303 - ExpressJS

node.js

WIK-NJS303

Durée estimée : 4h (hors TP final)
Intervenant : Jeremy Trufier <jeremy@wikodit.fr>

ynov

wikodit

WIK-NJS

Programme nodeJS

301 – Introduction
302 – Scripting et CLI
303 – Express.js
304 – MVC Frameworks
(305 – Tests unitaires)

1XX – 1er année (pas de notion d'algorithmie)
2XX – 2e année (notions d'algorithmie succintes)
3XX – 3e année (rappels et pratique, niveau moyen d'algorithmie)
4XX – 4e année (concepts avancés, niveau avancé d'algorithmie)
5XX – 5e année (approfondissement experts)

Au préalable

Un framework

- Serveur Web
- Minimaliste
- Performant
- Basé sur un système de middleware

Les différences

Sans express.js

```
const http = require('http')
const PORT = process.env.PORT || 8080

http.createServer((req, res) => {
  res.end('Bonjour à tous !')
}).listen(PORT, () => {
  console.log('Serveur sur port ', PORT)
})
```

Avec express.js

```
const express = require('express')
const app = express()
const PORT = process.env.PORT || 8080

app.get('/', (req, res) => {
  res.send('Bonjour à tous')
})

app.listen(PORT, () => {
  console.log('Serveur sur port : ', PORT)
})
```

En détail

Le routing

- Chaînage
- Routing par verbe http
- Route path
  - Simple
  - Regexp
  - Named parameters

```
app.all('*', (req, res, next) => {
  console.log('-> ALL *')
  next()
})

app.get('/', (req, res, next) => {
  console.log('-> GET /')
  res.send('GET /')
})

app.post('/', (req, res, next) => {
  console.log('-> POST /')
  res.send('POST /')
})

app.get('/users', (req, res, next) => {
  console.log('-> GET /users (liste d\'utilisateurs)')
  res.send('GET /users')
})

app.get('/users/:userId', (req, res, next) => {
  console.log('-> GET /users/:userId (userId : ' + req.params.userId +')')
  res.send('GET /users/:userId')
})

app.get(/^\/images\/.*\.(png, jpg, gif)$/, (req, res, next) => {
  console.log('-> GET /images/* (une image png, jpg, git)')
  res.send('GET /images/*')
})
```

Le routing (2)

```
function a(req, res, next){
  console.log('a')
  next()
}

function b(req, res, next){
  console.log('b')
  setTimeout(() => { // On passe au next() seulement après 5 secondes
    next()
  }, 5000)
}

app.get('/test', [a, b], (req, res, next) => {
  console.log('c')
  next()
}, (req, res, next) => {
  console.log('d')
  res.send('On est passé par a, b, c, d, avec 5 secondes d\'attentes avant c')
})
```

L'objet Request

- Contient les informations de la requête

Certain middlewares permettent de rajouter des informations sur l'objet request.

De plus, il est possible de rajouter des attributs sur l'objet `req` pour passer des infos entre les différentes méthodes du routing.

```
app.get('/:param1', (req, res) => {
  console.log(
    'Prot: ', req.protocol,
    ', Url: ', req.url,
    ', Method: ', req.method,
    ', Param1: ', req.params.param1,
    ', Query: ', req.query,
    ', Header Content-Type: ', req.get('Content-Type')
  )

  res.status(200).end()
})
```

L'objet Response

- Permet d'envoyer des données au client

```
// Réponse simple
res.send('Bonjour !')
```

```
// Set le statut, et simple retour
res.status(404)
res.end('Not Found')
```

```
// Une simple redirection
res.redirect(301, 'http://google.fr')
```

```
// Un retour différent selon le format demandé
res.format({
  html: () => { res.send('<p>Bonjour !</p>') }
  json: () => { res.send({ message: 'Bonjour !' }) }
})
```

```
// On set/get un header de la réponse. Attention une fois des données envoyé au client, les
// headers ne peuvent plusêtre modifié, cela peut résulter en une erreur.
res.set('ETag', '17b3cc1a-8eeb-11e6-ae22-56b6b6499611')
console.log(res.get('ETag'))
```

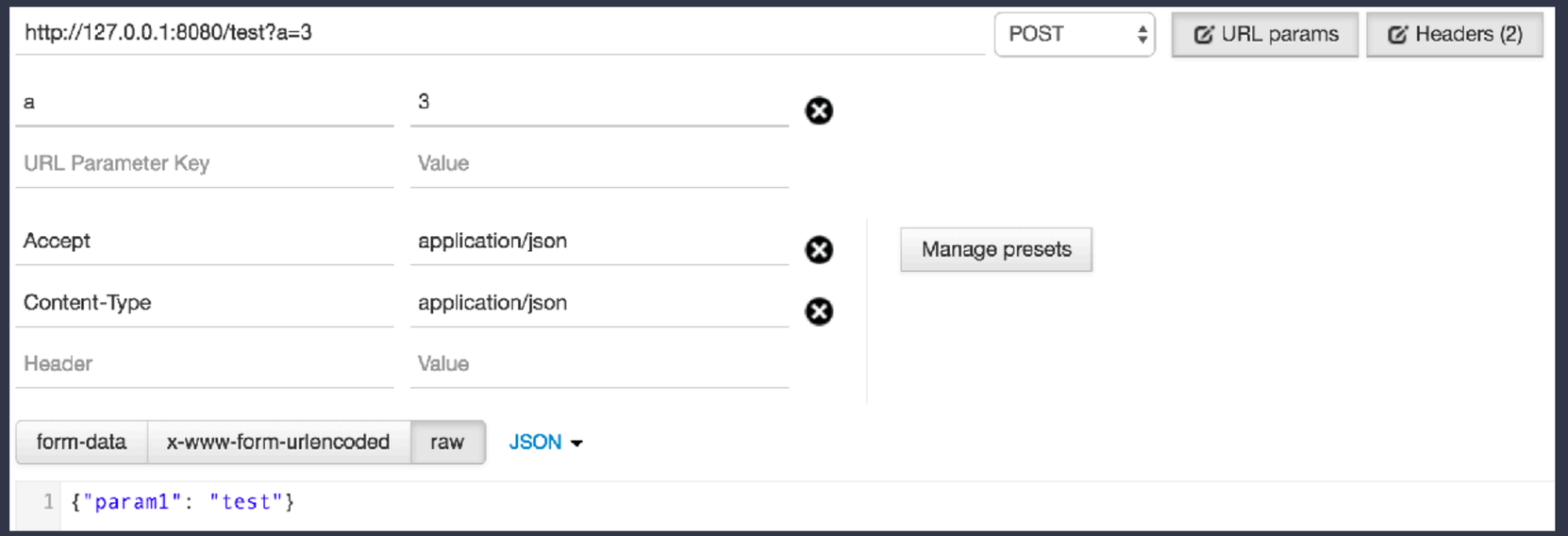
Comment tester ?

Dans le terminal

```
curl -i -H 'Accept: text/html' "http://localhost:8080/test?a=3"
```

```
curl -X POST -H 'Content-Type: application/json' -H 'Accept: text/html' -d
'{"param1": "test"}' "http://localhost:8080/test?a=3"
```

Avec Postman
(dans chrome)



Les middlewares

C'est quoi ?

- .get, .post, ... sont des middlewares
- Les plugins expressJS sont des middlewares
- Interception de requètes avant ou après la logique
- La requête client passe par un ou plusieurs middleware
- On passe au middleware suivant avec `next()`
- Un middleware peut terminer la réponse avec un `res.end` ou `res.send`

Exemple

```
var logger = function(req, res, next) {
  next()
  console.log(`REQUEST: ${req.method} ${req.url}`)
}

// Middleware qui log les requêtes
app.use(logger)

// Notre routing
app.get('/', (req, res, next) => {
  res.send('Page d\'accueil')
})

// autre routes
// ...

// Dernier middleware
app.use((req, res) => {
  res.status(404)
  res.end('Not Found')
})
```

Static middleware

```
app.use(express.static('assets'))
```

njs-303/assets/img/image.jpg sera accessible depuis http://localhost:8080/img/image.jpg

```
app.use('/styles', express.static('assets/css'))
```

njs-303/assets/css/app.css sera accessible depuis http://localhost:8080/styles/app.css

Body middleware

Attention, c'est un module NPM à installer !!

Pour parser les Content-Type: application/json =>

Pour parser les formulaire HTML =>

Nouvelle propriété req.body =>

```
const bodyParser = require('body-parser')

app.use(bodyParser.json())
app.use(bodyParser.urlencoded({
  extended: true
}))

app.post('/test', (req, res, next) => {
  console.log(req.body)
  res.send('On a parsé le body !')
})
```

TD1 : API Rest avec expressJS

1. Ouvrir une base de donnée SQLite et créer une table :
   - users (pseudo, email, firstname, lastname, createdAt, updatedAt)
2. Créer 4 routes
   - ALL / *=> "Bienvenue sur notre superbe API"*
   - POST /users *=> Ajouter un utilisateur*
   - GET /users/:userId *=> Récupérer un utilisateur*
   - GET /users?limit=20&offset=0 *=> Lister tous les utilisateurs*
   - DELETE /users/:userId *=> Supprimer un utilisateur*
   - PUT /users/:userId *=> Éditer un utilisateur*
   - autre *=> 501 Not Implemented*
3. Implémenter chaque route dans l'ordre çi-dessus, vérifier le bon fonctionnement à chaque fois. Les routes doivent répondre au format JSON.
4. Bonus : Utiliser la structure de réponse standardisée de http://jsonapi.org/
5. Bonus : Ajouter des filtres sur "/users" pour rechercher par prénom, nom , ... et un paramètre de tri : "order=xxxx&reverse=1"

Les templates

Les vues

- Nécessité de séparer les vues de la logique
- Différents moteurs de templates
  - EJS
  - Pug (anciennement Jade)
  - Handlebars
  - ... plusieurs dizaines (centaines?) d'autres !

Pug

```
$ npm install pug --save
```

index.js

```
app.set('view', './views')
app.set('view engine', 'pug')

app.get('/', () => {
  res.render('main', {
    title: 'Bonjour !',
    name: 'Toto',
    content: 'Ma première page'
  })
})
```

views/main.pug

```
html
  head
    title= title
  body
    h1 Bonjour #{name}
    p#main-paragraphe.center= content
```

EJS

```
$ npm install ejs --save
```

index.js

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

app.get('/', () => {
  res.render('main', {
    title: 'Bonjour !',
    name: 'Toto',
    content: 'Ma première page'
  })
})
```

views/main.ejs

```
<!DOCTYPE html>
<html>
<head>
    <title><%= title %></title>
</head>
<body class="container">
  <h1>Bonjour <%= name %></h1>
  <p id="main-paragraphe" class="center">
    <%= content %>
  </p>
</body>
</html>
```

Handlebars

```
$ npm install hbs --save
```

index.js

```
app.set('view', './views')
app.set('view engine', 'hbs')

app.get('/', () => {
  res.render('main', {
    title: 'Bonjour !',
    name: 'Toto',
    content: 'Ma première page'
  })
})
```

views/main.ejs

```
<!DOCTYPE html>
<html>
<head>
    <title>{{title}}</title>
</head>
<body class="container">
  <h1>Bonjour {{name}}</h1>
  <p id="main-paragraphe" class="center">
    {{content}}
  </p>
</body>
</html>
```

REST API & Convention

Routes standards sur une API REST
pour une resource Users :

- **GET** /users => views/users/index.pug
- **GET** /users/add => views/users/edit.pug (dans de rare cas : add.pug)
- **GET** /users/:userId => views/users/show.pug
- **GET** /users/:userId/edit => views/users/edit.pug
- **POST** /users => HTML : redirection /users ; JSON : status succès
- **UPDATE** /users/:userId => HTML : redirection /users ; JSON : status succès
- **DELETE** /users/:userId => HTML : redirection /users ; JSON : status succès

TD2 : Rajout de vues

1. Reprenez le TD 1
2. Ajouter la gestion du multi-format de réponse
3. Implémenter toutes les méthodes REST pour la resource 'users'
4. Chaque appel peut désormais retourner soit une page HTML dans le navigateur, aves des formulaires, tableaux, etc... Soit une structure JSON si le header 'Accept' est à 'application/json'

Namespace des routes et structure

Etude d'une structure de projet

https://github.com/Tronix117/wik-njs-303-skeleton

TD3 : Structure et authentification

1. Télécharger le squelette vu précédemment
2. Intégrer les éléments du TD2
3. Rajouter un champ mot de passe aux utilisateurs
4. Utiliser le module `bcrypt` pour hasher le mot de passe
5. Ajouter le middleware `cookie-parser`
6. Créer une table `sessions` (userId, accessToken, createdAt, expiresAt)
7. Créer une resource REST `/sessions`
   - GET / => Affiche un formulaire user/pass
   - POST / => Génère un accessToken et l'enregistre dans la table `sessions`: en HTML on set un cookie `accessToken`, en JSON, on retourne simplement `{accessToken: XXXX}`
   - DELETE / => Supprime un accessToken
8. Créer un middleware qui gère l'authentification :
   - Si mode HTML, alors on vérifie le cookie AccessToken
   - Si mode JSON, alors on vérifie le header X-AccessToken
   - Si pas d'accessToken ou accessToken expiré, en JSON on retourne une erreur, en HTML on redirige vers la page d'authentification

```
// Pour générer un accessToken aléatoire
require('crypto').randomBytes(48, function(err, buffer) {
  let token = buffer.toString('hex')
})
```

TP Todo List

TP : Todo List

1. Reprendre le TD3
2. Ajouter une table `todos`
3. Créer la resources REST `/todos` (userId, message, createdAt, updatedAt, completedAt)
4. La page d'accueil redirige vers `/todos`
5. Chaque utilisateur ne peut voir que ces propres Todos, chaque utilisateur doit pouvoir cocher une Todo, qui passe alors en fin de liste
6. Bonus : Team
   - Un utilisateur peut appartenir à une seule Team (ou aucune)
   - Un utilisateur appartenant à une team a une option "Voir mes todos / Voir les todos de mon équipe"
   - Les todos peuvent être assignées à un utilisateur de la même team
   - Les todos peuvent être terminées par n'importe quel utilisateur

[X] *Sortir les poubelles (par MaChérie, pour moi, complété le 11/10 à 7h30)*

Félicitations !!

Cours WIK-NJS-302 burned :)