

# NodeJS : bases d'une application

- Structure
- Serveur HTTP
- Gestion des requêtes et des réponses
- Comment exploiter l'URL
- Gestion de plusieurs URL
- Affichage des pages

# Structure de l'application

- Pour **créer une application**, on a besoin des éléments suivants :
  - Un **serveur HTTP**, pour servir les pages Web
  - Un **routeur**, pour répondre en fonction des URL
  - Un **gestionnaire de requêtes**, pour rediriger et traiter les requêtes
  - Un **modèle de vues**, pour afficher le contenu

# Serveur HTTP : `server.js`

- Inclure le module **http** de NodeJS
- Créer un serveur avec la méthode **createServer()**
- Ecouter un port (8888) avec la méthode **listen()**
- Recevoir une requête, objet **request**
- Envoyer une réponse, objet **response**
- Démarrer le serveur : **node server.js**
- Dans le navigateur : **http://localhost:8888**

# server.js : créer le serveur

- Avec l'instruction **require()**, on ajoute un module NodeJS. Le module http est déjà inclus dans l'installation de base de NodeJS
- La méthode **createServer()** permet de créer un serveur Web qui, avec la méthode **listen()** va écouter sur un certain port [ici : 8888]
- Mais rien ne se passe, puisqu'on ne gère aucune requête et on n'envoie aucune réponse

```
const Http = require('http');  
Http.createServer().listen(8888);
```

# server.js : envoyer une réponse

- On ajoute une **fonction anonyme** qui va permettre de d'envoyer une réponse correcte (code HTTP = 200) et qui va afficher Hello World sur le navigateur
- Mais on a toujours le même comportement quelque soit la requête

```
const Http = require('http');
Http.createServer(function(request, response){
  response.writeHead(200);
  response.end('Hello World NodeJS !');
}).listen(8888);
```

## server.js : séparer le message

- On sépare pour être plus clair :
  - l'**entête HTTP** avec le code d'état 200 et le type de contenu envoyé avec son code MIME [*response.writeHead()*]
  - le **message** à afficher [*response.write()*]
  - le **signal d'envoi** de la réponse [*response.end()*]

```
const Http = require('http');
Http.createServer(function(request, response){
    response.writeHead(200, {"Content-type":"text/plain"}); // 1
    response.write('Hello World : NodeJS !'); // 2
    response.end(); // 3
}).listen(8888);
```

# Code Status HTTP

- Le **code HTTP** permet de déterminer le résultat d'une requête ou d'indiquer une erreur au client.
- Les plus courants sont :
  - 200 : succès de la requête
  - 401 : utilisateur non authentifié
  - 403 : accès refusé
  - 404 : page non trouvée
  - 500 et 503 : erreur serveur
- Liste sur <https://developer.mozilla.org/fr/>

# Type MIME

- **MIME** : Multipurpose Internet Mail Extensions
- Le type MIME est un identifiant de format de données sur internet. Par exemple :
  - text/html
  - text/css
  - video/mp4
  - image/jpeg
  - application/pdf
- Liste sur : <https://developer.mozilla.org/fr/>



# server.js : envoi de code HTML

- Pour envoyer du HTML sur le navigateur il faut utiliser le type MIME text/html
- Si on veut ajouter des caractères spéciaux dans le texte, il faut ajouter aussi le jeu de caractères

"Content-type": "text/html; charset=utf-8"

```
const Http = require('http');
Http.createServer(function(request, response){
  response.writeHead(200, {"Content-type": "text/html; charset=utf-8"});
  response.write('<h1>Hello World : NodeJS !</h1>');
  response.end();
}).listen(8888);
```

# Le code : server.js

- Pour traiter la requête, on utilise l'objet **request**

```
// inclure le module http
const Http = require('http');
// définir les autres variables
const port = 8888;
var monServeur;

// créer un serveur qui écoute sur le port défini et traite la requête reçue
monServeur = Http.createServer(requeteRecue).listen(port);
console.log("Démarrage du serveur...");

// Fonction de traitement des requêtes
function requeteRecue(request, response) {
    console.log("Requete reçue : " + request.url);
    response.writeHead(200, {"Content-type": "text/html; charset=utf-8"});
    response.write('<h1>NodeJS</h1>');
    response.write('<p>La requête reçue est : ' + request.url + '</p>');
    response.end();
}
```

# Remarque à l'exécution

- Si on place un `console.log()` dans la fonction `requeteRecue()` qui traite la requête, on voit qu'avec certains navigateurs, il y a 2 requêtes :
  - <http://localhost:8888>
  - <http://localhost:8888/favicon.ico>

# Définir un module NodeJS

- Comment transformer le fichier `server.js` pour qu'il se comporte comme un **module NodeJS**
- On veut appeler ce module `server` à partir d'un **fichier de démarrage `index.js`**
- Créer une fonction **`demarrageServeur()`** qui **contient le code de `server.js`** et **exporter cette fonction** pour la rendre publique donc utilisable par d'autres modules

# Utilisation du module server

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

function demarrageServeur() {
  const port = 8888;
  var monServeur;

  function requeteRecue(request, response) {
    console.log("Requete reçue par demarrageServeur : " + request.url);
    response.writeHead(200, {"Content-type": "text/html; charset=utf-8"});
    response.write('<h1>NodeJS</h1>');
    response.write('<p>La requête reçue est : ' + request.url + '</p>');
    response.end();
  }

  // créer un serveur qui écoute sur le port défini et traite la requête reçue
  monServeur = Http.createServer(requeteRecue).listen(port);
  console.log("Démarrage du serveur via index.js");
}

exports.demarrer = demarrageServeur;
```

httpServeur.js

index.js

```
const serveurHTTP = require('./httpServeur');

console.log('Lancement de index.js');
serveurHTTP.demarrer();
```

# Affichage du HTML

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

function demarrageServeur() {
  const port = 8888;
  var monServeur;

  function requeteRecue(request, response) {
    console.log("Requete reçue par demarrageServeur : " + request.url);
    response.writeHead(200, {"Content-type": "text/html; charset=utf-8"});
    response.write(
      '<!DOCTYPE html>' +
      '<html>' +
      '<head>' +
      '<title>Une page avec NodeJS</title>' +
      '<meta charset="utf-8" />' +
      '</head>' +
      '<body>' +
      '<h1>Résultats avec NodeJS</h1>' +
      '<p>La requête reçue est : ' + request.url + '</p>' +
      '</body>' +
      '</html>'
    );
    response.end();
  }

  // créer un serveur qui écoute sur le port défini et traite la requête reçue
  monServeur = Http.createServer(requeteRecue).listen(port);
  console.log("Démarrage du serveur via index.js");
}

exports.demarrer = demarrageServeur;
```

# Affichage du HTML

- La réponse envoyée n'est pas du code HTML valide, il faut écrire la page complète avec DOCTYPE, <head>, <body>,...
- Processus très lourd et pas flexible
- On va utiliser plus tard des **systèmes de template** pour générer les pages. Il en existe énormément : Mustache, Handlebars, EJS, Underscore, Pug (Jade), React, Vue,...
- Ces systèmes de template peuvent être gérés avec le **framework Express**

# Gestion des requêtes

- Notre système renvoie toujours la même chose quelque soit la page demandée car on ne gère pas les requêtes
- Il faut **détecter le nom de la page demandée** et éventuellement les **paramètres** dans l'URL
- On peut utiliser :
  - le module natif URL pour connaître la page
  - le module natif QueryString pour les paramètres



# Module URL

- Depuis la v7 de NodeJS, on utilise les URL définies par l'API WHATWG et on peut récupérer les différentes parties :
  - Protocole (protocol)
  - Nom d'utilisateur (username)
  - Mot de passe (password)
  - Nom d'hôte (hostname)
  - Port (port)
  - Chemin (pathname)
  - Chaîne de requête (querystring : search)

# Composition de l'URL

## Legacy API (avant NodeJS v7.0.0)

href									
protocol		auth		host		path		hash	
				hostname	port	pathname	search		
							query		
" https: // user : pass @ sub.host.com : 8080 /p/a/t/h ? query=string #hash "									
				hostname	port				
protocol		username	password	host					
origin				origin		pathname	search	hash	
href									

## WHATWG API (après NodeJS v7.0.0)

# Exploiter l'URL

- On charge le module URL

```
var Url = require("url").URL;
```

- On doit récupérer l'URL dans la requête
- On doit ensuite compléter l'adresse avec le nom du serveur et le port si c'est une URL relative
- On peut enfin analyser son contenu pour en extraire toutes les parties (protocole, nom d'hôte, chemin, paramètres,...)

# Les différentes parties de l'URL

## Le code

```
const Http = require('http');
const Url = require('url').URL;

function demarrageServeur() {
  const serveur = 'http://localhost';
  const port = 8888;

  var monServeur;

  function requeteRecue(request, response) {
    let laRequete = request.url;
    let urlComplete = new Url(laRequete, serveur + ':' + port);

    let laPageDemandee = urlComplete.href;
    let laQueryString = urlComplete.search;
    let leChemin = urlComplete.pathname;
    let lePort = urlComplete.port;
    let leHost = urlComplete.hostname;

    response.writeHead(200, {"Content-type": "text/html; charset=utf-8"});
    response.write(
      '<!DOCTYPE html>' +
      '<html>' +
      '<head>' +
      '<title>Une page avec NodeJS</title>' +
      '<meta charset="utf-8" />' +
      '</head>' +
      '<body>' +
      '<h1>Résultats avec NodeJS</h1>' +
      '<p>La requête reçue est : <em>' + laRequete + '</em></p>' +
      '<p>La page demandée est : <em>' + laPageDemandee + '</em></p>' +
      '<p>Le nom du serveur (hostname) est : <em>' + leHost + '</em></p>' +
      '<p>Le port (port) est : <em>' + lePort + '</em></p>' +
      '<p>Les paramètres (search) sont : <em>' + laQueryString + '</em></p>' +
      '<p>Le chemin (pathname) est : <em>' + leChemin + '</em></p>' +
      '</body>' +
      '</html>'
    );
    response.end();
  }

  // créer un serveur qui écoute sur le port défini et traite la requête reçue
  monServeur = Http.createServer(requeteRecue).listen(port);
  console.log("Démarrage du serveur via index.js");
}

exports.demarrer = demarrageServeur;
```

## Le résultat



← → ↻ ⓘ localhost:8888/chemin/page?prenom=pierre&nom=sandron&ville=Bruxelles

Applications CF2mWebDev - Zo... localhost / MySQL |... YouTube Partage Web2020 Main Page

### Résultats avec NodeJS

La requête reçue est : /chemin/page?prenom=pierre&nom=sandron&ville=Bruxelles

La page demandée est : http://localhost:8888/chemin/page?prenom=pierre&nom=sandron&ville=Bruxelles

Le nom du serveur (hostname) est : localhost

Le port (port) est : 8888

Les paramètres (search) sont : ?prenom=pierre&nom=sandron&ville=Bruxelles

Le chemin (pathname) est : /chemin/page

# Analyser la QueryString

- Il faut utiliser la classe **URLSearchParams** définie dans le module **URL**
- La méthode **forEach** parcourt les paramètres

```
const UrlParams = require('url').URLSearchParams;
```

```
let lesParametres = new UrlParams(laQueryString);
```

```
detailsQueryString_HTML(lesParametres)
```

```
function detailsQueryString_HTML(params) {
```

```
    let stringHTML = '';
```

```
    stringHTML+= '<ul>';
```

```
    params.forEach(
```

```
        (valeur, parametre) => { stringHTML+= '<li>Le paramètre '+parametre+' vaut '+valeur+'</li>'; }
```

```
    );
```

```
    stringHTML+= '</ul>';
```

```
    return stringHTML;
```

```
}
```

# Gestion de plusieurs URL

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

function demarrageServeur() {
  const port = 8888;

  var monServeur;

  function requeteRecue(request, response) {
    let laRequete = request.url;

    switch(laRequete) {
      case '/':
      case '/about':
      case '/contacts': response.writeHead(200, {"Content-type": "text/html; charset=utf-8"}); break;
      default: response.writeHead(404, {"Content-type": "text/html; charset=utf-8"});
    }

    response.write(
      '<!DOCTYPE html>' +
      '<html>' +
      '<head>' +
      '<title>Routage avec NodeJS</title>' +
      '<meta charset="utf-8" />' +
      '</head>' +
      '<body>' +
      '<h1>Plusieurs pages avec NodeJS</h1>' +
      '<p>La requête reçue est : <em>' + laRequete + '</em></p>'
    );

    switch(laRequete) {
      case '/': response.write("<h2>Vous êtes donc sur la page d'accueil !</h2>"); break;
      case '/about': response.write("<h2>Vous êtes donc sur la page <em>A propos</em>.</h2>"); break;
      case '/contacts': response.write("<h2>Vous êtes donc sur la page de <em>contacts</em>.</h2>"); break;
      default: response.write("<h2>Erreur 404 : page inconnue !</h2>");
    }

    response.write(
      '</body>' +
      '</html>'
    );
    response.end();
  }

  // créer un serveur qui écoute sur le port défini et traite la requête reçue
  monServeur = Http.createServer(requeteRecue).listen(port);
  console.log("Démarrage du serveur avec routage via index2.js");
}

exports.demarrer = demarrageServeur;
```

# Gestion de plusieurs URL

- On voit sur l'exemple précédent que la gestion des différentes URL est très lourde
  - Il faut prendre en compte toutes les données à placer dans les réponses HTTP
  - Il faut prévoir toutes les URL possibles
- On va devoir utiliser une autre approche pour gérer ces URL, c'est la **gestion des routes**
- On va utiliser le **framework Express pour le routage**