée un objet app en appelant la fonction express().
- La méthode listen prend comme premier paramètre le numéro de port à écouter et comme second paramètre un callback qui permet d'afficher un message sur le serveur.

- ***Exemple :***

```
var express = require('express') ;
var app = express() ;
var port = 8888 ;

app.listen(port, function(){
  console.log('Le serveur fonctionne sur le port ',port) ;
})
```

Mise en place d'un routage simple avec Express

- Il suffit d'indiquer les différentes routes (les différentes URL) à laquelle l’application doit répondre.
- Pour définir le chemin d'accès à une page, il faut utiliser une méthode de requête HTTP spécifique (GET, POST, etc.).
- Ainsi, pour créer une route, par exemple la racine "/", une fonction de callback est appelée quand quelqu'un demande cette route.

- ***Exemple :***

```
var express = require('express') ;
var app = express() ;
var port = 8888 ;

app.listen(port, function(){
  console.log('Le serveur fonctionne sur le port '+port) ;
}) ;
app.get('/', function(req, res){
  res.send('Bonjour tout le monde') ;
}) ;
```

- L'avantage de la méthode get est qu'elle renvoie à son tour un objet similaire ; les appels peuvent donc être chaînés.
- ***Exemple :***

```
var express = require('express') ;
var app = express() ;
var port = 8888 ;
app.get('/', function(req, res){
  res.send('Bonjour tout le monde') ;  }) ;
app.get('/page1', function(req, res){
  res.send('Ceci est la page 1') ; })
app.get('/page2', function(req, res){
  res.send('Ceci est la page 2') ; }) ;
app.get('/page3', function(req, res){
  res.send('Ceci est la page 3') ; }) ;
app.listen(port, function(){
  console.log('Le serveur fonctionne sur le port '+port) ;  }) ;
```

Routes dynamiques

- Express permet de gérer des routes dynamiques, c'est-à-dire des routes dont certaines parties sont variables.
- On doit écrire:nomvariable dans l'URL de la route, ce qui aura pour effet de créer un paramètre accessible depuis req.params.nomvariable

- ***Exemple :***

```
var express = require('express') ;
var app = express() ;
var port = 8888 ;
app.get('/', function(req, res){
  res.send('Bonjour tout le monde') ;  }) ;
app.get('/page:num', function(req, res){
  res.send('Ceci est la page '+req.params.num) ;
}) ;
app.listen(port, function(){
  console.log('Le serveur fonctionne sur le port '+port) ;  }) ;
```

Gestion des erreurs 404

- Express permet aussi de gérer facilement les entêtes http, et notamment les réponses du serveur en cas d'erreur, la plus connue étant l'erreur 404 (lorsqu'une ressource ne peut pas être trouvée sur le serveur).

- ***Exemple :***

```
var express = require('express') ;
var app = express() ;
var port = 8888 ;
app.get('/', function(req, res){
  res.send('Bonjour tout le monde') ;
}) ;
app.use(function(req, res, next){
  //la ligne suivante spécifie l'encodage de la réponse
  res.setHeader('Content-Type', 'text/plain; charset=UTF-8') ;
  //La ligne suivante spécifie le message à envoyer par le serveur quand la réponse est un code d'erreur 404
  res.status(404).send('Page introuvable') ;
});
app.listen(port, function(){
  console.log('Le serveur fonctionne sur le port '+port) ;
});
```

Application

- Créer un projet node à l'aide de npm init, pour créer un projet.
- Ajouter Express comme dépendance.
- Afficher le texte "Tout marche bien" en réponse à une requête sur le port 8888 de localhost. Préciser l'encodage de la réponse.
- Afficher le texte "Cette page n'existe pas" en cas d'erreur 404.
- Afficher le texte "Tout marche bien pour cette page de test 1" sur la page localhost:8888/test1
- Afficher le texte "Tout marche bien pour cette page :x" sur la page localhost:8888/parametre:x où x désigne une valeur précisée par l’utilisateur.

app.route()

- On peut créer des gestionnaires de routage sous forme de chaîne pour un chemin de routage en utilisant **app.route().**
- Etant donné que le chemin est spécifié à un seul emplacement, la création de routes modulaires est utile car elle réduit la redondance et les erreurs.
- Voici quelques exemples de gestionnaires de chemin de chaînage définis à l’aide de app.route().

Exemple :

```
app.route('/book')
  .get(function(req, res) {
    res.send('retourner un livre');
  })
  .post(function(req, res) {
    res.send('Ajouter un livre');
  })
  .put(function(req, res) {
    res.send('MAJ livre');
  });
```

express.Router

- Utiliser la classe **express.Router** permet de créer des gestionnaires de routes modulaires et pouvant être montés.
- Une instance **Router** est un middleware et un système de routage complet ; pour cette raison, elle est souvent appelée « mini-app ».

- L'exemple suivant crée un routeur en tant que module, charge une fonction middleware, définit des routes et monte le module de routeur sur un chemin dans l'application principale.

- ***page2.js :***

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

// middleware spécifique à ce routeur
router.use(function timeLog(req, res, next) {
  console.log('Time: ', Date.now());
  next();
});
// définir l'itinéraire de la page d'accueil
router.get('/', function(req, res) {
  res.send('Cette page est la page home');
});
// définir l'itinéraire de la page detail
router.get('/detail', function(req, res) {
  res.send('Détail de la page');
});

module.exports = router;
```

- Puis, charger le module de routage dans l'application ***app.js*** :

```
var express = require('express');
var app = express();
var page2 = require('./page2');
var server = app.listen(3000, function () {
    var port = server.address().port
    console.log("Serveur sur le port  "+port) })
app.use('/page2', page2);
```

- L'application pourra dorénavant gérer des demandes dans /page2 et /page2/detail, et appeler la fonction middleware timeLog spécifique à la route.

Servir des fichiers statiques

- Express fournit un middleware intégré express.static pour servir des fichiers statiques, tels que des images, CSS, JavaScript, etc.
- Il suffit de transmettre le nom du répertoire dans lequel on conserve les actifs statiques au middleware **express.static** pour commencer à servir les fichiers directement.
- Par exemple, si on conserve les images, CSS et fichiers JavaScript dans un répertoire nommé public, on écrit :

```
app.use(express.static('public'));
```

- Express recherche les fichiers relatifs au répertoire statique, donc le nom du répertoire statique ne fait pas partie de l'URL.

- ***Exemple*** :

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

app.use(express.static('public'));

app.get('/', function (req, res) {
   res.send('Bonjour');
})
 var server = app.listen(8081, function () {
      var port = server.address().port
      console.log("Serveur sur le port  "+port) })
```

Ouvrir http://127.0.0.1:8081/images/logo.png dans le navigateur pour visualiser l'image.

Méthode GET

- Dans un formulaire utilisant la méthode GET, on utilise “process_get” pour gérer l’entrée correspondante.
- ***Exemple :***

index.html

```
<html>
  <body>
    <form action = "http://127.0.0.1:8081/process_get" method = "GET">
      Nom : <input type = "text" name = "nom">  <br>
      Prénom : <input type = "text" name = "prenom">
      <input type = "submit" value = "Valider">
    </form>
   </body>
</html>
```

Serveur.js :

```
var express = require('express');
var app = express();
app.use(express.static('public'));
app.get('/index.html', function (req, res) {
  res.sendFile( __dirname + "/" + "index.html" ); })
app.get('/process_get', function (req, res) {
  // Préparer un output en format JSON
  response = {
    nom:req.query.nom,
    prenom:req.query.prenom
  };
  console.log(response);
  res.end(JSON.stringify(response)); })
  var server = app.listen(8081, function () {
    var port = server.address().port
    console.log("Serveur sur le port  "+port) })
```

Méthode POST

- Dans un formulaire utilisant la méthode POST, il faut utiliser **process_post** pour gérer l'entrée correspondante.
- ***Exemple :***

index.html

```
<html>
  <body>
    <form action = "http://127.0.0.1:8081/process_post" method = "POST">
      Nom : <input type = "text" name = "nom">  <br>
      Prénom : <input type = "text" name = "prenom">
      <input type = "submit" value = "Valider">
    </form>
   </body>
</html>
```

Serveur.js :

```
var express = require('express');
var app = express();
//body-parser est un middleware express qui lit les entrées d'un formulaire et les stocke en tant qu'objects
javascript accessibles via req.body 'body-parser' doit être installé (via npm install --save body-parser )
var bodyParser = require('body-parser');
// Créer une  application/x-www-form-urlencoded parser
var urlencodedParser = bodyParser.urlencoded({ extended: false })
app.get('/index.html', function (req, res) {
  res.sendFile( __dirname + "/" + "index.html" ); })
app.post('/process_post', urlencodedParser, function (req, res) {
     response = {
     nom:req.body.nom,
     prenom:req.body.prenom    };
  console.log(response);
  res.end(JSON.stringify(response)); })
var server = app.listen(8081, function () {
    var port = server.address().port
 console.log("Serveur sur le port  "+port) })
```