eBook Gratuit

APPRENEZ express

eBook gratuit non affilié créé à partir des **contributeurs de Stack Overflow.**

#express

Table des matières

À propos

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: express

It is an unofficial and free express ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official express.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Chapitre 1: Commencer avec express

Remarques

Express.js est, selon les développeurs, un "framework web rapide, sans autorisation et minimaliste pour Node.js."

Conçu pour être minimal et flexible, Express offre un ensemble de fonctionnalités pour la création d'applications Web et mobiles. Des méthodes HTTP au middleware intégré, Express est conçu pour vous fournir les fonctionnalités dont vous avez besoin pour créer une application Web ou mobile sur Node.js.

Si vous souhaitez créer une application sur Node.js Express est un excellent choix, que vous utilisiez vanilla Express ou l'un des nombreux frameworks basés sur Express ou construits sur Express. Certains de ces cadres peuvent être trouvés ici .

Versions

Versions d' **ici** .

Version	Remarques	Date de sortie
4.15.3		2017-05-16
4.15.2		2017-03-06
4.15.1		2017-03-05
4.15.0		2017-03-01
4.14.1		2017-01-28
4.14.0		2016-06-16
4.13.4		2016-01-21
4.13.3.		2015-08-02
4.13.2		2015-07-31
4.13.1		2015-07-05
4.13.0		2015-06-20
4.12.4		2015-05-17
4.12.3		2015-03-17

Version	Remarques	Date de sortie
4.12.2		2015-03-02
4.12.1		2015-03-01
4.12.0		2015-02-23
4.11.2		2015-01-20
4.11.1		2015-01-20
4.11.0		2015-01-13
4.10.8		2015-01-13
4.10.7		2015-01-04
4.10.6		2014-12-12
4.10.5		2014-12-10
4.10.4		2014-11-24
4.10.3		2014-11-23
4.10.2		2014-11-09
4.10,1		2014-10-28
4.10.0		2014-10-23
4.9.8		2014-10-17
4.9.7		2014-10-10
4.9.6		2014-10-08
4.9.5		2014-09-24
4.9.4		2014-09-19
4.9.3		2014-09-18
4.9.2		2014-09-17
4.9.1		2014-09-16
4.9.0		2014-09-08
4.8.8		2014-09-04

Version	Remarques	Date de sortie
4.8.7		2014-08-29
4.8.6		2014-08-27
4.8.5		2014-08-18
4.8.4		2014-08-14
4.8.3		2014-08-10
4.8.2		2014-08-07
4.8.1		2014-08-06
4.8.0		2014-08-05
4.7.4		2014-08-04
4.7.3		2014-08-04
4.7.2		2014-07-27
4.7.1		2014-07-26
4.7.0		2014-07-25
4.6.1		2014-07-12
4.6.0		2014-07-11
4.5.1		2014-07-06
4.5.0		2014-07-04
4.4.5		2014-06-26
4.4.4		2014-06-20
4.4.3		2014-06-11
4.4.2		2014-06-09
4.4.1		2014-06-02
4.4.0		2014-05-30
4.3.2		2014-05-28
4.3.1		2014-05-23

Version	Remarques	Date de sortie
4.3.0		2014-05-21
4.2.0		2014-05-11
4.1.2		2014-05-08
4.1.1		2014-04-27
4.1.0		2014-04-24
4.0.0		2014-04-09
3.21.2	De là à	2015-07-31
3.21.1	3.18.6 dates	2015-07-05
3.21.0	semble mal	2015-06-18
3.20.3		2015-05-17
3.20.2		2015-03-16
3.20.1		2015-02-28
3.20.0		2015-02-18
3.19.2		2015-02-01
3.19.1		2015-01-20
3.19.0		2015-01-09
3.18.6		2014-12-12
3.18.5		2014-12-11
3.18.4		2014-11-23
3.18.3		2014-11-09
3.18.2		2014-10-28
3.18.1		2014-10-22
3.18.0		2014-10-17
3.17.8		2014-10-15
3.17.7		2014-10-08

Version	Remarques	Date de sortie
3.17.6		2014-10-02
3.17.5		2014-09-24
3.17.4		2014-09-19
3.17.3		2014-09-18
3.17.2		2014-09-15
3.17.1		2014-09-08
3.17.0		2014-09-08
3.16.10		2014-09-04
3.16.9		2014-08-29
3.16.8		2014-08-27
3.16.7		2014-08-18
3.16.6		2014-08-14
3,16.5		2014-08-11
3.16.4		2014-08-10
3.16.3		2014-08-07
3.16.2		2014-08-07
3.16.1		2014-08-06
3.16.0		2014-08-05
3.15.3		2014-08-04
3.15.2		2014-07-27
3.15.1		2014-07-26
3.15.0		2014-07-22
3.14.0		2014-07-11
3.13.0		2014-07-03
3.12.1		2014-06-26

Version	Remarques	Date de sortie
3.12.0		2014-06-21
3.11.0		2014-06-19
3.10.5		2014-06-11
3.10.4		2014-06-09
3.10.3		2014-06-05
3.10.2		2014-06-03
3.10.1		2014-06-03
3.10.0		2014-06-02
3.9.0		2014-05-30
3.8.1		2014-05-27
3.8.0		2014-05-21
3.7.0		2014-05-18
3.6.0		2014-05-09
3.5.3		2014-05-08
3.5.2		2014-04-24
3.5.1		2014-03-25
3.5.0		2014-03-06
3.4.8		2014-01-13
3.4.7		2013-12-10
3.4.6		2013-12-01
3.4.5		2013-11-27
3.4.4		2013-10-29
3.4.3		2013-10-23
3.4.2		2013-10-18
3.4.1		2013-10-15

Version	Remarques	Date de sortie
3.4.0		2013-09-07
3.3.8		2013-09-02
3.3.7		2013-08-28
3.3.6		2013-08-27
3.3.4		2013-07-08
3.3.3		2013-07-04
3.3.2		2013-07-03
3.3.1		2013-06-27
3.3.0		2013-06-27
3.2.6		2013-06-02
3.2.5		2013-05-21
3.2.4		2013-05-09
3.2.3		2013-05-07
3.2.2		2013-05-03
3.2.1		2013-04-29
3.2.0		2013-04-15
3.1.2		2013-04-12
3.1.1		2013-04-01
3.1.0		2013-01-25
3.0.6		2013-01-04
3.0.5		2012-12-19
3,0.4		2012-12-05
3.0.3		2012-11-13
3.0.2		2012-11-08
3.0.1		2012-11-01

Version	Remarques	Date de sortie
3.0.0		2012-10-23
3.0.0rc5		2012-09-18
3.0.0rc4		2012-08-30
3.0.0rc3		2012-08-13
3.0.0rc2		2012-08-03
3.0.0rc1		2012-07-24
3.0.0beta7		2012-07-16
3.0.0beta6		2012-07-13
3.0.0beta5		2012-07-03
3.0.0beta4		2012-06-25
3.0.0beta3		2012-06-15
3.0.0beta2		2012-06-06
3.0.0beta1		2012-06-01
3.0.0alpha5		2012-05-30
3.0.0alpha4		2012-05-09
3.0.0alpha3		2012-05-04
3.0.0alpha2		2012-04-26
3.0.0alpha1		2012-04-15
2.5.9		2012-04-02
2.5.8		2012-02-08
2.5.7		2012-02-06
2.5.6		2012-01-13
2.5.5		2012-01-08
2.5.4		2012-01-02
2.5.3		2011-12-30

Version	Remarques	Date de sortie
2.5.2		2011-12-10
2.5.1		2011-11-17
2.5.0		2011-10-24
2.4.7		2011-10-05
2.4.6		2011-08-22
2.4.5		2011-08-19
2.4.4		2011-08-05
2.4.3		2011-07-14
2.4.2		2011-07-06
2.4.1		2011-07-06
2.4.0		2011-06-28
2.3.12		2011-06-22
2.3.11		2011-06-04
2.3.10		2011-05-27
2.3.9		2011-05-25
2.3.8		2011-05-24
2.3.7		2011-05-23
2.3.6		2011-05-20
2.3.5		2011-05-20
2.3.4		2011-05-08
2.3.3		2011-05-03
2.3.2		2011-04-27
2.3.1		2011-04-26
2.3.0		2011-04-25
2.2.2		2011-04-12

Version	Remarques	Date de sortie
2.2.1		2011-04-04
2.2.0		2011-03-30
2.1.1		2011-03-29
2.1.0		2011-03-24
2.0.0		2011-03-17
2.0.0rc3		2011-03-17
2.0.0rc2		2011-03-17
2.0.0rc		2011-03-14
2.0.0beta3		2011-03-09
2.0.0beta2		2011-03-07
2.0.0beta		2011-03-03
1.0.8		2011-03-01
1.0.7		2011-02-07
1.0.6		2011-02-07
1.0.5		2011-02-05
1.0.4		2011-02-05
1.0.3		2011-01-13
1.0.2		2011-01-10
1.0.1		2010-12-29
1.0.0		2010-11-16
1.0.0rc4		2010-10-14
1.0.0rc3		2010-09-20
1.0.0rc2		2010-08-17
1.0.0rc		2010-07-28
1.0.0beta2		2010-07-23

Version	Remarques	Date de sortie
1.0.0beta		2010-07-15
0,14,0		2010-06-15
0.13.0		2010-06-01
0.12.0		2010-05-22
0.11.0		2010-05-06
0,10,1		2010-05-03
0.10.0		2010-04-30
0.9.0		2010-04-14
0.8.0		2010-03-19
0.7.6		2010-03-19
0.7.5		2010-03-16
0.7.4		2010-03-16
0.7.3		2010-03-16
0.7.2		2010-03-16
0.7.1		2010-03-16
0.7.0		2010-03-15
0.6.0		2010-03-11
0.5.0		2010-03-10
0.4.0		2010-02-11
0.3.0		2010-02-11
0.2.1		2010-02-05
0.2.0		2010-02-03
0.1.0		2010-02-03
0.0.2		2010-01-10
0.0.1	Libération Intiale	2010-01-03

Examples

Installation

Express JS est le framework goto pour développer `Web Applications` , des `APIs` et presque tout type de `Backend` utilisant Node.

Pour installer `express` , il suffit de lancer la commande **npm**

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

Et tu as fini.

Créer et exécuter un nouveau serveur express

créer un fichier `app.js` et ajouter ce code

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

// when "/" is opened in url, this function will be called.
app.get('/', function (req, res) {
  res.json({ code: 200, message: 'success' });
})

app.listen( 3000, function () {
  console.log('Express server running at http://localhost:3000');
});
```

- Dans votre terminal, exécutez le `node app.js` et
- Ouvrez l'URL `http://localhost:3000` dans le navigateur Web pour voir votre serveur express nouvellement créé.

C'est également une bonne idée d'installer `body-parser` et `express-session` avec `express` car la plupart du temps, vous voudrez lire les données envoyées dans les requêtes `POST` et gérer les sessions utilisateur.

- analyseur de corps sur github
- session express sur github

Hello World App, en utilisant ExpressJS 4 et Node> = 4

Préface

Vous aurez besoin de `node >= 4` et d' `express 4` pour ce projet. Vous pouvez obtenir la dernière distribution de `node` depuis leur page de téléchargement .

Avant ce tutoriel, vous devez initialiser votre projet de noeud en exécutant

```
$ npm init
```

à partir de la ligne de commande et en remplissant les informations souhaitées. Notez que vous pouvez modifier les informations à tout moment en modifiant le fichier `package.json` .

Installation

Installez `express` avec `npm` :

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

Après avoir installé Express en tant que module de noeud, nous pouvons créer notre point d'entrée. Cela devrait être dans le même répertoire que notre `package.json`

```
$ touch app.js
```

Contenu du répertoire

Le dossier doit avoir la structure de répertoires suivante:

```
<project_root>
 |-> app.js
 |-> node_modules/
 '-> package.json
```

Code

Ouvrez `app.js` dans votre éditeur préféré et suivez ces quatre étapes pour créer votre première application Express:

```
// 1. Import the express library.
import express from 'express';

// 2. Create an Express instance.
const app = express();

// 3. Map a route. Let's map it to "/", so we can visit "[server]/".
app.get('/', function(req, res) {
   res.send('Hello World');
});

// 4. Listen on port 8080
app.listen(8080, function() {
   console.log('Server is running on port 8080...');
});
```

Exécution

A partir du répertoire du projet, nous pouvons exécuter notre serveur en utilisant la commande

```
$ node app.js
```

Vous devriez voir le texte

```
$ Our Express App Server is listening on 8080...
```

Maintenant, visitez `http://localhost:8080/` et vous verrez le texte "Hello World!"

Félicitations, vous avez créé votre première application Express!

Démarrer une application avec le générateur Express

Pour commencer rapidement avec Express, vous pouvez utiliser le générateur Express qui créera un squelette d'application pour vous.

Tout d'abord, installez-le globalement avec npm:

```
npm install express-generator -g
```

Vous devrez peut-être mettre `sudo` avant cette commande si vous obtenez une erreur "autorisation refusée".

Une fois le générateur installé, vous pouvez lancer un nouveau projet comme celui-ci:

```
express my_app
```

La commande ci-dessus créera un dossier appelé `my_app` avec un fichier `package.json` , un fichier `app.js` et quelques sous-dossiers tels que `bin` , `public` , `routes` , `views` .

Maintenant, naviguez jusqu'au dossier et installez les dépendances:

```
cd first_app
npm install
```

Si vous êtes sous Linux ou MacOS, vous pouvez lancer l'application comme ceci:

```
DEBUG=myapp:* npm start
```

Ou, si vous êtes sous Windows:

```
set DEBUG=myapp:* & npm start
```

Maintenant, chargez http: // localhost: 3000 / dans votre navigateur Web et vous devriez voir les mots "Welcome to Express".

Créer une application EJS

```
a@coolbox:~/workspace$ express --ejs my-app
a@coolbox:~/workspace$ cd my-app
a@coolbox:~/workspace/my-app$ npm install
a@coolbox:~/workspace/my-app$ npm start
```

Lire Commencer avec express en ligne: https://riptutorial.com/fr/express/topic/1616/commencer-avec-express

Chapitre 2: Comment fonctionne ExpressJs

Examples

Traitement des demandes / réponses

Le sucre syntaxique

La plupart des exemples de démarrage d'ExpressJs incluent ce morceau de code

```
var express = require('express');
var app = express();
...
app.listen(1337);
```

Eh bien, `app.listen` est juste un raccourci pour:

```
var express = require('express');
var app = express();
var http = require('http');
http.createServer(app).listen(1337);
```

L'App Express

Le célèbre `http.createServer` accepte une fonction connue sous le nom de gestionnaire. Le gestionnaire prend deux paramètres de **requête** et de **réponse en** tant qu'entrées, puis les manipule à l'intérieur de sa portée pour faire diverses choses.

Donc, fondamentalement, `app = express()` est une fonction qui se déroule en tant que gestionnaire et traite les requêtes et les réponses via un ensemble de composants spéciaux appelés middlewares.

Middlewares Stack

Un middleware de base est une fonction qui **demande** trois arguments, une **réponse** et la **suivante** .

Ensuite, par `app.use` , un middleware est monté sur la pile Express App Middlewares. La requête et la réponse sont manipulées dans chaque middleware puis redirigées vers la suivante via l'appel de `next()` .

Par exemple, le code ci-dessous:

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

```
var app = express();

app.use((request, response, next) => {
    request.propA = "blah blah";
    next();
});

app.use('/special-path', (request, response, next) => {
    request.propB = request.propA + " blah";
    if (request.propB === "blah blah blah")
        next();
    else
        response.end('invalid');
});

app.use((request, response, next) => {
    response.end(request.propB);
});

app.listen(1337);
```

Peut être traduit en gros à:

```
var http = require('http');
http.createServer((request, response) => {

    //Middleware 1
    if (isMatch(request.url, '*')) {
        request.propA = "blah blah";
    }

    //Middleware 2
    if (isMatch(request.url, "/special-path")) {
        request.propB = request.propA + " blah";
        if (request.propB !== "blah blah blah")
            return response.end('invalid');
    }

    //Middleware 3
    if (isMatch(request.url, "*")) {
        return response.end(request.propB);
    }
});

server.listen(1337);
```

Lire Comment fonctionne ExpressJs en ligne:
https://riptutorial.com/fr/express/topic/7815/comment-fonctionne-expressjs

Chapitre 3: Écrire un middleware express

Syntaxe

1. Spécifiez l'instance de express que vous souhaitez utiliser. Ceci est couramment *app* .
2. Définissez la méthode HTTP pour laquelle la fonction s'applique. Dans l'exemple, c'est *get* .
3. Définissez le chemin auquel la fonction s'applique. Dans l'exemple, c'est *'/'* .
4. Définir comme une fonction avec le mot-clé *function* .
5. Ajoutez les paramètres requis: req, res, next. (Voir note dans la section remarques)
6. Mettez du code dans la fonction pour faire ce que vous voulez

Paramètres

Paramètre	Détails
req	L'objet de requête.
res	L'objet de réponse
prochain	Le prochain appel de middleware ().

Remarques

Une fonction middleware est une fonction ayant accès à l'objet de requête (*req*), à l'objet réponse (*res*) et à la fonction middleware *next ()* dans le cycle demande-réponse de l'application. La fonction middleware *next ()* est généralement désignée par une variable nommée next.

Les fonctions de middleware sont conçues pour effectuer les tâches suivantes:

- Exécutez n'importe quel code.
- Apportez des modifications aux objets requête et réponse. (Voir l'exemple requestTime)
- Terminez le cycle de demande-réponse.
- Appelez le middleware suivant dans la pile. (En appelant le middleware *next*)

Note: Il n'est pas nécessaire de le nommer ensuite. Mais si vous utilisez autre chose, personne ne saura ce que vous voulez dire et vous serez renvoyé. Et votre code ne fonctionnera pas. Alors, nommez-le ensuite. Cette règle s'applique à l'objet demande et réponse. Certaines personnes utiliseront respectivement la requête et la réponse au lieu de req et res. C'est très bien. Cela gâche les frappes, mais ça va.

Examples

Logger Middleware

Si vous ne connaissez pas le middleware dans Express, consultez la section Présentation dans la section Remarques.

Tout d'abord, nous allons configurer une application simple Hello World qui sera référencée et ajoutée lors des exemples.

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

app.get('/', function(req, res) {
    res.send('Hello World!');
});

app.listen(3000);
```

Voici une fonction de middleware simple qui enregistrera "LOGGED" quand il sera appelé.

```
var myLogger = function (req,res,next) {
    console.log('LOGGED');
    next();
};
```

*L'appel de **next ()** appelle la fonction de middleware suivante dans l'application.*

Pour charger la fonction, appelez app.use () et spécifiez la fonction que vous souhaitez appeler. Cela se fait dans le bloc de code suivant qui est une extension du bloc Hello World.

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

var myLogger = function (req, res, next) {
    console.log('LOGGED');
    next();
};

app.use(myLogger);

app.get('/', function(req, res) {
    res.send('Hello World!');
});

app.listen(3000);
```

Maintenant, chaque fois que l'application reçoit une demande, elle imprime le message "LOGGED" au terminal. Alors, comment pouvons-nous ajouter des conditions plus spécifiques lorsque le middleware est appelé? Regardez l'exemple suivant et voyez.

requestTime Middleware

Créons un middleware qui ajoute une propriété appelée requestTime à l'objet request.

```
var requestTime = function (req, res, next) {
    req.requestTime = Date.now();
```

```
    next();
};
```

Modifions maintenant la fonction de journalisation de l'exemple précédent pour utiliser le middleware requestTime.

```
myLogger = function (req, res, next, requestTime) {
    console.log('LOGGED at ' + requestTime);
    next();
};
```

Ajoutons le middleware à notre application:

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

myLogger = function (req, res, next) {
    console.log('LOGGED at ' + req.requestTime);
    next();
};

var requestTime = function(req, res, next) {
    req.requestTime = Date.now();
    next();
};

app.use(requestTime);

app.use(myLogger);

app.get('/', function(req, res) {
    res.send('Hello World!');
});

app.listen(3000);
```

Maintenant, l'application enregistre l'heure à laquelle la demande a été faite. Cela couvre les bases de l'écriture et de l'utilisation du middleware Express. Pour plus d'informations, voir Utilisation d'Express Middleware .

!!! TODO: Créer en utilisant la section Express Middleware !!!

Middleware CORS

Cet exemple montre comment une requête http d'origine peut être traitée à l'aide d'un middleware.

Contexte CORS

CORS est une méthode de contrôle d'accès adoptée par tous les principaux navigateurs pour éviter les vulnérabilités inhérentes aux scripts croisés. Dans la sécurité générale du navigateur, les scripts doivent maintenir que toutes les demandes XHR doivent être effectuées uniquement à la source à partir de laquelle les mêmes scripts sont servis. Si une demande XHR est faite en dehors du domaine auquel appartiennent les scripts, la réponse sera rejetée.

Cependant, si le navigateur prend en charge CORS, il ferait une exception à cette règle si les en-têtes appropriés de la réponse indiquent que le domaine d'où provient la demande est autorisé. L'en-tête suivant indique que n'importe quel domaine est autorisé:

```
Access-Control-Allow-Origin: *
```

Exemple

L'exemple suivant montre comment le middleware Express peut inclure ces en-têtes dans sa réponse.

```
app.use(function(request, response, next){

    response.header('Access-Control-Allow-Origin', '*');
    response.header('Access-Control-Allow-Methods', 'GET,PUT,POST,DELETE,OPTIONS');
    response.header('Access-Control-Allow-Headers', 'Content-Type, Authorization, Content-
Length, X-Requested-With');

   //Handle Preflight
   if (reqest.method === 'OPTIONS') {
      response.status(200).send();
   }
   else {
      next();
   }

});
```

Gestion du contrôle en amont

La dernière partie de l'exemple ci-dessus gère le contrôle en amont. Le contrôle en amont est une requête OPTIONS spéciale que le navigateur envoie pour tester CORS si la demande contient des en-têtes personnalisés.

Références utiles

MDN - Tutoriel Http CORS

Lire Écrire un middleware express en ligne: https://riptutorial.com/fr/express/topic/6993/ecrire-un-middleware-express

Chapitre 4: en utilisant https avec express

Examples

Utiliser https avec express

D'abord, vous devez générer des clés publiques et privées en utilisant OpenSSL (tutoriel).

```
var express = require("express");
var http =require ("http");
var https=require ("https");
var fs=require("fs");
var app=express();
var httpsKeys={
key:fs.readFileSync("<key.pem>");
crtifcte:fs.readFileSync("<certificate.pem>");
};
http.createserver(app).listen(3000);
https.createserver(httpsKeys,app).listen(3030);
```

Lire en utilisant https avec express en ligne: https://riptutorial.com/fr/express/topic/7844/en-utilisant-https-avec-express

Chapitre 5: Enregistrement

Remarques

`morgan` est un middleware HTTP de journalisation de requêtes pour node.js

Examples

Installation

Tout d'abord, installez le `morgan` Middleware dans votre projet.

```
npm install --save morgan
```

Journalisation Express simple de toutes les demandes à STDOUT

Ajoutez le code suivant à votre fichier `app.js` :

```
var express = require('express')
var morgan = require('morgan')

var app = express()

app.use(morgan('combined'))

app.get('/', function (req, res) {
  res.send('hello, world!')
})
```

Maintenant, lorsque vous accédez à votre site Web, vous verrez dans la console que vous avez utilisée pour démarrer le serveur que les demandes sont consignées

Écrivez les journaux Express dans un seul fichier

D'abord, installez `fs` et le `path` dans votre projet

```
npm install --save fs path
```

Ajoutez le code suivant à votre fichier `app.js` :

```
var express = require('express')
var fs = require('fs')
var morgan = require('morgan')
var path = require('path')

var app = express()

// create a write stream (in append mode)
```

```
var accessLogStream = fs.createWriteStream(path.join(__dirname, 'access.log'), {flags: 'a'})

// setup the logger
app.use(morgan('combined', {stream: accessLogStream}))

app.get('/', function (req, res) {
  res.send('hello, world!')
})
```

Lorsque vous accédez à votre site Web, un fichier `access.log` été créé dans votre répertoire de projet.

Écrire des journaux Express dans un fichier journal en rotation

Tout d'abord, installez `fs` , `file-stream-rotator` et `path` dans votre projet

```
npm install --save fs file-stream-rotator path
```

Ajoutez le code suivant à votre fichier `app.js` :

```
var FileStreamRotator = require('file-stream-rotator')
var express = require('express')
var fs = require('fs')
var morgan = require('morgan')
var path = require('path')

var app = express()
var logDirectory = path.join(__dirname, 'log')

// ensure log directory exists
fs.existsSync(logDirectory) || fs.mkdirSync(logDirectory)

// create a rotating write stream
var accessLogStream = FileStreamRotator.getStream({
  date_format: 'YYYYMMDD',
  filename: path.join(logDirectory, 'access-%DATE%.log'),
  frequency: 'daily',
  verbose: false
})

// setup the logger
app.use(morgan('combined', {stream: accessLogStream}))

app.get('/', function (req, res) {
  res.send('hello, world!')
})
```

Maintenant, lorsque vous accédez à votre site Web, vous verrez qu'un répertoire de `log` été créé et un fichier journal avec un format de nom d' `access-%DATE%.log` été créé dans votre répertoire de journaux.

Lire Enregistrement en ligne: https://riptutorial.com/fr/express/topic/7191/enregistrement

Chapitre 6: Expliquer le routage dans Express

Examples

Routeur Express

Le routeur express vous permet de créer plusieurs "mini-applications" pour pouvoir nommer vos routes API, publiques, d'authentification et autres dans des systèmes de routage distincts.

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

router.get('/', function(req, res){
   res.send('Get request received');
});

router.post('/', function(req, res){
   res.send('Post requestreceived');
});

app.use('/', router);

app.listen(8080);
```

Gestionnaires de routage par chaîne pour un chemin d'accès à l'aide de app.route

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

app.route('/user')
  .get(function (req, res) {
    res.send('Get a random user')
  })
  .post(function (req, res) {
    res.send('Add a user')
  })
  .put(function (req, res) {
    res.send('Update the user details')
  })
  .delete(function (req, res) {
    res.send('Delete a user')
  });
```

Lire Expliquer le routage dans Express en ligne:
https://riptutorial.com/fr/express/topic/6536/expliquer-le-routage-dans-express

Chapitre 7: générateur express

Paramètres

Paramètre	Définition
-h, --help	informations d'utilisation de sortie
-V, --version	sortir le numéro de version
-e, --ejs	Ajout de la prise en charge du moteur de template pour pjs (JavaScript incorporé) (par défaut jade, renommé Pug)
--hbs	ajouter un support de modèle pour le guidon
-H, --hogan	ajouter le support moteur hogan.js
--git	ajouter .gitignore
-f, --force	forcer sur le répertoire non vide
-c <moteur>, --css <moteur>	ajout de la prise en charge de <feuille> de stylesheet (moins, stylet, boussole, sass) (par défaut css)

Remarques

Le générateur express est un excellent outil pour faire évoluer rapidement un projet. Une fois que vous avez compris l'organisation qu'elle implémente, vous économisez du temps.

Examples

Installation du générateur Express

```
npm --install express-generator -g
```

Créer une application

```
express my-app
```

Démarrer l'application

Utiliser l'option de démarrage

```
npm start
```

Utiliser Nodemon

```
nodemon
```

En utilisant pour toujours

```
forever start 'js file name'
```

Arrêter pour toujours

```
forever stop ''js file name'
```

Pour redémarrer à jamais

```
forever restart 'js filename'
```

Lister le serveur en train de ruiner en utilisant pour toujours

```
forever list
```

Lire générateur express en ligne: https://riptutorial.com/fr/express/topic/4512/generateur-express

Chapitre 8: Intégration de base de données express

Examples

Connectez-vous à MongoDB avec Node & Express

Tout d'abord, assurez-vous d'avoir installé *mongodb* et *express* via npm. Ensuite, dans un fichier intitulé classiquement *db.js* , utilisez le code suivant:

```
var MongoClient = require('mongodb').MongoClient

var state = {
    db: null,
}

exports.connect = function(url, done) {
    if (state.db) return done()

    MongoClient.connect(url, function(err, db) {
        if(err) return done(err)
        state.db = db
        done()
    })
}

exports.get = function() {
    return state.db
}

exports.close = function(done) {
    if (state.db) {
        state.db.close(function(err, result) {
            state.db = null;
            state.mode = null;
            done(err);
        })
    }
}
```

Ce fichier se connecte à la base de données et vous pouvez simplement utiliser l'objet **db** renvoyé par la méthode **get** .

Maintenant, vous devez inclure le fichier db en l'exigeant dans votre fichier app.js. En supposant que votre fichier *db.js se* trouve dans le même répertoire que *app.js,* vous pouvez insérer la ligne:

```
var db = require('./db');
```

Ceci, cependant, ne vous connecte pas réellement à votre instance MongoDB. Pour ce faire, insérez le code suivant avant d'appeler votre méthode app.listen. Dans notre exemple, nous intégrons le traitement des erreurs et la méthode app.listen dans la connexion à la base de

données. Veuillez noter que ce code ne fonctionne que si vous exécutez votre instance mongo sur la même machine que celle sur laquelle l'application Express est installée.

```
db.connect('mongodb://localhost:27017/databasename', function(err) {
    if (err) {
        console.log('Unable to connect to Mongo.');
        process.exit(1);
    } else {
        app.listen(3000, function() {
            console.log('Listening on port 3000...');
        });
    }
});
```

Votre application Express devrait maintenant être connectée à votre base de données Mongo. Félicitations!

Lire Intégration de base de données express en ligne:
https://riptutorial.com/fr/express/topic/7002/integration-de-base-de-donnees-express

Chapitre 9: La gestion des erreurs

Syntaxe

- `app.use(function(err, req, res, next) {})` // middleware de base

Paramètres

prénom	La description
`err`	Objet avec informations d'erreur
`req`	Objet de requête HTTP
`res`	Objet de réponse HTTP
`next`	fonction utilisée pour démarrer l'exécution du middleware suivant

Examples

Échantillon de base

Contrairement aux autres fonctions de middleware, les fonctions middleware de gestion des erreurs ont quatre arguments au lieu de trois: `(err, req, res, next)` .

Échantillon:

```
app.use(function(err, req, res, next) {
  console.error(err.stack);
  res.status(500).send('Error found!');
});
```

Lire La gestion des erreurs en ligne: https://riptutorial.com/fr/express/topic/2739/la-gestion-des-erreurs

Chapitre 10: Le routage

Examples

Routing Hello World

Le fichier d'application principal charge le fichier de routes où les routes sont définies.

app.js

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

app.use('/', require('./routes'));

app.listen('3000');
```

routes.js

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

router.get('/', function(req, res) {
    res.send('Hello World!');
});

module.exports = router;
```

Middleware de routage

Le middleware est exécuté avant l'exécution de la route et peut décider d'exécuter ou non le routeur en fonction de l'URL.

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

router.use(function (req, res, next) {
    var weekDay = new Date().getDay();
    if (weekDay === 0) {
        res.send('Web is closed on Sundays!');
    } else {
        next();
    }
})

router.get('/', function(req, res) {
    res.send('Sunday is closed!');
});

module.exports = router;
```

Un middleware spécifique peut également être envoyé à chaque gestionnaire de routeur.

```
var closedOnSundays = function (req, res, next) {
    var weekDay = new Date().getDay();
    if (weekDay === 0) {
        res.send('Web is closed on Sundays!');
    } else {
        next();
    }
}

router.get('/', closedOnSundays, function(req, res) {
    res.send('Web is open');
});

router.get('/open', function(req, res) {
    res.send('Open all days of the week!');
});
```

Routes multiples

Le fichier d'application principal charge tous les fichiers de routes dans lesquels vous souhaitez définir des itinéraires. Pour ce faire, nous avons besoin de la structure de répertoires suivante: routes app.js / routes index.js / users.js

app.js

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

app.use('/', require('./routes/index'));
app.use('/users', require('./routes/users'))

app.listen('3000');
```

routes / index.js

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

router.get('/', function(req, res) {
    res.send('Index Page');
});

router.get('/about', function(req, res) {
    res.send('About Page');
});

module.exports = router;
```

routes / users.js

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

router.get('/', function(req, res) {
    res.send('Users Index Page');
});
```

```
router.get('/list', function(req, res) {
    res.send('Users List Page');
});

module.exports = router;
```

Exécuter `$ node app.js` devrait maintenant `$ node app.js` pages aux URL suivantes:

- localhost: 3000 / - Affiche "Index Page"
- localhost: 3000 / about - Affiche "About Page"
- localhost: 3000 / users - Affiche la "page d'index des utilisateurs"
- localhost: 3000 / users / list - Affiche la "Liste des utilisateurs"

Lire Le routage en ligne: https://riptutorial.com/fr/express/topic/2589/le-routage

Chapitre 11: Manipulation de fichiers statiques

Syntaxe

1. Pour servir des fichiers statiques (images, fichiers CSS, fichiers JS, etc.), utilisez la fonction middleware **express.static** .
2. Transmettez le nom du répertoire qui contient les actifs à **express.static** pour servir directement les fichiers. (Regardez l' *exemple de base*)
3. Vous pouvez utiliser plusieurs répertoires, appelez simplement **express.static** plusieurs fois. N'oubliez pas qu'Express recherche les fichiers dans l'ordre dans **lequel** vous définissez les répertoires avec **express.static** . (Regardez l' *exemple des répertoires multiples*)
4. Vous pouvez créer un préfixe de chemin virtuel (c'est-à-dire un préfixe où le chemin n'existe pas dans le système de fichiers) avec **express.static** , spécifiez simplement un chemin de montage. (Regardez l' *exemple de préfixe de chemin virtuel*)
5. Tous les chemins précédents sont relatifs au répertoire à partir duquel vous lancez le processus de *noeud* . Donc, il est généralement plus sûr d'utiliser le chemin absolu du répertoire que vous voulez servir. (Rechercher dans le *chemin d'accès absolu au répertoire de fichiers statiques*)
6. Vous pouvez combiner les options de cette méthode, comme indiqué dans l'exemple de *chemin absolu vers le répertoire et le chemin d'accès virtuel*

Remarques

Tous les exemples peuvent être exécutés dans le noeud. Copiez et collez simplement dans un projet de nœud avec Express installé et exécutez-les avec le **nom de fichier du nœud** . Pour un exemple d'installation de Express, cliquez ici et assurez-vous que npm installed suit les instructions d'installation des paquets à installer "express".

Examples

Exemple de base

```
// Basic code for Express Instance
var express = require('express');
var app = express();

// Serve static files from directory 'public'
app.use(express.static('public'));

// Start Express server
app.listen(3030);
```

Exemple de répertoires multiples

```
// Set up Express
var express = require('express');
var app = express();

// Serve static assets from both 'public' and 'files' directory
app.use(express.static('public');
app.use(express.static('files');

// Start Express server
app.listen(3030);
```

Exemple de préfixe de chemin virtuel

```
// Set up Express
var express = require('express');
var app = express();

// Specify mount path, '/static', for the static directory
app.use('/static', express.static('public'));

// Start Express server
app.listen(3030);
```

Exemple de chemin absolu vers des fichiers statiques

```
// Set up Express
var express = require('express');
var app = express();

// Serve files from the absolute path of the directory
app.use(express.static(__dirname + '/public'));

// Start Express server
app.listen(3030);
```

Chemin d'accès absolu à l'exemple de préfixe de répertoire et de chemin virtuel

```
// Set up Express
var express = require('express');
var app = express();

/*  Serve from the absolute path of the directory that you want to serve with a
 */ virtual path prefix
app.use('/static', express.static(__dirname + '/public'));

// Start Express server
app.listen(3030);
```

Fichiers statiques de base et favicon servent d'exemple

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

```
var favicon = require('serve-favicon');

var app = express();

app.use(favicon(__dirname + '/public/img/favicon.ico'));
app.use(express.static(path.join(__dirname, 'public')));

app.listen(3000, function() {
  console.log("Express App listening on port 3000");
})
```

Lire Manipulation de fichiers statiques en ligne:
https://riptutorial.com/fr/express/topic/6954/manipulation-de-fichiers-statiques

Chapitre 12: Relier

Examples

Connect et Express

Express est basé sur Connect, qui fournit les fonctionnalités de middleware d'Express. Pour comprendre ce qu'est la connexion, vous pouvez voir qu'elle fournit la structure de base de l'application que vous utilisez lorsque vous utilisez express

```
const connect = require('connect')

const app = connect()
app.listen(3000)
```

Cela ouvrira un serveur http "vide" qui répondra 404 à toutes les demandes.

Middleware

Le middleware est attaché à l'objet app, généralement avant que listen soit appelé. Exemple de middleware de journalisation simple:

```
app.use(function (req, res, next) {
    console.log(`${req.method}: ${req.url}`)
    next()
})
```

Tout ce que cela va faire est de connecter `GET: /example` si vous voulez localiser `localhost:3000/example` . Toutes les demandes renverront toujours 404 car vous ne répondez pas avec des données.

Le prochain middleware de la chaîne sera exécuté dès que le précédent aura appelé `next()` . Nous pourrons ainsi répondre aux demandes en ajoutant un autre middleware comme celui-ci:

```
app.use(function (req, res, next) {
    res.end(`You requested ${req.url}`)
})
```

Maintenant, lorsque vous demandez "localhost: 3000 / example", `you will be greeted with "You requested /example". There is no need to call` ensuite` cette fois puisque ce middleware est le dernier de la chaîne (mais rien de grave n'arrivera si vous le faisiez),

Programme complet jusqu'ici:

```
const connect = require('connect')

const app = connect()
```

```
app.use(function (req, res, next) {
    console.log(`${req.method}: ${req.url}`)
    next()
})

app.use(function (req, res, next) {
    res.end(`You requested ${req.url}`)
    next()
})

app.listen(3000)
```

Erreurs et middleware d'erreur

Si nous souhaitons limiter l'accès à notre application, nous pourrions également écrire un middleware pour cela! Cet exemple ne vous autorise à accéder que les jours précédents, mais un exemple réel pourrait être, par exemple, *l'authentification des utilisateurs* . Un bon endroit pour mettre cela serait après le middleware de journalisation, mais avant tout contenu est envoyé.

```
app.use(function (req, res, next) {
    if (new Date().getDay() !== 4) {
        next('Access is only granted on thursdays')
    } else {
        next()
    }
})
```

Comme vous pouvez le voir dans cet exemple, envoyer une erreur est aussi simple que de fournir un paramètre à la fonction `next()` .

Maintenant, si nous visitons le site Web un jour différent de celui du jeudi, nous serions accueillis avec une erreur 500 et la chaîne `'Access is only granted on thursdays'` .

Maintenant, ce n'est pas suffisant pour notre site. Nous préférons envoyer à l'utilisateur un message HTML dans un autre middleware:

```
app.use(function (err, req, res, next) {
    res.end(`<h1>Error</h1><p>${err}</p>`)
})
```

Cela fonctionne un peu comme un bloc catch: toute erreur dans le middleware avant le middleware d'erreur sera envoyée à l'ancien. Un middleware d'erreur est identifié par ses 4 paramètres.

Vous pouvez également utiliser le middleware d'erreur pour récupérer l'erreur en appelant à nouveau la méthode suivante:

```
app.use(function (err, req, res, next) {
    // Just joking, everybody is allowed access to the website!
    next()
})
```

Lire Relier en ligne: https://riptutorial.com/fr/express/topic/4031/relier

Chapitre 13: Voir la configuration du moteur

Introduction

Souvent, le serveur doit servir des pages dynamiquement. Par exemple, un utilisateur Mr.X visite la page et voit quelque chose comme "Bienvenue Mr. X sur ma page d'accueil". Dans ce cas, les vues peuvent être utiles. être pratique Les variables peuvent être injectées de manière dynamique dans HTML à l'aide du moteur de vue. Le moteur de visualisation permet de rendre les vues. Une vue peut être affichée dans un dossier appelé voir et servir à la demande. méthode de résolution.

Remarques

installez ejs en utilisant ce qui suit (je sais que c'est évident)

```
sudo npm install ejs --save
```

Examples

1: mise en place des vues

```
var express=require("express");    //express is included
var path=require("path");    //path is included

var app=express();    //app is an Express type of application

app.set("views",path.resolve(__dirname,"views"));    //tells express about the location of the
views in views folder
app.set("view engine","ejs");    //tells express that ejs template engine is used
```

2. Exemple de fichier EJS (référez-vous à 1.set up ... before this)

Ce qui suit est un fichier ejs.

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Hello, world!</title>
    </head>
    <body>
        <%= message %>
    </body>
</html>
```

3.rendering view with express (référez-vous au fichier 2.EJS ... avant cela)

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

```
response.render("index",{                   //render the index when root(/) is requested
    message:"rendered view with ejs"
        });
});
```

4.après le rendu HTML final est créé (voir 3.rendering ... avant cela)

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <title>Hello, world!</title>
</head>
<body>
    message:"rendered view with ejs"
</body>
</html>
```

Lire Voir la configuration du moteur en ligne: https://riptutorial.com/fr/express/topic/8104/voir-la-configuration-du-moteur

Crédits

S. No	Chapitres	Contributeurs
1	Commencer avec express	Akshay Khale, Community, David Vogel, Dima Grossman, dkimot, Everettss, Gregory Worrall, Guillaume Lrv, Jared Hooper, jawadhoot, Kilmazing, Random User, Sumner Evans, user6939352
2	Comment fonctionne ExpressJs	rocketspacer
3	Écrire un middleware express	Charlie H, dkimot
4	en utilisant https avec express	nilakantha singh deo
5	Enregistrement	Mor Paz
6	Expliquer le routage dans Express	Dima Grossman, Sujithrao
7	générateur express	rickrizzo, Rupali Pemare
8	Intégration de base de données express	dkimot
9	La gestion des erreurs	gevorg, jawadhoot, Kilmazing
10	Le routage	jawadhoot, Kelvin, phobos, S.L. Barth, zurfyx
11	Manipulation de fichiers statiques	dkimot, Sujithrao
12	Relier	Henrik Karlsson, Overflowh
13	Voir la configuration du moteur	Daniele Giussani, nilakantha singh deo