Express.js

m2iformation.fr

Sommaire

UTOPIOS
TRANSFORMATION DIGITALE INTELLIGENTE

Introduction

Définition

Express.js est un framework web pour Node.js qui fournit des fonctionnalités telles que :

- La gestion des **routes**, des **requêtes** et des **réponses**
- Le support des **middleware**, des **modules** qui peuvent modifier ou enrichir les requêtes et les réponses
- La facilitation du développement d'applications web dynamiques avec des **moteurs de template**
- L'intégration avec des **bases de données** et des services externes
- La création **d'API RESTful** avec une syntaxe simple et expressive

Installation

Initialiser une application Node.js :

```
npm init
```

Installer le package express :

```
npm install express
```

Il suffit maintenant d'importer express dans un fichier JavaScript

```
const express = require("express");
```

Création d'une application Express

Voici un exemple d'application Express :

```
const express = require("express");
const app = express();
const port = 3000;

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

app.listen(port, () => {
  console.log(`Example app listening on port ${port}`);
});
```

UTOPIOS
TRANSFORMATION DIGITALE INTELLIGENTE

Routing

Définition du routing

- Le routage consiste à déterminer la manière dont une application répond à une demande d'un client vers un point de terminaison particulier, qui est un **URI** (ou route) et **une méthode de requête HTTP spécifique** (GET, POST, etc.)
- Chaque route peut avoir **une ou plusieurs fonctions de traitement**, qui sont exécutées lorsque la route correspond
- Par exemple, dans `http://foo.com/products/id`, `/products/id` est la route

Déclaration d'une route

- Sous Express elle prend cette forme:

```
app.METHOD(PATH, HANDLER);
```

- **app** est une instance d'express
- **METHOD** est une méthode de requête HTTP, en minuscules
- **PATH** est un chemin sur le serveur
- **HANDLER** est la fonction exécutée lorsque l'itinéraire est trouvé

```
app.get("/", (req, res) => {
  res.send("Hello World!");
});
```

Les méthodes de express

Express prend en charge les méthodes correspondant à toutes les méthodes de requête HTTP : get, post, put, delete , patch.

- app.all() : permet de définir un middleware pour toutes les méthodes http
- app.use() : Monte la ou les fonctions d'un middleware sur le chemin spécifié: la fonction de l'intergiciel est exécutée lorsque la base du chemin demandé correspond au chemin

Les paramètres des routes

- Les routes peuvent être des chaînes, des paternes de chaines ou encore des expression régulières
- Express utilise le package path-to-regexp pour définir ses routes
- Voici un exemple de route simple:

```
app.get("/about", (req, res) => {
  res.send("about");
});
```

Routes basées sur des paternes de chaines

- Cette route correspondra à abcd, abbcd, abbbcd, etc.

```
app.get("/ab+cd", (req, res) => {
  res.send("ab+cd");
});
```

- Ce chemin d'accès correspondra à abcd, abxcd, abRANDOMcd, ab123cd, etc.

```
app.get("/ab*cd", (req, res) => {
  res.send("ab*cd");
});
```

Routes basées sur les Regexp

- Match n'importe quelle route contenant le caractère 'a'

```
app.get(/a/, (req, res) => {
  res.send("/a/");
});
```

- Match toutes les routes finissant par fly

```
app.get(/.*fly$/, (req, res) => {
  res.send("/.*fly$/");
});
```

Utiliser des paramètres dans les routes

Les paramètres sont des segments d'URL nommés pour capturer des valeurs spécifiées à leur position dans l'URL.
Les valeurs récupérées sont allouées à l'objet `req.params` avec le nom du paramètre comme clé.

```
Route: /users/:userId/books/:bookId
Requête d'URL: http://localhost:3000/users/34/books/8989
req.params: { "userId": "34", "bookId": "8989" }
```

```
app.get("/users/:userId/books/:bookId", (req, res) => {
  res.send(req.params);
});
```

Multiples callbacks de route

Plusieurs callback peuvent être associées à une même route, pour cela il est nécessaire de passer l'object `next` en paramètre.

```
const cb0 = function (req, res, next) {
  console.log("CB0");
  next();
};

const cb1 = function (req, res) {
  res.send("Hello from C!");
};

app.get("/example/c", [cb0, cb1]);
```

Les méthodes de l'objet réponse

Method	Description
res.download()	Envoie un fichier à télécharger
res.end()	Termine le processus de réponse
res.json()	Envoie une réponse en JSON
res.jsonp()	Envoie une réponse qui supporte le JSONP
res.redirect()	Redirige une requête
res.render()	Génère une vue
res.send()	Envoie une réponse sous différentes formes
res.sendFile()	Envoie un fichier sous forme de flux d'octets
res.sendStatus()	Défini un code http et l'envoie sous forme de chaîne dans le corps de la réponse

app.route()

Il est possible d'enchaîner les gestionnaires de routes avec la méthode app.route().

```
app
  .route("/book")
  .get((req, res) => {
    res.send("Get a random book");
  })
  .post((req, res) => {
    res.send("Add a book");
  })
  .put((req, res) => {
    res.send("Update the book");
  });
```

Définir un routeur

- Il est possible de séparer les différentes routes de notre application à l'aide d'un routeur.
- Cela permet de simplifier la maintenabilité de l'application tout en rendant le code plus modulaire
- Un routeur peut être instancié à l'aide de la classe `express.Router()`

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

Utiliser un routeur

- Définition du routeur

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

router.get("/", (req, res) => {
  res.send("Birds home page");
});

module.exports = router;
```

- Chargement du routeur dans l'application principale

```
const birds = require("./birds.js");
app.use("/birds", birds);
```

UTOPIOS
TRANSFORMATION DIGITALE INTELLIGENTE

Les middlewares

Définition d'un middleware

Les fonctions de **middleware** sont des fonctions qui peuvent accéder à l'objet request, l'objet response et à la fonction middleware suivant dans le cycle demande-réponse de l'application.
Elles servent à :

- Exécuter tout type de code
- Apporter des modifications aux objets de demande et de réponse
- Terminer le cycle de demande-réponse
- Appeler le middleware suivant dans la pile

Utilisation d'un middleware

- Si la fonction middleware en cours ne termine pas le cycle de demande-réponse, elle doit appeler la fonction `next()` pour transmettre le contrôle à la fonction middleware suivant. Sinon, la demande restera bloquée ⚠️.

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

app.get("/", (req, res, next) => {
  // code du middleware
  next(); // fonction à appeler pour paler au middleware suivant
});
```

Ordre des middlewares

- L'ordre de chargement des middleware est important : **les fonctions middleware chargées en premier sont également exécutées en premier**
- Puisque vous avez accès à l'objet **Request**, à l'objet **Response**, à la fonction middleware suivant dans la pile et à l'API Node.js complète, le champ des possibles avec les fonctions middleware est infini

Les types de middleware

Une application Express peut utiliser les types de middleware suivants :

- Middleware niveau application
- Middleware niveau routeur
- Middleware de traitement d’erreurs
- Middleware intégré
- Middleware tiers

Middleware niveau application

- Les middles middleware d'application se greffe au niveau de l'instance d'express.
- On peut utiliser les fonctions `app.use()` ou `app.METHOD()`

Sur tous les endpoints

```
const app = express();

app.use((req, res, next) => {
  console.log("Time:", Date.now());
  next();
});
```

Sur un endpoint particulier

```
app.use("/user/:id", (req, res, next) => {
  console.log("Request Type:", req.method);
  next();
});
```

Middleware niveau routeur

Le middleware niveau routeur fonctionne de la même manière que le middleware niveau application, à l'exception près qu'il est lié à une instance de express.Router()

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

router.use((req, res, next) => {
  console.log("Time:", Date.now());
  next();
});
```

Middleware traitement d'erreurs

Les middleware de traitement d’erreurs fonctionnent de la même façon que d’autres fonctions middleware, à l’exception près qu’il faudra 4 arguments au lieu de 3, et plus particulièrement avec la signature (err, req, res, next) :

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

Middlewares intégrés

Depuis la version 4.x de Express la plupart des middlewares ne sont plus intégrés à express.

- **body-parser** : Analyse les corps des requêtes entrantes dans un middleware avant les handlers, disponible sous la propriété `req.body`
- **errorhandler** : middleware pour l'analyse détaillé des erreurs en développement
- **morgan** : middleware de logging
- **serve-static** : permet de gérer plusieurs répertoires de fichiers statiques
- **express.json()** : transforme les données au format json en objet javascript via `req.body`

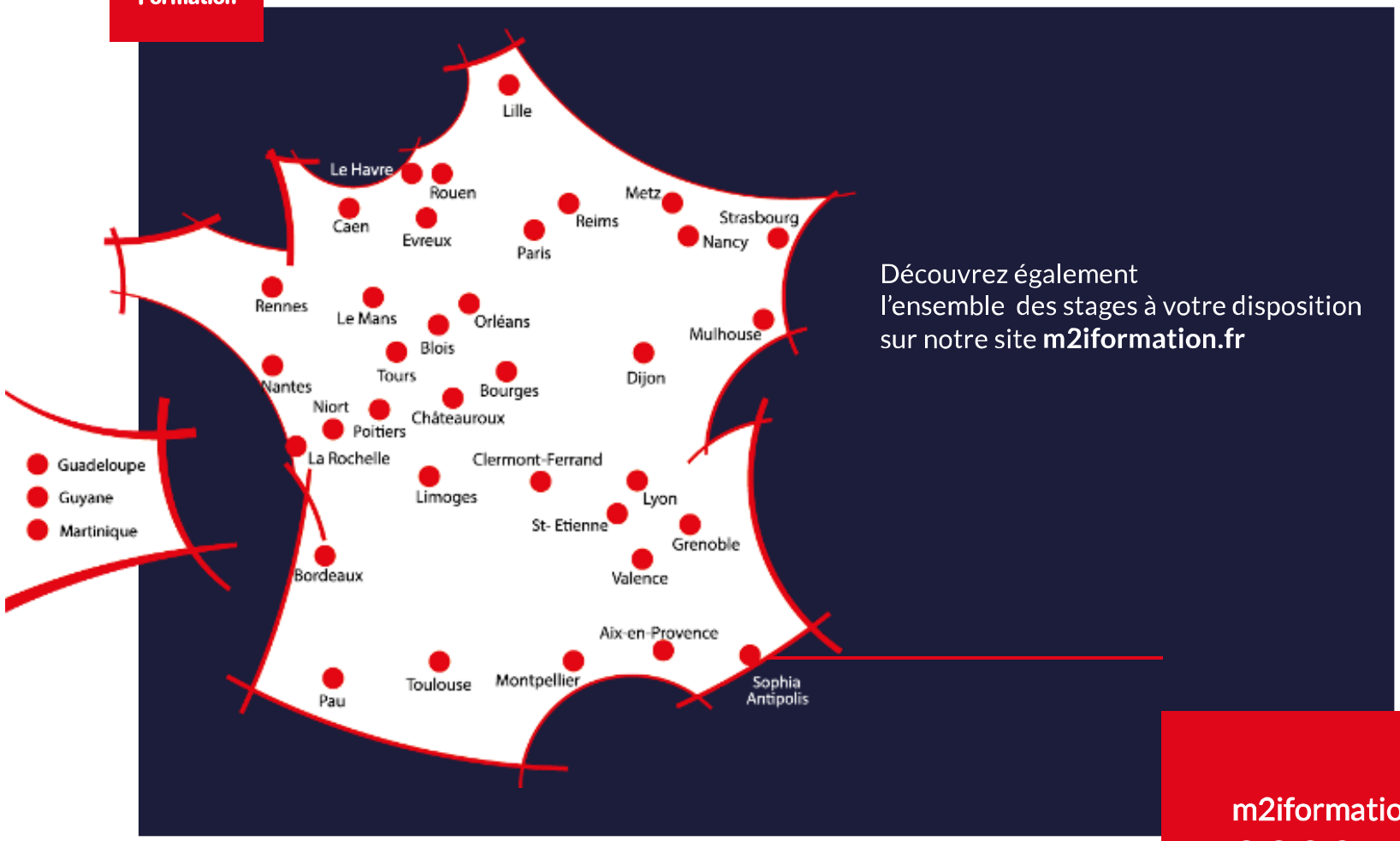
Utilisation d'un middleware intégré

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

// Ajouter le middleware express.json()
app.use(express.json());

// Définir une route qui reçoit une requête POST avec un corps JSON
app.post("/api/data", (req, res) => {
  // Accéder au corps de la requête comme un objet JavaScript
  console.log(req.body);
  res.send("Données reçues");
});
```

UTOPIOS
TRANSFORMATION DIGITALE INTELLIGENTE

Merci pour votre attention

Des questions ?



Découvrez également
l'ensemble des stages à votre disposition
sur notre site **m2iformation.fr**

m2iformation.fr