

9. Express.js

Jean-Luc Falcone

Août 2019

Express.js (2010-)

Fast,
unopinionated,
minimalist web framework
for Node.js

Hello, World !

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

app.get('/', (req, res) => res.send('Hello World!'));

app.listen(3000,
  () => console.log('Example app listening on port 3000!')
);
```

Anatomie

- `app.verb(path, function(req, res, [next]) {...})`

`verb` est un verbe http utilisé par le client

`path` est le chemin de la ressource appelée par le client

`req` est un objet permettant d'accéder à la requête

`res` est un objet permettant de construire la réponse

`next` arg. optionnel permettant de chaîner des routes

Verbes

```
app.get(...);
```

```
app.post(...);
```

```
app.put(...);
```

```
app.delete(...);
```

```
app.all(...); //Tous les verbes
```

Chemins

- Les chemins sont représentés par des chaînes de caractères (avec caractères spéciaux) ou des regex.
- Dans une chaînes de caractères ?, +, * ainsi que les parenthèses fonctionnent comme pour les regex.
- Les paramètres de requêtes ne font pas partie du chemin.

Exemples

```
app.get( "/blog/posts", ... );  
app.get( "/info.html", ... );  
app.get( "/number/0+1", ... );  
app.get( /info.html/, ... );
```

Paramètres dans le chemin

- On peut définir des paramètres à partir des éléments d'un chemin en précédant un identifiant ': '.
- L'objet `req.params` permet d'accéder à ces valeurs.
- Par exemple, la route:

```
app.get( "/blog/post/:postID/comment/:commentID",  
  function(req,res) {  
    console.log( req.params.postID + " "  
                  + req.params.commentID );  
  }  
);
```

reconnaitra le chemin `/blog/post/123/comment/2`

Paramètres dans le chemin (2)

On peut utiliser des points et des tirets dans les chemins pour séparer les paramètres.

Par exemple:

```
app.get( "/download/file/:name.:extension", ... );
```

Permettra de récupérer séparément un nom et une extension, pour générer un format de sortie à la demande.

Paramètres dans le chemin (3)

On peut aussi passer une regex pour valider chaque élément du chemin:

```
app.get( "/blog/post/:postID(\\d+)/comment/:comID(\\d+)",  
  ...  
);
```

Ne pas oublier de doubler les backslashes dans la chaîne de caractères.

Objet request

`req.originalUrl` URL d'origine

`req.query` *Query parameters* de l'URL

`req.get(...)` Accède à un *header* HTTP par clé

`req.body` Contenu de la requête (requiert un *body parser*)

`req.cookie` Contenu de la requête (requiert un *cookie parser*)

Objet response

`res.status(...)` Code de retour

`res.json(...)` Converti et Envoie une valeur en json

`res.send(...)` Envoie une valeur de type arbitraire

`res.redirect(...)` Redirections

`res.sendFile(...)` Envoie un fichier

`res.end(...)` Termine le traitement de la réponse.

`res.set(...)` Définit un header HTTP

Objet response: Exemple

```
res.json({ user: 'tobi' });
```

```
res.status(500).json({ error: 'message' });
```

```
res.status(404).send('<p>Not Found</p>');
```

```
res.redirect(301, "/blog/post/1234");
```

Regrouper les routes

```
app.route('/blog/post/:postID')  
  .get(function (req, res) {  
    ...  
  })  
  .post(function (req, res) {  
    ...  
  })  
  .put(function (req, res) {  
    ...  
  })
```

Routeurs explicites

File articles.js

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

router.get('/', function (req, res) {
  //list all articles
});
router.get('/:articleID', function (req, res) {
  //send a single article
});
module.exports = router;
```

File main.js

```
const articles = require('./articles')
app.use('/articles', const)
```

Middleware

Dans la terminologie `express.js` un middle-ware est une fonction intermédiaire qui peut s'interposer entre deux call-backs.

Middleware: definition

Il s'agit juste d'une fonction avec les 3 paramètres vus ci-dessus:

```
function logURL( req, res, next ) {  
  console.log( "GOT URL: " + req.originalURL );  
  next();  
}
```


Middleware: Utilisation

//Au niveau de toute l'application, pour chaque route:

```
app.use( logURL );
```

//Au niveau de toute l'application, pour un chemin:

```
app.use( "/blog/post/:id", logURL );
```

//Pour une route seule :

```
app.delete( "/blog/post/:id", logURL,  
            function(req,res) { ...} );
```

//Au niveau d'un routeur:

```
const app = express();  
const router = express.Router();  
router.use( logURL );
```

Middleware configurable

```
function logURL( message ) {  
  return function( req, res, next ) {  
    console.log( message + " " + req.originalURL );  
    next();  
  }  
}  
  
app.use( logURL( "The URL is" ) );
```

Middleware de gestion des erreurs

Prennent un argument `err` qui contient des infos sur l'erreur:

```
app.use(function (err, req, res, next) {  
  console.error(err.stack);  
  res.status(500).send("Guru meditation...");  
});
```

Middlewares Prédéfinis: static

static permet de servir des fichiers statiques. Par exemple pour servir les fichier d'un répertoire nommé public:

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

Plusieurs options existent pour gérer les droits, le contrôle du cache, etc.:

```
app.use(express.static(  
  'icons',  
  { index: false, immutable: true,  
    maxAge: 600*1000 }  
));
```

Middlewares Prédéfinis: json

Pour pouvoir utiliser des requêtes, PUT et POST, il faut un *Body Parser*. A partir de la version 4.16, `express.json` est prédéfini:

```
const app = express();  
app.use( express.json() );  
  
app.post('/blog/posts', function(req, res) {  
    //req.body contient un objet JSON  
    ...  
});
```

Autre middlewares

Il existe de nombreux middlewares disponibles sous la forme de paquets NPM:

`body-parser` parse d'autre formes de contenus

`cookie-parser` permet de gérer les cookies

`timeout` ajoute un *timeout* pour la gestion des requêtes

`passport` authentification (OAuth, OpenID, etc.)

`cors` *Cross-Origin Resource Sharing*, permet de partager des ressources depuis un autre domaine.