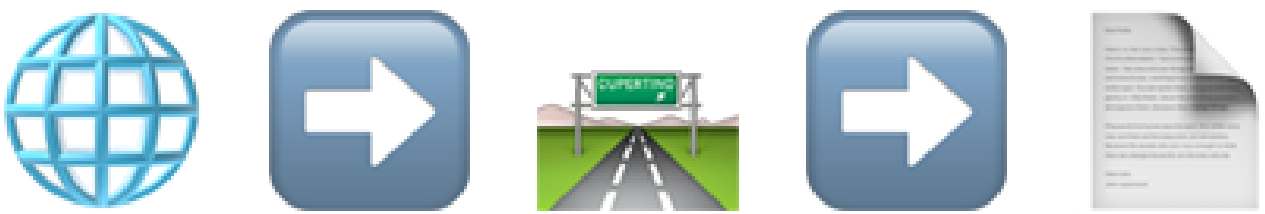
🚀 Express.js & Routing
BUT2 - S3 INFO

Du serveur HTTP aux APIs modernes

Avec les Template Engines (EJS)



Joseph Azar

Maître de Conférences - Université Marie Louis Pasteur

Chercheur FEMTO-ST

🤔 C'est quoi Express.js ?

> **Express.js** est un framework minimaliste et flexible pour Node.js qui facilite la création de serveurs web et d'APIs.

Express simplifie le développement de serveurs web en Node.js

🚫 Sans Express (Node.js pur)

Créer un serveur avec Node.js pur nécessite beaucoup de code

```
const http = require('http');

const server = http.createServer((req, res) => {
  if (req.url === '/' && req.method === 'GET') {
    res.writeHead(200, {'Content-Type': 'text/html'});
    res.end('Hello World');
  } else if (req.url === '/about' && req.method === 'GET') {
    res.writeHead(200, {'Content-Type': 'text/html'});
    res.end('About page');
  } else {
    res.writeHead(404);
    res.end('Not found');
  }
});

server.listen(3000);
```

> **Problème :** Code verbeux et difficile à maintenir

✅ Avec Express

Le même serveur, beaucoup plus simple

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

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

app.get('/about', (req, res) => {
  res.send('About page');
});

app.listen(3000, () => {
  console.log('Server started on port 3000');
});
```

> ✨ **Beaucoup plus simple et lisible!**

📦 Installation d'Express

> **Étape 1 : Initialiser un projet Node.js**

```
npm init -y
```

Cette commande crée un fichier `package.json` avec la configuration par défaut

📦 Installation d'Express (suite)

Étape 2 : Installer Express

```
npm install express
```

Cette commande télécharge et installe Express dans votre projet

📦 Votre Premier Serveur Express

```
// app.js
const express = require('express');
const app = express();
const port = 3000;

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

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

Lancer le serveur : `node app.js`

Le Routing avec Express

Comment gérer les différentes URLs de votre application

🎯 Les Routes Basiques

Une route détermine comment votre application répond à une requête client pour un endpoint spécifique

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

// Route GET - Récupérer des données
app.get('/', (req, res) => {
  res.send('Page d\'accueil');
});

// Route POST - Envoyer des données
app.post('/contact', (req, res) => {
  res.send('Formulaire de contact soumis');
});

// Route PUT - Mettre à jour des données
app.put('/user/:id', (req, res) => {
  res.send(`Mise à jour de l'utilisateur ${req.params.id}`);
});

// Route DELETE - Supprimer des données
app.delete('/user/:id', (req, res) => {
  res.send(`Suppression de l'utilisateur ${req.params.id}`);
});
```

📡 Méthodes HTTP Principales

- **GET** : Récupérer des ressources
- **POST** : Créer de nouvelles ressources
- **PUT/PATCH** : Mettre à jour des ressources
- **DELETE** : Supprimer des ressources

Ces méthodes correspondent aux opérations CRUD : **C**reate, **R**ead, **U**pdate, **D**elete

📍 Paramètres de Route

Les paramètres permettent de capturer des valeurs dynamiques dans l'URL

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

// Paramètre simple
app.get('/users/:id', (req, res) => {
  const userId = req.params.id;
  res.send(`Profil de l'utilisateur ${userId}`);
});

// Plusieurs paramètres
app.get('/users/:userId/posts/:postId', (req, res) => {
  const { userId, postId } = req.params;
  res.send(`Post ${postId} de l'utilisateur ${userId}`);
});

// Paramètre avec regex (seulement des nombres)
app.get('/products/:id(\\d+)', (req, res) => {
  res.send(`Produit numéro ${req.params.id}`);
});
```

📌 Exemples de Paramètres de Route

```
Exemples d'URLs :

GET /users/123
→ userId = "123"

GET /users/42/posts/7
→ userId = "42", postId = "7"

GET /products/456
→ id = "456"
```

Important : Les paramètres de route sont **obligatoires**

🔍 Query Strings (Chaînes de Requête)

Les query strings permettent de passer des paramètres **optionnels** dans l'URL

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

// Exemple : Recherche avec filtres
app.get('/search', (req, res) => {
  const { q, category, sort } = req.query;

  res.json({
    searchTerm: q,
    category: category || 'all',
    sortBy: sort || 'relevance'
  });
});

// Exemple : Pagination
app.get('/products', (req, res) => {
  const page = parseInt(req.query.page) || 1;
  const limit = parseInt(req.query.limit) || 10;

  res.json({
    page,
    limit,
```

🔍 Exemples de Query Strings

Exemples d'URLs :

```
GET /search?
q=laptop&category=electronics&sort=price

GET /products?page=2&limit=20
```

📝 Différence : Params vs Query

Type	Syntaxe	Exemple	Utilisation
Route Params	/users/:id	/users/123	Obligatoires, font partie du chemin
Query Strings	?page=1&limit=10	/products?page=2	Optionnels, séparés par ?

Règle générale :

- Utilisez **params** pour identifier une ressource
- Utilisez **query** pour filtrer ou paginer

L'objet Request (req)

L'objet `req` contient toutes les informations sur la requête HTTP

```
app.get('/demo', (req, res) => {
  console.log('URL:', req.url);               // /demo?name=john
  console.log('Method:', req.method);         // GET
  console.log('Params:', req.params);         // Paramètres de route
  console.log('Query:', req.query);           // { name: 'john' }
  console.log('Headers:', req.headers);       // En-têtes HTTP
  console.log('Body:', req.body);             // Corps (POST/PUT)
  console.log('IP:', req.ip);                 // Adresse IP
  console.log('Path:', req.path);             // /demo
  console.log('Hostname:', req.hostname);     // localhost

  res.send('Voir la console!');
});
```

 Propriétés de Request - Résumé

Les plus utilisées :

- `req.params` : Paramètres de route (/users/:id)
- `req.query` : Query strings (?page=1)
- `req.body` : Corps de la requête (POST/PUT)
- `req.headers` : En-têtes HTTP
- `req.method` : Méthode HTTP (GET, POST...)
- `req.url` : URL complète

📥 Méthodes de Response (res)

L'objet `res` permet d'envoyer des réponses au client

```
// Envoyer du texte simple
app.get('/text', (req, res) => {
  res.send('Hello World');
});

// Envoyer du JSON
app.get('/json', (req, res) => {
  res.json({ message: 'Hello', status: 'success' });
});

// Envoyer un fichier HTML
app.get('/html', (req, res) => {
  res.sendFile(__dirname + '/index.html');
});

// Rediriger
app.get('/old-page', (req, res) => {
  res.redirect('/new-page');
});
```

📥 Méthodes de Response - Status Codes

```
// Définir le status code
app.get('/not-found', (req, res) => {
  res.status(404).send('Page non trouvée');
});

// Chaîner les méthodes
app.get('/api/user', (req, res) => {
  res.status(200).json({ name: 'John', age: 30 });
});

// Créer une ressource (201 Created)
app.post('/api/user', (req, res) => {
  res.status(201).json({ message: 'Utilisateur créé' });
});

// Erreur serveur
app.get('/error', (req, res) => {
  res.status(500).json({ error: 'Erreur serveur' });
});
```

⚙️ Les Middlewares

Un middleware est une fonction qui a accès aux objets req et res

> **Analogie :** Un middleware c'est comme un contrôle de sécurité à l'aéroport. Chaque passager (requête) doit passer par plusieurs contrôles avant d'embarquer (atteindre la route)

Requête → Middleware 1 → Middleware 2 → Route

⚙️ Exemple de Middleware

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

// Middleware de logging - pour TOUTES les routes
app.use((req, res, next) => {
  console.log(`[${new Date().toISOString()}] ${req.method} ${req.url}`);
  next(); // IMPORTANT : passer au suivant
});

// Middleware pour une route spécifique
app.use('/admin', (req, res, next) => {
  console.log('Accès à la zone admin');
  next();
});

// Route finale
app.get('/admin/dashboard', (req, res) => {
  res.send('Dashboard admin');
});

app.listen(3000);
```

⚠️ **N'oubliez jamais `next()`!** Sans ça, la requête reste bloquée.

🔧 Middlewares Intégrés

Express fournit plusieurs middlewares essentiels

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

// 1. Parser les données JSON (req.body)
app.use(express.json());

// 2. Parser les formulaires URL-encoded
app.use(express.urlencoded({ extended: true }));

// 3. Servir des fichiers statiques (CSS, images, JS)
app.use(express.static('public'));
```

🔧 Utiliser req.body

Après avoir ajouté les middlewares, vous pouvez accéder au corps de la requête

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

// Middleware requis pour req.body
app.use(express.json());
app.use(express.urlencoded({ extended: true }));

// Maintenant on peut utiliser req.body!
app.post('/user', (req, res) => {
  console.log(req.body); // { name: 'John', age: 30 }

  res.json({
    message: 'Utilisateur créé',
    data: req.body
  });
});

app.listen(3000);
```

📁 Fichiers Statiques

Structure des dossiers :

```
project/
├── app.js
└── public/
    ├── css/
    │   └── style.css
    ├── js/
    │   └── script.js
    └── images/
        └── logo.png
```

Avec `app.use(express.static('public'))`, les fichiers sont accessibles directement :

```
http://localhost:3000/css/style.css
http://localhost:3000/js/script.js
http://localhost:3000/images/logo.png
```

Express Router

Le Router permet de créer des modules de routes réutilisables

```
// routes/users.js
const express = require('express');
const router = express.Router();

// Middleware pour ce routeur uniquement
router.use((req, res, next) => {
  console.log('User route accessed');
  next();
});

// Routes relatives au routeur
router.get('/', (req, res) => {
  res.send('Liste des utilisateurs');
});

router.get('/:id', (req, res) => {
  res.send(`Utilisateur ${req.params.id}`);
});

router.post('/', (req, res) => {
  res.send('Créer un utilisateur');
});
```

Utiliser le Router

```
// app.js
const express = require('express');
const app = express();
const userRoutes = require('./routes/users');

// Monter le routeur
app.use('/api/users', userRoutes);

app.listen(3000);
```

URLs disponibles :

GET /api/users → Liste des utilisateurs
GET /api/users/123 → Utilisateur 123
POST /api/users → Créer un utilisateur

🚨 Gestion des Erreurs

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

// Routes normales
app.get('/', (req, res) => {
  res.send('Home page');
});

// Middleware 404 - AVANT le middleware d'erreur
app.use('*', (req, res, next) => {
  const error = new Error('Route non trouvée');
  error.status = 404;
  next(error);
});

// Middleware d'erreurs - TOUJOURS en dernier
// Reconnu par ses 4 paramètres (err, req, res, next)
app.use((err, req, res, next) => {
  console.error(err.stack);
  res.status(err.status || 500).json({
    error: {
      message: err.message,
      status: err.status || 500
    }
  });
});
```

🚨 Ordre de Gestion des Erreurs

L'ordre est crucial :

1. Routes normales
2. Middleware 404 (catch all)
3. Middleware de gestion d'erreurs

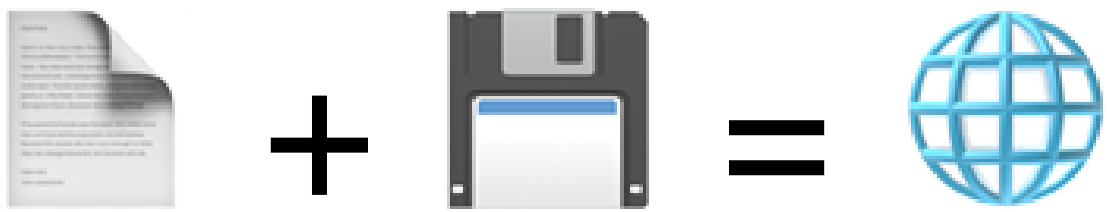
Routes → 404 Handler → Error Handler

Template Engines

Générer du HTML dynamique avec EJS

🎨 C'est quoi un Template Engine ?

> **Un template engine** permet de générer du HTML dynamique côté serveur en injectant des données dans des templates.



Template + Données = HTML dynamique

🚫 Sans Template Engine

Générer du HTML dans le code JavaScript

```
app.get('/user', (req, res) => {
  const name = 'John';
  const age = 30;

  const html = `
    <html>
      <head><title>Profile</title></head>
      <body>
        <h1>${name}</h1>
        <p>Age: ${age}</p>
      </body>
    </html>
  `;

  res.send(html);
});
```

Problème : Difficile à maintenir, mélange HTML et JavaScript

✅ Avec EJS - Partie Serveur

```
// app.js
const express = require('express');
const app = express();

// Configurer EJS
app.set('view engine', 'ejs');

app.get('/user', (req, res) => {
  // Rendre le template avec des données
  res.render('user', {
    name: 'John',
    age: 30
  });
});

app.listen(3000);
```

✅ Avec EJS - Template

```ejs
<!-- views/user.ejs -->
<!DOCTYPE html>
<html>
  <head>
    <title>Profile</title>
  </head>
  <body>
    <h1><%= name %></h1>
    <p>Age: <%= age %></p>
  </body>
</html>
```

> ✨ **Avantages :** Séparation du code et de la vue, plus facile à maintenir

🆚 Template Engines vs Frameworks

Caractéristique	Template Engines (EJS, Pug)	Frameworks (React, Vue)
Rendu	Côté serveur (SSR)	Côté client (CSR)
HTML généré	Au moment de la requête	Dans le navigateur
Interactivité	Limitée	Très interactive (SPA)
SEO	Excellent	Plus complexe
Complexité	Simple	Plus complexe

🤔 Quand utiliser quoi ?

✅ **Template Engines (EJS, Pug)**

- Sites web classiques
- Blogs et sites de contenu
- Applications simples
- Quand le SEO est crucial
- Pages à faible interactivité

🤔 Quand utiliser des Frameworks ?

✅ **Frameworks Frontend (React, Vue, Angular)**

- Applications web complexes
- Dashboards interactifs
- Single Page Applications (SPA)
- Applications type Google Docs
- Forte interactivité requise

🔧 Installation d'EJS

Étape 1 : Installation

```
npm install ejs
```

Étape 2 : Configuration

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

// Définir EJS comme moteur de template
app.set('view engine', 'ejs');

// Optionnel : changer le dossier des views
app.set('views', './my-views');
```

📁 Structure d'un Projet EJS

```
project/
├── app.js
├── package.json
└── views/
    ├── index.ejs
    ├── about.ejs
    └── partials/
        ├── header.ejs
        └── footer.ejs
```

> **Par défaut**, Express cherche les templates dans le dossier `views/`

Syntaxe EJS - Variables

```
<!-- Afficher une variable (échappée) -->
<p>Nom: <%= name %></p>

<!-- Afficher du HTML brut (ATTENTION : risque XSS) -->
<div><%- htmlContent %></div>
```

⚠️ **Sécurité :**

- `<%= %>` : Échappe le HTML (sécurisé)
- `<%- %>` : N'échappe pas (risque XSS)

Syntaxe EJS - Code JavaScript

```
<!-- Code JavaScript (sans affichage) -->
<% const greeting = "Hello"; %>
<% const fullGreeting = greeting + " " + name; %>

<!-- Afficher la variable créée -->
<p><%= fullGreeting %></p>

<!-- Commentaires EJS (n'apparaissent pas dans le HTML final) -->
<%# Ceci est un commentaire EJS %>
```

Syntaxe EJS - Conditions

```
<% if (age >= 18) { %>
  <p>Vous êtes majeur</p>
<% } else { %>
  <p>Vous êtes mineur</p>
<% } %>

<!-- Condition ternaire -->
<p>Statut: <%= user.isAdmin ? 'Admin' : 'Utilisateur' %></p>

<!-- Vérifier l'existence -->
<% if (user.email) { %>
  <p>Email: <%= user.email %></p>
<% } %>
```

Syntaxe EJS - Boucles

```
<!-- Boucle forEach -->
<ul>
  <% users.forEach(user => { %>
    <li><%= user.name %> - <%= user.email %></li>
  <% }); %>
</ul>

<!-- Boucle for classique -->
<ul>
  <% for (let i = 0; i < products.length; i++) { %>
    <li><%= products[i].name %></li>
  <% } %>
</ul>

<!-- Boucle for...of -->
<% for (const product of products) { %>
  <p><%= product.name %>: <%= product.price %>€</p>
<% } %>
```

📝 Résumé des Balises EJS

`<%= %>`	Afficher une variable (échappée)
`<%- %>`	Afficher du HTML brut
`<% %>`	Code JavaScript
`<%# %>`	Commentaire

🎯 Exemple Complet - Serveur

```
// app.js
const express = require('express');
const app = express();

app.set('view engine', 'ejs');

app.get('/', (req, res) => {
  const data = {
    title: 'Ma Page d\'accueil',
    user: {
      name: 'Alice',
      age: 25,
      isAdmin: true
    },
    products: [
      { id: 1, name: 'Laptop', price: 999 },
      { id: 2, name: 'Mouse', price: 25 },
      { id: 3, name: 'Keyboard', price: 75 }
    ]
  };

  res.render('index', data);
});

app.listen(3000);
```

🎯 Exemple Complet - Template

```ejs
<!-- views/index.ejs -->
<!DOCTYPE html>
<html>
<head>
  <title><%= title %></title>
</head>
<body>
  <h1>Bienvenue, <%= user.name %>!</h1>

  <% if (user.isAdmin) { %>
    <p>🔑 Vous êtes administrateur</p>
  <% } %>

  <h2>Nos Produits</h2>
  <ul>
    <% products.forEach(product => { %>
      <li>
        <strong><%= product.name %></strong> -
        <%= product.price %>€
      </li>
    <% }); %>
  </ul>
</body>
</html>
```

🧩 EJS Partials - Header

Les partials permettent de réutiliser des morceaux de templates

```
<!-- views/partials/header.ejs -->
<!DOCTYPE html>
<html>
<head>
  <title><%= title %></title>
  <link rel="stylesheet" href="/css/style.css">
</head>
<body>
  <nav>
    <a href="/">Accueil</a>
    <a href="/about">À propos</a>
    <a href="/contact">Contact</a>
  </nav>
```

🧩 EJS Partials - Footer

```ejs
<!-- views/partials/footer.ejs -->
  <footer>
    <p>© 2025 Mon Site Web</p>
    <p>
      <a href="/privacy">Politique de confidentialité</a> |
      <a href="/terms">Conditions d'utilisation</a>
    </p>
  </footer>
</body>
</html>
```

🧩 Utiliser les Partials

```
<!-- views/index.ejs -->
<%- include('partials/header') %>

  <h1>Page d'accueil</h1>
  <p>Contenu de la page...</p>

<%- include('partials/footer') %>
```

```
<!-- views/about.ejs -->
<%- include('partials/header') %>

  <h1>À propos</h1>
  <p>Informations sur notre site...</p>

<%- include('partials/footer') %>
```

> ✨ **Avantage :** Modifier le header/footer une fois met à jour toutes les pages!

📝 Récapitulatif - Express.js

- ✅ Framework minimaliste pour Node.js
- ✅ Routing simple et puissant
- ✅ Middlewares pour étendre les fonctionnalités
- ✅ Gestion des erreurs structurée
- ✅ Support des fichiers statiques
- ✅ Organisation avec Express Router

📝 Récapitulatif - Template Engines

- ✅ Génération HTML côté serveur
- ✅ EJS : syntaxe simple avec <%= %>
- ✅ Partials pour réutiliser du code
- ✅ Séparation code/présentation
- ✅ Excellent pour le SEO
- ✅ Différent des frameworks frontend

🔑 Points Clés à Retenir

- **Params** (/users/:id) : Obligatoires, partie du chemin
- **Query** (?page=1) : Optionnels, paramètres de filtre
- **Middleware** : Fonctions qui s'exécutent avant les routes
- **next()** : Ne jamais l'oublier dans un middleware
- **Template Engines** : Pour sites web classiques avec bon SEO

🔮 La Semaine Prochaine

REST API & RESTful Design

- 🎯 Principes de l'architecture REST
- 📡 Créer des APIs RESTful complètes
- 🔐 Authentification et sécurité
- 📊 Bonnes pratiques API
- 🧪 Tester ses APIs

📚 Ressources - Documentation

Documentation officielle

- 📖 Express.js : expressjs.com
- 📖 EJS : ejs.co
- 📖 MDN HTTP : developer.mozilla.org
- 📖 Node.js : nodejs.org

📚 Ressources - Outils

Outils utiles pour le développement

- 🔧 Postman : Tester les APIs
- 🔧 Thunder Client : Extension VS Code
- 🔧 nodemon : Auto-restart du serveur
- 🔧 REST Client : Extension VS Code

? Questions ?

Place au TD pratique!



Joseph Azar

Maître de Conférences - Université Marie Louis Pasteur

Chercheur FEMTO-ST

joseph.azar@univ-fcomte.fr