

# Express

Web Application Framework

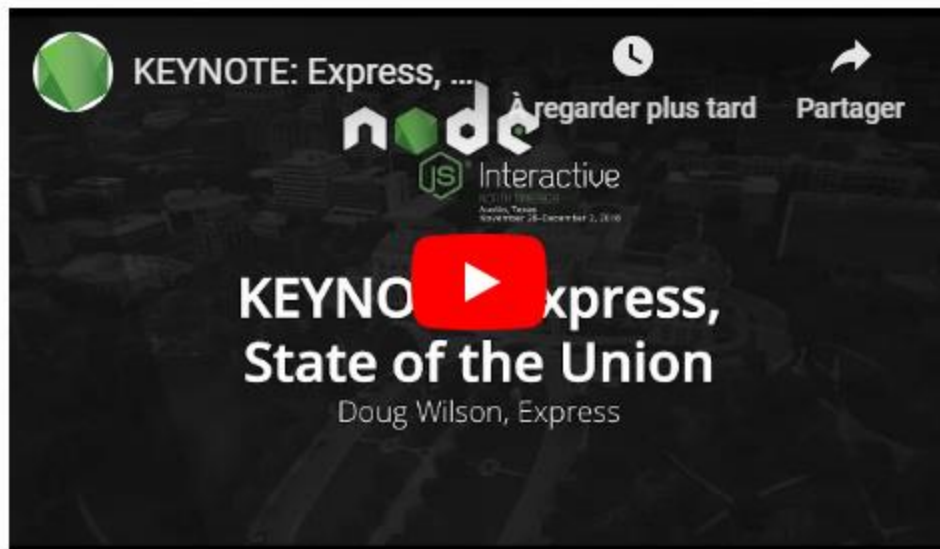
# Agenda

- ▶ Présentation
- ▶ Le routage
- ▶ Les middlewares
- ▶ Les templates

# Express

Infrastructure Web  
minimaliste, souple  
et rapide pour  
**Node.js**

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



# Présentation

- ▶ Express est un serveur web pour Node.js. Il étend les fonctionnalités de base du serveur web de Node.js (http)
- ▶ Express doit être vu comme un framework dans le sens où il apporte diverses fonctionnalités :
  - ▶ Serveur web HTTP
  - ▶ Module de routage (en fonction de l'URI, exécution d'une action précise) : dans le même principe que les routeurs des framework web modernes (Symfony2, Laravel, Django).
  - ▶ Middleware (fonctions appelées après routage)
  - ▶ Gestion des erreurs

# Routage de base

- ▶ **Routage** fait référence à la détermination de la méthode dont une application répond à une demande client adressée à un noeud final spécifique, c'est-à-dire un URI (ou chemin) et une méthode de demande HTTP (GET, POST, etc.).
- ▶ Chaque route peut avoir une ou plusieurs fonctions de gestionnaire, qui sont exécutées lorsque la route est mise en correspondance.

- ▶ *La définition de la route utilise la structure suivante :*

***app.METHOD(PATH, HANDLER)***

*Où :*

- ▶ *app est une instance d'express.*
- ▶ *METHOD est une méthode de demande HTTP (POST,GET,PATCH,DELETE ...).*
- ▶ *PATH est un chemin sur le serveur.*
- ▶ *HANDLER est la fonction exécutée lorsque la route est mise en correspondance.*

Les exemples suivants illustrent la définition de routes simples.

Réponse Hello World! sur la page d'accueil :

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

Réponse à une demande POST sur la route racine (/), sur la page d'accueil de l'application :

```
app.post('/', function (req, res) {  
  res.send('Got a POST request');  
});
```

Réponse à une demande PUT sur la route /user :

```
app.put('/user', function (req, res) {  
  res.send('Got a PUT request at /user');  
});
```

Réponse à une demande DELETE sur la route /user :

```
app.delete('/user', function (req, res) {  
  res.send('Got a DELETE request at /user');  
});
```

# Méthodes de routage

- ▶ Une méthode de routage est dérivée de l'une des méthodes HTTP, et est liée à une instance de la classe express.
- ▶ Express prend en charge les méthodes de routage suivantes qui correspondent aux méthodes HTTP  
get, post, put, head, delete, options, trace, copy, lock, mkcol, move, purge, propfind, proppatch, unlock, report, mkactivity, checkout, merge, m-search, notify, subscribe, unsubscribe, patch, search, and connect.



# Méthodes de routage

- La méthode `app.all()`, qui n'est pas dérivée d'une méthode HTTP. Cette méthode est utilisée pour charger des fonctions middleware à un chemin d'accès pour toutes les méthodes de demande.

```
app.all('/secret', function (req, res, next) {  
  console.log('Accessing the secret section ...');  
  next(); // pass control to the next handler  
});
```

# Chemins de routage (cahînes)

Ce chemin de routage fera correspondre des demandes à la route racine, /.

```
app.get('/', function (req, res) {  
  res.send('root');  
});
```

Ce chemin de routage fera correspondre des demandes à /about.

```
app.get('/about', function (req, res) {  
  res.send('about');  
});
```

Ce chemin de routage fera correspondre des demandes à /random.text.

```
app.get('/random.text', function (req, res) {  
  res.send('random.text');  
});
```

# Chemins de routage (masque de chaînes)

Ce chemin de routage fait correspondre `acd` et `abcd`.

```
app.get('/ab?cd', function(req, res) {  
  res.send('ab?cd');  
});
```

Ce chemin de routage fait correspondre `abcd`, `abbc`, `abbbcd`, etc.

```
app.get('/ab+cd', function(req, res) {  
  res.send('ab+cd');  
});
```

Ce chemin de routage fait correspondre `abcd`, `abxcd`, `abRABDOMcd`, `ab123cd`, etc.

```
app.get('/ab*cd', function(req, res) {  
  res.send('ab*cd');  
});
```

Ce chemin de routage fait correspondre `/abe` et `/abcde`.

```
app.get('/ab(cd)?e', function(req, res) {  
  res.send('ab(cd)?e');  
});
```

# Chemins de routage (expressions régulières)

Ce chemin de routage fera correspondre tout élément dont le nom de chemin comprend la lettre "a".

```
app.get(/a/, function(req, res) {  
  res.send('/a/');  
});
```

Ce chemin de routage fera correspondre butterfly et dragonfly, mais pas butterflyman, dragonfly man, etc.

```
app.get(/.*fly$/, function(req, res) {  
  res.send('/.*fly$/');  
});
```

# Gestionnaires de routage

Une fonction de rappel unique peut traiter une route.

```
app.get('/example/a', function (req, res) {  
  res.send('Hello from A!');  
});
```

Plusieurs fonctions de rappel peuvent traiter une route

```
app.get('/example/b', function (req, res, next) {  
  console.log('the response will be sent by the next function ...');  
  next();  
}, function (req, res) {  
  res.send('Hello from B!');  
});
```

Un tableau de fonctions de rappel peut traiter une route.

```
var cb0 = function (req, res, next) {  
  console.log('CB0');  
  next();  
}  
  
var cb1 = function (req, res, next) {  
  console.log('CB1');  
  next();  
}  
  
var cb2 = function (req, res) {  
  res.send('Hello from C!');  
}  
  
app.get('/example/c', [cb0, cb1, cb2]);
```

## Une combinaison de fonctions indépendantes et des tableaux de fonctions

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

```
var cb1 = function (req, res, next) {  
  console.log('CB1');  
  next();  
}
```

```
app.get('/example/d', [cb0, cb1], function (req, res, next) {  
  console.log('the response will be sent by the next function ...');  
  next();  
}, function (req, res) {  
  res.send('Hello from D!');  
});
```



## 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.

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

# express.Router

- Utilisez la classe `express.Router` pour créer des gestionnaires de route modulaires

birds.js

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

// middleware that is specific to this router
router.use(function timeLog(req, res, next) {
  console.log('Time: ', Date.now());
  next();
});

// define the home page route
router.get('/', function(req, res) {
  res.send('Birds home page');
});

// define the about route
router.get('/about', function(req, res) {
  res.send('About birds');
});

module.exports = router;
```

```
var birds = require('./birds');
...
app.use('/birds', birds);
```

Remarque: Si vous voulez gérer les erreurs 404, vous devez inclure les lignes suivantes **à la fin de votre code** obligatoirement (juste avant `app.listen`):

```
// ... Tout le code de gestion des routes (app.get) se trouve au dessus  
app.use(function(req, res, next){  
  res.setHeader('Content-Type', 'text/plain');  
  res.send(404, 'Page introuvable !');  
});  
app.listen(8080);
```

# Méthodes de réponse

Méthode	Description
<a href="#"><u>res.download()</u></a>	Vous invite à télécharger un fichier.
<a href="#"><u>res.end()</u></a>	Met fin au processus de réponse.
<a href="#"><u>res.json()</u></a>	Envoie une réponse JSON.
<a href="#"><u>res.jsonp()</u></a>	Envoie une réponse JSON avec une prise en charge JSONP.
<a href="#"><u>res.redirect()</u></a>	Redirige une demande.
<a href="#"><u>res.render()</u></a>	Génère un modèle de vue.
<a href="#"><u>res.send()</u></a>	Envoie une réponse de divers types.
<a href="#"><u>res.sendFile</u></a>	Envoie une réponse sous forme de flux d'octets.
<a href="#"><u>res.sendStatus()</u></a>	Définit le code de statut de réponse et envoie sa représentation sous forme de chaîne comme corps de réponse.

# Les middlewares

- ▶ Les middleware sont des fonctions qui gèrent les requêtes. Elles se situent entre la requête brute et la réponse finale (d'où le nom « middle ware » : entre deux).
- ▶ **Les middlewares** peuvent accéder à l'objet Request (req), l'objet response (res) et à la fonction middleware suivante dans le cycle demande-réponse de l'application.
- ▶ La fonction middleware suivante est couramment désignée par une variable nommée next.
- ▶ Les fonctions middleware effectuent les tâches suivantes :
  - ▶ 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.
- ▶

- ▶ Le serveur créé avec Express peut avoir une pile de fonctions middleware. Quand une requête arrive, elle est déléguée à la première fonction middleware de la pile.
- ▶ Les fonctions middleware sont toujours invoquées dans l'ordre où elles sont ajoutées.
- ▶ Chaque fonction middleware peut accéder aux objets request et response, ainsi qu'à la fonction middleware suivante dans la pile.
- ▶ Chaque fonction middleware peut décider de « répondre » en appelant des méthodes sur l'objet response, et / ou en passant la requête à la fonction middleware suivante dans la pile en appelant la fonction next().
- ▶ Si une fonction middleware ne termine pas le cycle requête / réponse, elle doit obligatoirement appeler next() pour passer le contrôle à la fonction middleware suivante.

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

Méthode HTTP à laquelle la fonction middleware s'applique.

Chemin (route) auquel la fonction middleware s'applique.

Fonction de middleware.

```
app.get('/', function(req, res, next) {  
  next();  
})
```

Argument de rappel à la fonction middleware, appelée "next" par convention.

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

Argument de **réponse** HTTP à la fonction middleware, appelé "res" par convention.

Argument de **demande** HTTP à la fonction middleware, appelé "req" par convention.

# Middleware: Example1

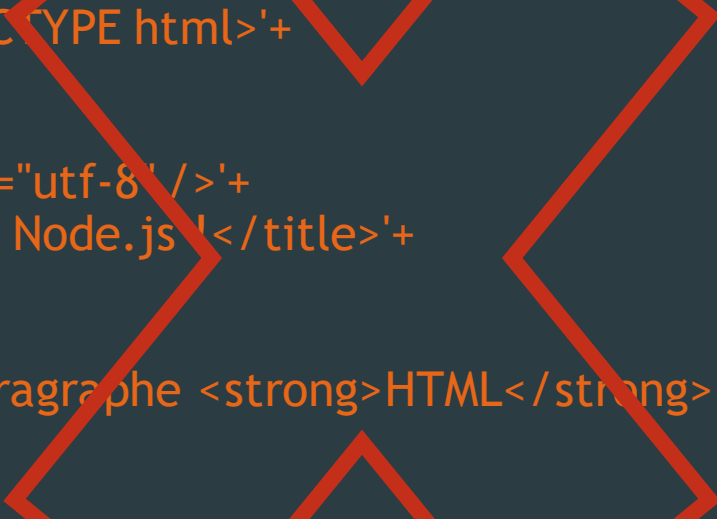
```
var express = require('express');
var app = express();
var myLogger = function (req, res, next) {
  console.log('LOGGED');
  next();
};
app.use(myLogger);
app.get('/', function (req, res) {
  res.send('Hello World!');
});
app.listen(3000);
```



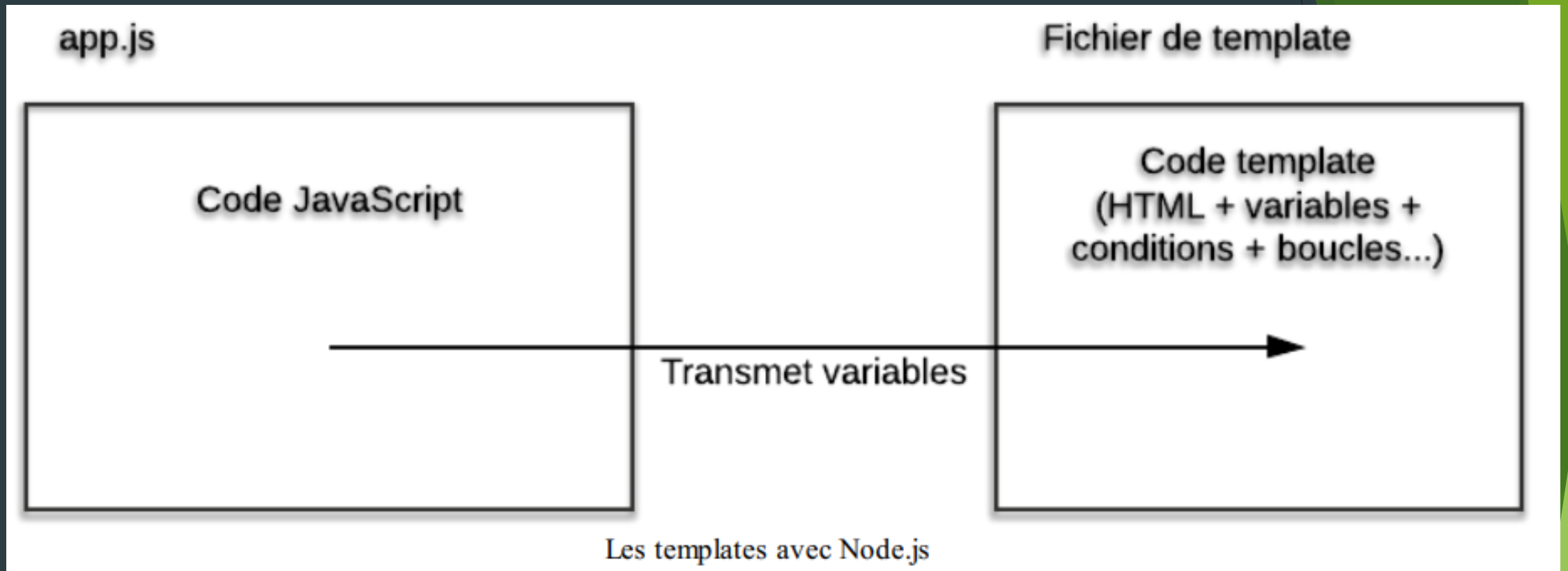
# Middleware: Example2

```
var express = require('express');
var app = express();
var requestTime = function (req, res, next) {
  req.requestTime = Date.now();
  next();
};
app.use(requestTime);
app.get('/', function (req, res) {
  var responseText = 'Hello World!';
  responseText += 'Requested at: ' + req.requestTime + ' ';
  res.send(responseText);
});
app.listen(3000);
```

# Les templates



- `res.write('<!DOCTYPE html>'+  
'<html>'+  
'<head>'+  
'<meta charset="utf-8" />'+  
'<title>Ma page Node.js'</title>'+  
'</head>'+  
'<body>'+  
'<p>Voici un paragraphe <strong>HTML</strong> !</p>'+  
'</body>'+  
'</html>');`



Langages de templates, : Twig, Smarty, Haml, JSP, Jade, EJS, Mustache, Dust, ...

<https://colorlib.com/wp/top-templating-engines-for-javascript/>

# EJS: Exemple1

App1.js

```
app.get('/EJSDemo', function(req, res) {  
  res.render('demo.ejs');  
});
```

Ce code fait appel à un fichier **accueil.ejs** qui doit se trouver dans un sous-dossier appelé "**views**"

/views/demo.ejs

```
<h1>Ceci est une démonstration de EJS</h1>
```

# EJS: Exemple2

App2.js

```
app.get('/Participants', function(req, res) {  
  var TabNoms = ['Rodica', 'Lenia', 'Rémi', 'David', 'Sebastien'];  
  res.render('participants.ejs', {noms: TabNoms});  
});
```

/views/participants.ejs

```
<p>Un participant au hasard:  
<%= noms[Math.floor(Math.random() * (noms.length + 1))] %>  
</p>
```

# Références

- ▶ <https://expressjs.com>
- ▶ <https://openclassrooms.com/fr/courses/1056721-des-applications-ultra-rapides-avec-node-js>