

Programmation Web : les bases

MPA / JavaScript / Node.js / Express / MVC BINV1051-Web1

Contents

1	Semaine 1 - Infos générales	4
1.1	Syllabus - Slides de ce cours	4
1.2	Autres fonctionnalités RevealJS	4
1.3	Objectifs du cours	4
1.4	Engagement pédagogique (1)	4
1.5	Engagement pédagogique (2)	5
1.6	Supports du cours	5
1.7	Autres supports	5
1.8	Environnement de travail	5
1.9	Installations nécessaires (1)	6
1.10	Installations nécessaires (2)	6
1.11	Remarques très importantes (1)	6
1.12	Remarques très importantes (2)	6
2	Semaine 1 - Objectifs de la semaine	7
3	Semaine 1 - Langage JavaScript	7
3.1	Types de langages de programmation	7
3.2	Instructions JavaScript (JS)	7
3.3	console.log	7
3.4	Variables	7
3.5	let	8
3.6	var	8
3.7	const	8
3.8	Variables - Remarque importante	8
3.9	Commentaires	8
3.10	Types des variables	9
3.11	Types des variables	9
3.12	Opérateurs d'égalité	9
3.13	Opérateurs logiques	9
3.14	Conditions	9
3.15	Objets	10
3.16	Format JSON	10
3.17	Boucles (1)	10
3.18	Boucles (2)	10
3.19	Boucles (3)	11
3.20	Tableaux	11
3.21	Fonctions	11
3.22	Fonctions comme valeur de variable	11
3.23	Fonctions anonymes	12
3.24	Fonctions fléchées	12
3.25	Paramètres des fonctions	12
3.26	Paramètres par valeur	12

3.27	Paramètres par référence	13
4	Semaine 1 - Node.js (MPA)	13
4.1	Introduction Node.js	13
4.2	Architecture Web (MPA)	14
4.3	Utilisation Node.js	14
4.4	Utilisation Node.js	15
4.5	Modules Node.js	15
4.6	Modules - NPM	15
4.7	Modules - Package.json	15
4.8	Modules - Nodemon	16
4.9	Modules Express	16
5	Semaine 2 - Objectifs de la semaine	16
6	Semaine 2 - Express (MPA)	16
6.1	Architecture Express	17
6.2	Facilités apportées par Express	17
6.3	Générateur Express (1)	17
6.4	Générateur Express (2)	17
6.5	Lancer une application Express	18
6.6	Serveur dynamique	18
6.7	Routing - Router	18
6.8	Routing - Paramètres	18
6.9	Routing - Réponse	19
6.10	Routing res.send / res.sendFile	19
6.11	Routing res.render	19
6.12	Serveur de fichiers statiques	19
7	Semaine 2 - Vues et moteurs de template	19
7.1	Vues et moteurs de template - Handlebars directories	19
7.2	Vues et moteurs de template - Handlebars	20
7.3	Vues et moteurs de template - Render	20
7.4	Vues et moteurs de template - Rendu conditionnel	20
7.5	Vues et moteurs de template - Boucle	21
7.6	Vues et moteurs de template - Vues partielles (partials)	21
7.7	Vues partielles (partials) - Configuration app.js (1)	21
7.8	Vues partielles (partials) - Création d'un répertoire partials (2)	21
7.9	Vues partielles (partials) - Création des vues partielles (3)	22
7.10	Vues partielles (partials) - Intégration de la vue partielle dans le master layout (4)	22
7.11	eq - enregistrement	22
7.12	eq - utilisation	22
8	Semaine 3 - Objectifs de la semaine	23
9	Semaine 3 - Middlewares	23
9.1	Middlewares	23
9.2	Middlewares	23
9.3	Utilisation d'un middleware	24
9.4	Ecrire un middleware	24
10	Semaine 3 - MVC	24
10.1	MVC c'est quoi ?	24
10.2	Architecture Express avec MVC	25
10.3	Explications MVC	25

10.4 Les vues (views)	25
10.5 Les données (models) - Exemple	25
10.6 Les données (models) - Utilisation	26
10.7 Les contrôleurs (routes)	26
10.8 Conventions d'organisation	26
10.9 Conventions d'organisation - Exemple	27
11 Semaine 4 - Objectifs de la semaine	27
12 Base de données	27
12.1 Introduction	27
12.2 SQLite	28
12.3 Installations	28
12.4 Connexion à SQLite via Datagrip (1)	28
12.5 Connexion à SQLite via Datagrip (2)	28
12.6 Connexion à SQLite via Datagrip (3)	29
12.7 Connexion à SQLite via Datagrip (4)	29
12.8 Création d'une table via Datagrip	30
12.9 Commandes SQL - Conseils et Astuces	30
12.10 Base de données & Express	30
12.11 Module better-sqlite3 - Installation	30
12.12 Création du fichier db.conf.js	31
12.13 Utilisation des prepared statements (1)	31
12.14 Utilisation des prepared statements (2)	31
12.15 Méthodes better-sqlite3	31
12.16 Modification du modèle	31
13 Semaine 5 - Objectifs de la semaine	32
14 Sessions	32
14.1 Introduction (1)	32
14.2 Introduction (2)	32
14.3 Express et Sessions (1)	32
14.4 Express et Sessions (2)	32
14.5 Express et Sessions (3)	33
14.6 Express et Sessions (4)	33
14.7 Express et Sessions (5)	33
14.8 Chiffrement de données (1)	33
14.9 Chiffrement de données (2)	33
14.10 Chiffrement de données (3)	34
15 Semaine 6 - Objectifs de la semaine (BONUS !!!)	34
15.1 Envoi et traitement de documents	34
15.2 Introduction	34
15.3 Envoi de document - formulaire	34
15.4 Envoi de document - Multer(1)	35
15.5 Envoi de document - Multer(2)	35
15.6 Envoi de document - Multer(3)	35
15.7 Envoi de document - Multer(4)	35
16 Clôture du cours JS	36
17 Rappels	36
17.1 MVC	36
17.2 Routes	36

17.3 Vues	37
17.4 Modèles	37
17.5 Examen	37
17.6 Examen	37
18 Solutions des exercices	37
18.1 Consulter les solutions	37
18.2 Récupérer les solutions via Git	37
18.3 Préliminaires	37
18.4 Se connecter à Gitlab.vinci.be	38
18.5 Définir un mot de passe sur Gitlab.vinci.be	38
18.6 Introduction - Git	38
18.7 Installation de Git	39
18.8 Utilisation Git	39
18.9 Vidéo Git	39

1 Semaine 1 - Infos générales

1.1 Syllabus - Slides de ce cours

Ces slides ont été fait avec R studio et revealjs. Revealjs est framework de présentation basé sur du JavaScript !

Hormis le côté fun de RevealJS, plusieurs fonctionnalités vous seront utiles lors des exercices :

- Vous avez un **menu** hiérarchique pour accéder directement à un chapitre (en bas à gauche)
- Vous pouvez **rechercher** un mot dans les slides (en haut à gauche)
- Les flèches de navigation vers la gauche et la droite permettent de passer d'un chapitre à l'autre
- Les flèches de navigation vers le bas et le haut permettent de naviguer à l'intérieur d'un chapitre

1.2 Autres fonctionnalités RevealJS

- la touche 'o' (overview) vous donnera une vue globale de tous les slides
- la touche 'espace' pour un parcours séquentiel (normal des slides)

1.3 Objectifs du cours

- Compétences visées
 - Savoir écrire une Multi-Page Application (MPA) utilisant Node.js/Express
 - Gérer le contexte d'un utilisateur selon le mécanisme de session
 - Respecter l'architecture MVC lors de l'élaboration d'un site Web
- Compétences prérequis
 - Base HTML et CSS
 - Base en anglais (commentaires de code en anglais)
 - Bases d'un langage de programmation (structures conditionnels, boucles, fonctions...)

1.4 Engagement pédagogique (1)

- Détails des engagements pédagogiques : Fiche UE
- Evaluation UE :
 - HTML : 10%
 - * Q1
 - * Projet
 - * pas représentable ni en juin, ni en septembre
 - JavaScript : 90%

- * Q2
- * Examen
- * Ecrit sur PC (juin et représentable en septembre)

1.5 Engagement pédagogique (2)

- Détails des engagements pédagogiques : Fiche UE
- Evaluation UE :
 - Projet Web : 100%
 - * Q2
 - * Projet
 - * pas représentable en septembre

1.6 Supports du cours

Supports principaux :

- Théorie : Ces slides
- Démonstrations : Vidéos des démonstrations
 - Attention elles ne remplacent pas les présentations faites aux cours !
- Exercices : Énoncés des exercices
- Solutions : Solutions des exercices via Git disponibles le vendredi

Tous ces supports sont accessibles et regroupés sur MooVin

1.7 Autres supports

Autres supports très utiles !

- Mozilla Network

The screenshot shows the MDN Web Docs homepage for JavaScript. The navigation bar includes links to Technologies, References & Guides, and Feedback. A search bar is highlighted with a red box. Below the navigation bar, there's a section for 'Technologies web pour développeurs' with a link to JavaScript. A banner indicates the page is translated from English. The main content area is titled 'JavaScript' and includes a 'Table of contents' with links to Tutorials, Reference, and Tools & resources. The 'Related Topics' section lists 'JavaScript' and 'Tutoriel'.

- Syllabus HTML

1.8 Environnement de travail

Vous pouvez travailler pour ce cours :

- sous le système d'exploitation de votre choix (Windows, Linux, MacOS)
- sur la machine de votre choix (machine de l'école, votre propre portable)

Sur les machines de l'école, tout est déjà installé.

Si vous souhaitez travailler sur votre propre machine, consultez le slide suivant sur les installations nécessaires

1.9 Installations nécessaires (1)

- Node.js LTS :

Il est important de bien choisir la version stable de Node.js à savoir la version LTS (Long Time Support) !!!

- Windows, MacOS : Lien Node.js LTS
- Linux (Debian, Ubuntu) :

Laissez les choix par défaut si des questions (case à cocher, ...) sont posées pendant l'installation.

Testez votre installation Node.js dans une console/terminal :

1.10 Installations nécessaires (2)

- Visual Studio Code :
 - Windows, MacOS : Lien Visual Studio Code
 - Linux (Debian, Ubuntu) :

Sous Windows, cochez “Ajouter VS code” au menu contextuel lors de l'installation

- Firefox/Chrome/Safari

Nous vous conseillons vivement d'utiliser Firefox comme navigateur ! En effet, Firefox est le plus respectueux des normes. Donc ce qui fonctionne sous Firefox a de fortes chances de fonctionner sur un autre navigateur, l'inverse n'est pas vrai.

1.11 Remarques très importantes (1)

- Cours Web 1 très important
 - Cours métier
 - Rythme soutenu (6h pendant 6 semaines)
 - Prépare au Projet Intégrateur Web
 - Les exercices de la dernière séance sont moins nombreux ou en BONUS !

Pour ces raisons, investissez-vous dans ce cours maintenant ! Terminez les exercices qui sont prévus durant la semaine. Tout retard sera difficile à rattraper car la matière avance chaque semaine.

1.12 Remarques très importantes (2)

- Comprenez ce que vous faites !
 - Copier/coller du code sans comprendre n'est pas une bonne méthode
 - Lisez le message d'erreur avant d'appeler le professeur et essayez de comprendre
 - Se baser uniquement sur les vidéos et lire les solutions n'est pas la bonne méthode

Pour ces raisons, investissez-vous dans ce cours maintenant ! Terminez les exercices qui sont prévus durant la semaine. Tout retard sera difficile à rattraper car la matière avance chaque semaine.

2 Semaine 1 - Objectifs de la semaine

- Prise en main de l'environnement de développement (Node.js / VScode)
- Savoir utiliser les éléments de base du langage JavaScript (condition, boucle, tableaux, ...)
- Comprendre l'architecture Web Client-Serveur et son application avec Node.js
- Requête HTTP avec méthode GET
- Comprendre le système de modules dans Node.js
- Savoir installer des modules dans Node.js

3 Semaine 1 - Langage JavaScript

3.1 Types de langages de programmation

- Compilé
 - Transformation du code source en binaire par un compilateur
 - Ex : .java -> .class
 - Exécution du code binaire
 - Souvent langage statiquement typé
- Interprété
 - Exécution directe du code source
 - Ex : Javascript
 - Souvent langage dynamiquement typé

3.2 Instructions JavaScript (JS)

Une instruction JS (« statement ») se compose de :

- Expressions
- Valeurs
- Opérateurs (+, -, ...)
- Mots clés (for, break, ...)
- Commentaires

Les instructions sont normalement séparées par des “;”

Les “;” ne sont pas obligatoires en JS mais vivement conseillés

3.3 console.log

console.log vous sera très utile en JS notamment pour le débogage. Elle permet d'afficher un message ou encore le contenu d'une variable dans la console.

```
console.log("coucou");  
console.log(x);  
console.log("contenu de la variable x : + " + x);
```

3.4 Variables

- Sensibles à la casse
- Type de la variable déterminé à l'exécution (langage dynamiquement typé)
- Déclarations de variables :
 - let : portée associée à un bloc
 - var : portée globale
 - const : portée associée à un bloc et immuable (la valeur ne peut pas être changée)
- Il est possible également d'utiliser directement une variable sans utiliser let, var, const !

3.5 let

```
if (true) {  
  let blockScope = "Hello";  
  console.log(blockScope); // Hello  
}  
console.log(blockScope); // Uncaught ReferenceError: blockScope is not  
                           // defined
```

3.6 var

```
if (true) {  
  var globalVar = "Hello";  
  console.log(globalVar); // Hello  
}  
console.log(globalVar); // Hello
```

3.7 const

```
if (true) {  
  const constVar = "Hello";  
  console.log(constVar); // Hello  
  const SITE_URL = "http://MyCMS.org";  
  console.log(SITE_URL); // http://MyCMS.org  
  constVar = "Hi";  
  console.log(constVar); // Uncaught TypeError: Assignment to constant  
                           // variable.  
}
```

3.8 Variables - Remarque importante

Nous avons présenté ici les différentes possibilités pour déclarer ou non les variables.

Dans le cadre de ce cours, nous nous obligerons à respecter la convention suivante :

- Toutes les variables seront déclarées !
- Les variables seront déclarées avec let ou const !
- Aucune variable sans déclaration !
- Aucune variable déclarée avec var !

3.9 Commentaires

- Commentaire sur une ligne : //
- Commentaire sur plusieurs lignes : /* */

```
function raiseAlert(message) {  
  // Single line comment  
  console.log(message);  
  /* Regular comment  
     on multiple lines  
   */  
  console.log("An alert has been raised.");  
}
```


3.10 Types des variables

Rappel : JS est un langage dynamiquement typé !

- Number (Nombre) : un seul type pour les entiers, réels, doubles.
- String (Chaîne) : comprise entre guillemets simples ou doubles.
- Bool (Booléen) : true / false
- Array (Tableau) :
- Object (Objet) :

Remarque: on peut également utiliser des string avec des backsticks (‘) permettant d’interpréter des variables \$var.

3.11 Types des variables

```
console.log(typeof 12); // number
console.log(typeof "I love JS"); // string
console.log(typeof true); // boolean
console.log(typeof undeclaredVariable); // undefined
```

3.12 Opérateurs d'égalité

- Stricte sans conversion de type : === ou !==
- Avec conversion de type : == ou !=

```
1 === 1; // true
"1" === 1; // false
1 == 1; // true
"1" == 1; // true
0 == false; // true
0 == null; // false
var object1 = { key: "value" },
object2 = { key: "value" };
object1 == object2; //false
```

Préférons !== et === quand c'est possible pour éviter les pièges du dynamiquement typé !

3.13 Opérateurs logiques

- && : ET logique
- || : OU logique
- ! : NOT logique

Plus de détails : MDN Network

3.14 Conditions

- if ... else

```
let isAuthenticated = false;
if (isAuthenticated) {
  console.log("Render the HomePage.");
  console.log("You are authenticated.");
} else {
  console.log("Render the Login Page."); // Render the Login Page.
  console.log("You are not authenticated."); // You are not authenticated.
}
```

3.15 Objets

- Utilisation des objets

```
// usage of objects
let today = new Date(); // instance of object
today.getFullYear(); // call a method(function) of the object
Date.now(); // call static method
```

- Création d'objets - notation littérale

```
// create objects without class -- LITTERAL NOTATION
// theses objects are simply a collection of properties
let vegetable = { name : "carrot", color: "orange" };
```

3.16 Format JSON

Le format JSON(JavaScript Object Notation) correspond simplement à la notation littérale (voir slide précédent) des objets en JS. Ce format est très utilisé car simple et très compréhensible.

package.json l'utilise notamment pour décrire les modules constituant un projet.

```
// JSON.stringify convertit une valeur JavaScript en une string JSON
// Pour afficher/déboguer un objet JSON
console.log(JSON.stringify(object));
```

3.17 Boucles (1)

- for ... of / for ... in

```
// for ... of pour les tableaux
let vegetables = ["onion","garlic"];
for (let vege of vegetables)
{
    console.log(vege);
}

// for ... in pour les objets JSON
const student = { name: 'Monica', age: 12 }
for (let key in student ) {
    // display the properties
    console.log("key => " + student[key]);
}

// expected output:
// name => Monica
// age => 12
```

3.18 Boucles (2)

Parcours d'un tableau d'objets ...

```
let students = [{name:'Monica', age: 12}, {name:'Sandra', age: 13} ];
for (let stud of students) {
    for(let key in stud) {
        console.log("Key : " + key + " Value : " + stud[key]);
    }
}
```

3.19 Boucles (3)

- for, while

```
// for traditionnel si besoin d'un index
for (let index = 0; index < 5; index++) {
  console.log(index); // 0 1 2 3 4
}
// while existe aussi
let vegetables = ["onion","garlic"];
let i = 0;
while(i<vegetables.length)
{
  console.log(vegetables[i]);
  i++;
}
```

Utilisons de préférence le for ... in et for ... of !

3.20 Tableaux

- []
- propriété length
- push / pop

```
let vegetables = ["onion","garlic"];
let i = 0;
while(i<vegetables.length)
{
  console.log(vegetables[i]);
  i++;
}

vegetables.push("carrot"); // array vegetables now : ["onion", "garlic", "carrot"]
// debug / show array
console.log ( JSON.stringify(vegetables));
```

Toujours créer un tableau(même vide) avec l'opérateur [] !

3.21 Fonctions

```
function welcomeMessage(message) {
  return "Message : " + message;
}

let message = welcomeMessage("Welcome to everyone!");
console.log(message); // Message : Welcome to everyone!
```

3.22 Fonctions comme valeur de variable

En JS, il est possible (et c'est utilisé fréquemment) d'assigner une fonction comme valeur de variable.

```
function welcomeMessage(message) {
  return "Message : " + message;
}
```

```
let x = welcomeMessage;
message = x("Hi");
console.log(message); // Message : Hi
```

3.23 Fonctions anonymes

En JS, les fonctions ne doivent pas avoir nécessairement un nom. Ceci est également fortement utilisé !

```
const welcome = function (message) {
  return "Message : " + message;
};

message = welcome("Hello world ; ");
console.log(message); // Message : Hello world ; )
```

3.24 Fonctions fléchées

Les fonctions fléchées sont souvent anonymes et ont une syntaxe plus courte.

```
const welcome2 = (message) => {
  return "Message : " + message;
};

message = welcome2("Hello world...");
console.log(message); // Message : Hello world...

// OTHER EXAMPLE
const higher = n => n + 1;
console.log(higher(1)); // 2
```

C'est ce type de fonction que vous rencontrerez le plus souvent et que l'on vous demandera de réaliser !

3.25 Paramètres des fonctions

- Optionnels : undefined pour un paramètre manquant
- Portée locale au sein de la fonction
- Passage d'argument par valeur, sauf pour les objets
- Passage d'un objet par référence

3.26 Paramètres par valeur

```
let myMessage="Hello";
print(myMessage);
function print(myMessage) {
  console.log(myMessage); // Hello
  myMessage = "Good bye";
}

console.log(myMessage); // Hello
```

3.27 Paramètres par référence

```
let myMessage = { content: "Hello" };
consolePrint(myMessage);

function consolePrint(myMessage) {
  console.log(myMessage.content); // Hello
  myMessage.content = "Good bye";
}

console.log(myMessage.content); // Good bye
```

4 Semaine 1 - Node.js (MPA)

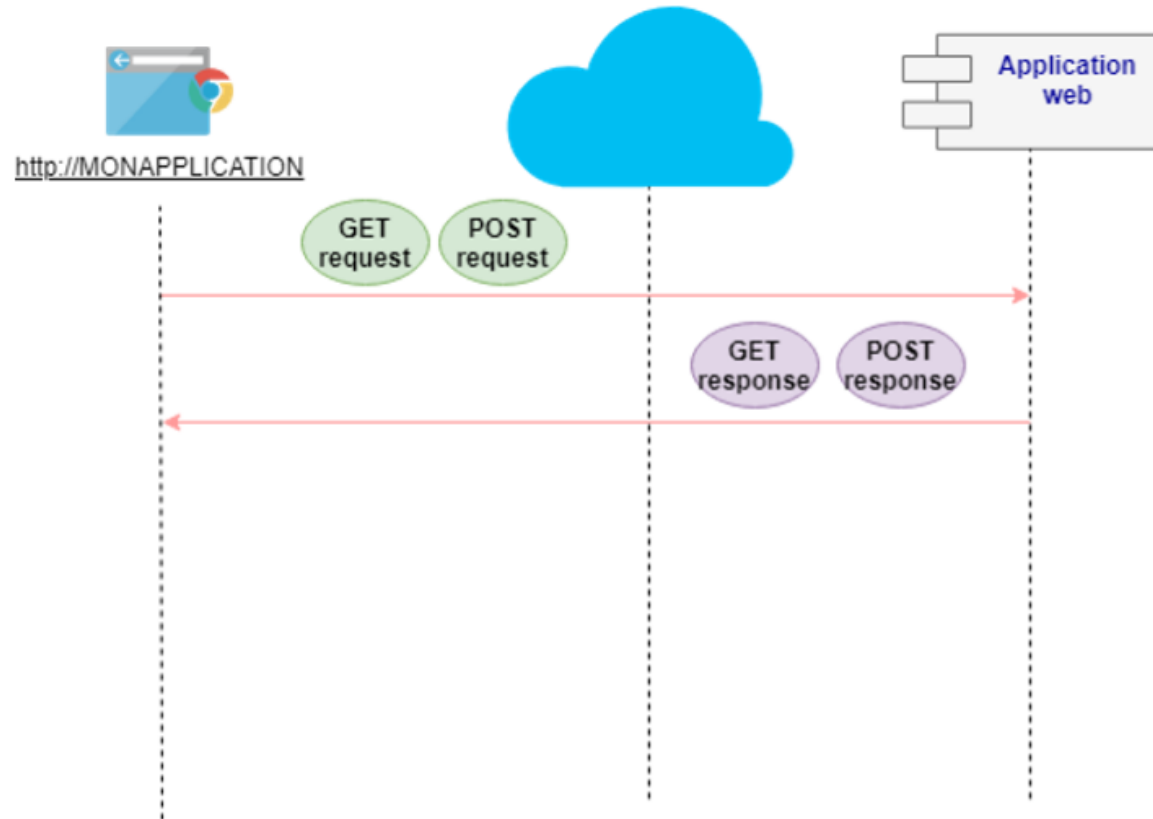
4.1 Introduction Node.js

- Node.js est une plateforme logicielle écrite en JS embarquant un serveur HTTP
- Node.js est donc capable de répondre à des requêtes HTTP de type GET et POST
- Node.js permet l'installation facile de modules via la commande :

MPA signifie Multi-Page Application par opposition aux SPA (Single Page Application) que vous verrez en Bloc 2. Une application MPA renverra ("render") à chaque requête une nouvelle page/vue.

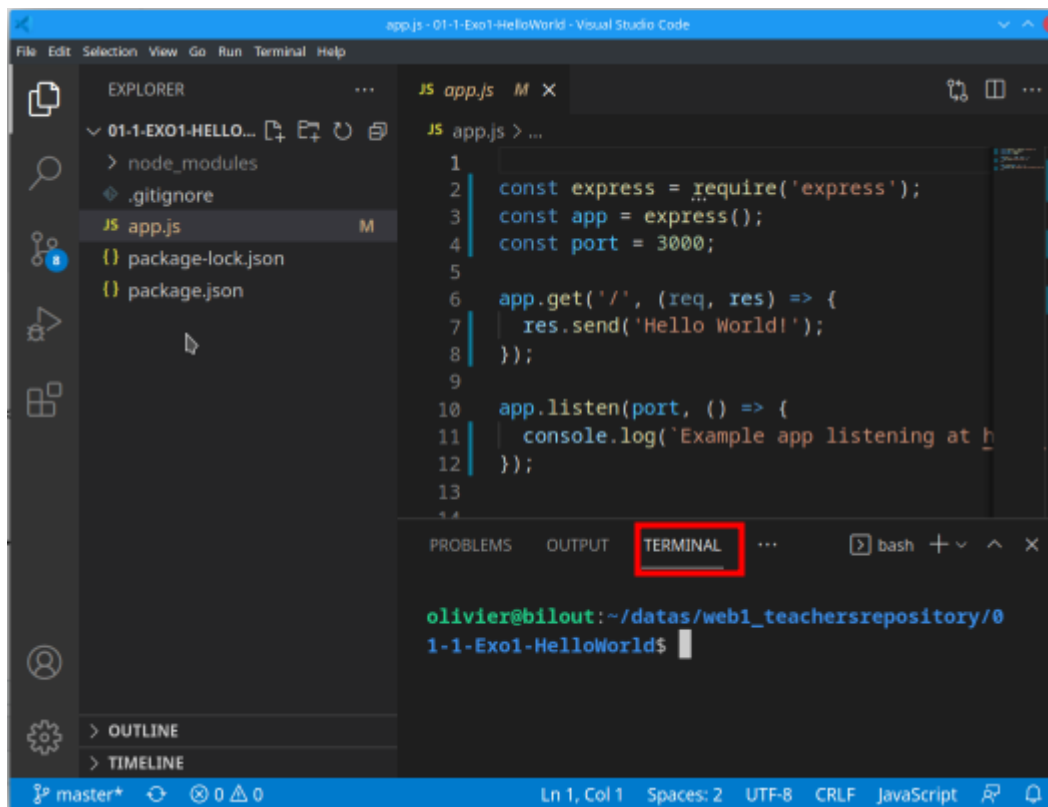
Remarque : Node.js n'est pas qu'un serveur web mais dans le cadre de ce cours nous nous limiterons à cet aspect.

4.2 Architecture Web (MPA)



4.3 Utilisation Node.js

Dans Visual Studio Code, ajoutez la fenêtre “Terminal” (Terminal -> New Terminal).



4.4 Utilisation Node.js

Lancez une application via la commande :

dans le terminal

Cliquez (CTRL + clic) ensuite sur le lien <http://localhost:3000> présent dans le terminal OU entrez cette adresse directement dans votre navigateur

Pour arrêter votre application dans Node.js -> CTRL-C

4.5 Modules Node.js

Node.js permet l'installation de modules. Cela permet d'utiliser le code d'autres développeurs ou encore de la communauté. L'objectif de ces modules est de faciliter la réutilisation du code et ainsi d'accélérer le développement d'applications.

Nous utiliserons dans ce cours quelques modules au fil des séances.

4.6 Modules - NPM

- NPM : Node Package Manager
- Commandes NPM :
 - Création d'un projet Node.js :
 - Installation d'un module :
 - Installation d'un module uniquement pour le développement :

4.7 Modules - Package.json

Tous les installations de modules du projet sont reprises dans le fichier package.json

Quand vous récupérez un projet Node.js (les solutions du prof via Git par ex.), il est important d'exécuter la commande

afin que les modules nécessaires au projet soient installés suivant son fichier package.json.

4.8 Modules - Nodemon

Le module Nodemon est très utile pour recharger automatiquement le projet dans Node.js dès qu'une modification a été faite. Sans cela, vous devez à chaque fois arrêter et relancer votre application.

- Installation :
- Lancement application :

Remarquez le -save-dev pour installer nodemon uniquement pour l'environnement de développement. C'est en effet un module uniquement utile pour les développeurs, pas pour la production.

4.9 Modules Express

Express est un framework facilitant le développement d'application sous Node.js.

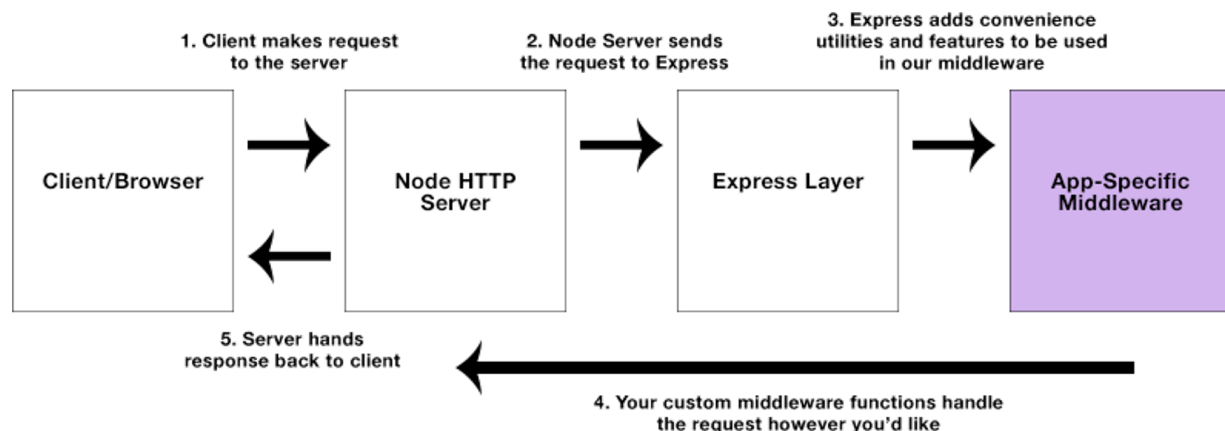
- Installation :
- Lancement application :

5 Semaine 2 - Objectifs de la semaine

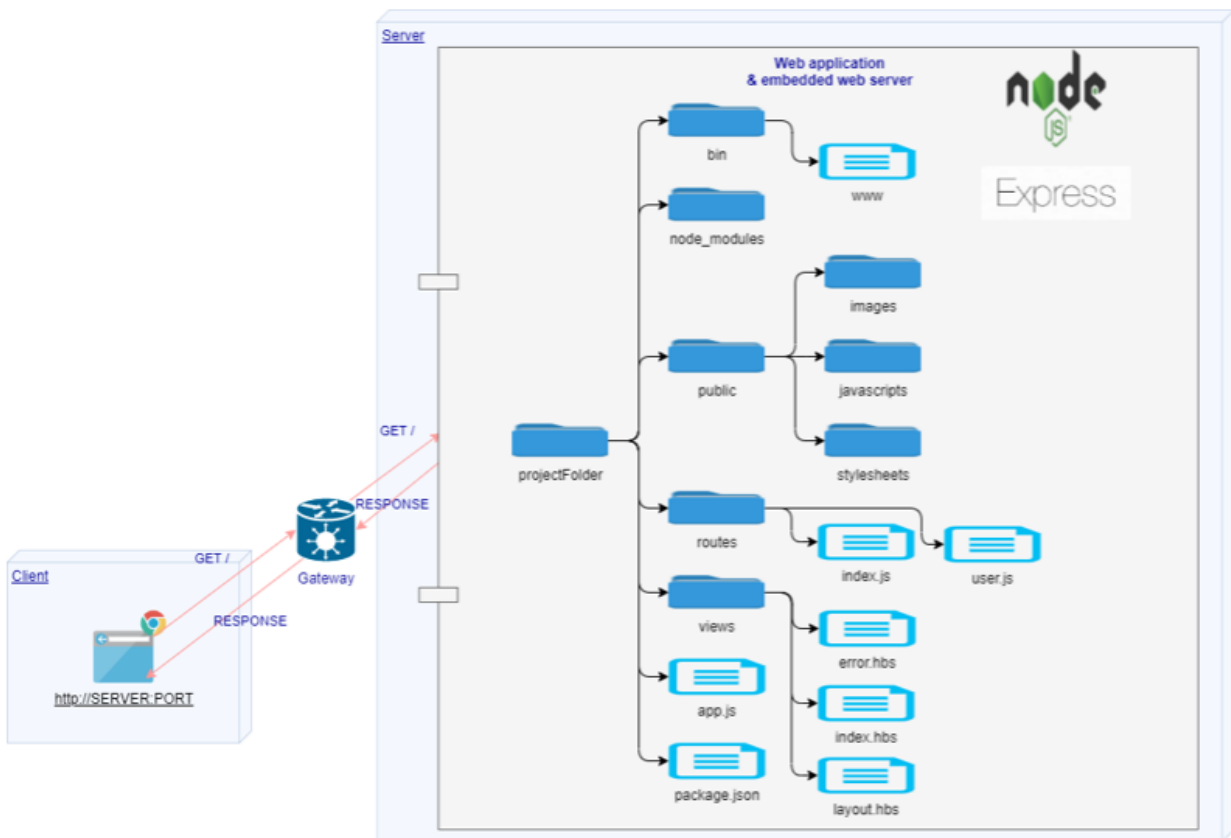
- Express
 - Générateur Express
 - Routing Express
 - Moteur de template de vue (hbs)
- Traitement de formulaire - réception de paramètres GET / POST

6 Semaine 2 - Express (MPA)

Site Express Express est un framework facilitant le développement d'application sous Node.js.



6.1 Architecture Express



6.2 Facilités apportées par Express

- Générateur Express : Configuration et démarrage d'une application
- Serveur dynamique : Gestion du serveur HTTP (on peut par exemple changer le port)
- Middlewares : Fonctions ayant accès à la requête et réponse HTTP afin de la traiter
- Routing : Express répondra aux requêtes sur les URL définies par le routing
- Vues et moteur de templating : Les réponses seront renvoyées via un moteur de template
- Serveur de fichiers statiques : Gestion des fichiers images et CSS du site/application

Tout ceci est détaillé dans les slides suivants et sera utilisé !

6.3 Générateur Express (1)

Le générateur Express facilite grandement la création d'un site/application Express.

- Créer une app avec le générateur :
- Installation des dépendances (données dans le fichier package.json) :

Remarque : npx permet d'exécuter une commande et d'installer ce qui est nécessaire pour cette commande de manière temporaire. C'est tout indiqué pour exécuter le générateur express que nous n'exécuterons qu'une seule fois par projet.

6.4 Générateur Express (2)

Décortiquons la commande

- Appel du générateur express via npx

- Utilisation d’handlebars pour les vues
- Répertoire et nom de l’application

6.5 Lancer une application Express

Regardez package.json et vous verrez que cette commande lance la commande start qui se trouve dans la rubrique scripts

node peut être remplacé par **nodemon** dans cette commande pour autant que **nodemon** soit installé !

6.6 Serveur dynamique

Le générateur Express a créé un fichier “www” dans le répertoire “bin”. Celui-ci contient le serveur HTTP.

```
const app = require('../app');
const http = require('http');
const port = normalizePort(process.env.PORT || '3000');
app.set('port', port);
const server = http.createServer(app);
server.listen(port);
```

Nous ne modifierons rien dans ce fichier hormis le port de l’application (3000 par défaut) si nécessaire.

6.7 Routing - Router

- Utilisation d’un routeur pour rassembler les routes ayant un point commun
- Définition d’un routeur :

```
// file app.js
const usersRouter = require('./routes/users');
app.use('/', indexRouter);
app.use('/users', usersRouter);
```

```
// file routes/users.js
const express = require('express');
const router = express.Router();
/* GET /users/ */
router.get('/', function(req, res, next) {
  res.send('respond with a resource');
});
module.exports = router;
```

6.8 Routing - Paramètres

- Récupération des paramètres d’un formulaire HTML :
 - GET : req.query.“input name”
 - POST : req.body.“input name”

```
<form method="post" action="/register">
  <input class="form-control" id="email" type="text" name="email" />
  <input type="submit">
</form>
```

```
/* POST new user */
router.post("/register", function (req, res, next) {
```

```
console.log("POST /register:", req.body.email) ;
res.render("index");
});
```

6.9 Routing - Réponse

Le but dans une application MPA est de renvoyer une réponse à une requête. La réponse sera renvoyée via l'une des 3 instructions suivantes :

- **res.render** : renvoyer une page HTML rendue via un moteur de template (handlebars)
- **res.send** : renvoyer une réponse (une string, un objet JSON, ...)
- **res.sendFile** : renvoyer un fichier
- **res.redirect** : rediriger vers une autre route

En gras, les actions les plus courantes pour une MPA.

6.10 Routing res.send / res.sendFile

```
app.get('/', (req, res) => {
  res.send("Bonjour");
});

app.get('/index.html', (req, res) => {
  res.sendFile(__dirname + "/index.html");
});
```

6.11 Routing res.render

res.render renverra une vue et utilisera un moteur de template (handlebars). Voir Chapitre suivant.

6.12 Serveur de fichiers statiques

- Utilisation du Middleware express.static avant les routes (app.js)

```
app.use(express.static(path.join(__dirname, "public"))); // Serve static assets
```

- Répertoire public servant les fichiers statiques
 - nécessaires à vos views : .js, .css, .jpg, .png, ...
 - appelés directement par le browser

```
// style.css must be in public/stylesheets
<link rel="stylesheet" href="/stylesheets/style.css"> // in view.layout.hbs
```

7 Semaine 2 - Vues et moteurs de template

- Utilisation d'un template engine pour créer des view dynamiques à partir de fichiers templates
- Template engines : Handlebars, Mustache, Pug, Jade...
- Pour ce cours : Handlebars (hbs)
- handlebars est un module à installer

7.1 Vues et moteurs de template - Handlebars directories

- /views/layout.hbs :
 - Master layout
 - {{{body}}} : lieu utilisé pour chaque view template

- /views/index.hbs :
 - contenu de la view « index » rendue dans le {{{body}}} du master layout
 - contenu de la vue == HTML + paramètres pour hbs entre {{param}}

7.2 Vues et moteurs de template - Handlebars

Exemple layout.hbs :

```
<!DOCTYPE html>
<html>
  <head>
    <title>{{title}}</title>
    <link rel='stylesheet' href='/stylesheets/style.css' />
  </head>
  <body>
    {{{body}}}
  </body>
</html>
```

Exemple index.hbs :

```
<section>
  <!-- h1Title -> hbs parameter see next slides -->
  <h1>{{h1Title}} </h1>
</section>
<footer>
  <p>footer site truc</p>
</footer>
```

7.3 Vues et moteurs de template - Render

- La fonction render prend en paramètre le nom de la vue et les paramètres de la vue

```
/* GET home page. */
router.get("/", function (req, res, next) {
  // first param of render -> index == index.hbs in directory views
  // second param of render -> JSON object with params
  res.render("index", {
    h1Title: "JavaScript & Node.js full course" }
  });
});
```

```
// file index.hbs in views directory
<section>
<h1> {{h1Title}} </h1>
</section>
```

7.4 Vues et moteurs de template - Rendu conditionnel

```
{{#if isAuthenticated}}
  <a class="nav-item nav-link" href="/">Home</a>
  <a class="nav-item nav-link" href="/list">List users</a>
  <a class="nav-item nav-link" href="/logout">Logout</a>
  <a class="nav-item nav-link" href="/">{{user}}</a>
{{else}}
  <a class="nav-item nav-link" href="/">Home</a>
```

```

<a class="nav-item nav-link" href="/register">Register</a>
<a class="nav-item nav-link" href="/login">Login</a>
{{/if}}

```

7.5 Vues et moteurs de template - Boucle

#each : Boucle pour rendre des éléments à partir d'un object ou array

```

<ul class="list-group list-group-horizontal-lg">
  {{#each userList}}
    <li class="list-group-item">{{this}}</li>
  {{/each}}
</ul>

```

7.6 Vues et moteurs de template - Vues partielles (partials)

Certaines portions du code (HTML) des vues est identique pour toutes les vues -> il est utile de factoriser ce code afin d'éviter la duplication de code (code redondant)

Pour ce faire, plusieurs étapes :

- 1) Configuration dans app.js des vues partielles
- 2) Création d'un répertoire "partials" sous le répertoire views
- 3) Création des vues partielles
- 4) Intégration de la vue partielle dans le master layout

Les différentes étapes sont présentées sur les slides suivants.

7.7 Vues partielles (partials) - Configuration app.js (1)

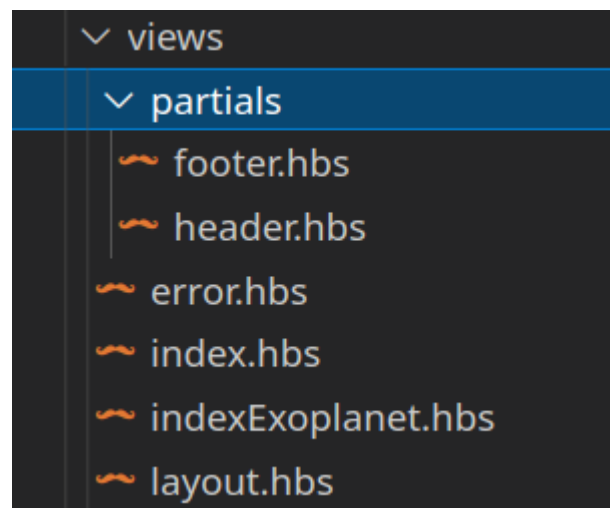
```

// register partials views
// important to do that before set engine
// BEFORE app.set('view engine', 'hbs');
var hbs = require('hbs');

hbs.registerPartials(__dirname + '/views/partials');

```

7.8 Vues partielles (partials) - Création d'un répertoire partials (2)



7.9 Vues partielles (partials) - Création des vues partielles (3)

Création d'un fichier .hbs contenant le code HTML redondant

Exemple : partialview.hbs

```
<section>
  <p> ce code HTML sera intégré pour chaque vue </p>
</section>
```

7.10 Vues partielles (partials) - Intégration de la vue partielle dans le master layout (4)

Modification du fichier layout.hbs

```
<!DOCTYPE html>
<html>
<head>
<title>{{title}}</title>
<link rel='stylesheet' href='/stylesheets/style.css' />
</head>
<body>
{{body}}
<!--pay attention of character > in front of partialview name -->
{{>partialview}}
</body>
</html>
```

7.11 eq - enregistrement

eq permet de faire un test d'égalité dans une vue handlebars. Pour utiliser eq, vous devez l'enregistrer dans app.js.

```
//IFEQ
hbs.registerHelper('eq', function (a, b) {
  if (a === b) {
    return true;
  }
  else {
    return false;
  }
});
```

7.12 eq - utilisation

```
<section>
  <p>
    {{#if (eq user.email "olivier.choquet@vinci.be") }}
      Bonjour Olivier !
    {{/if}}
  </p>
</section>
```

8 Semaine 3 - Objectifs de la semaine

- Un petit mot sur les middlewares
- Développer une application Node.js/Express respectant l'architecture MVC
 - Utilisation d'un modèle
 - Utilisation d'un contrôleur/routeur

9 Semaine 3 - Middlewares

9.1 Middlewares

Express utilise des “middlewares” pour traiter les requêtes HTTP. Chaque requête HTTP passera par plusieurs “middlewares” afin de produire une réponse HTTP. Un middleware est donc un morceau de code (une fonction) qui effectuera un traitement sur la requête et/ou réponse.

Les middlewares peuvent agir à différents niveaux.

- Application-level middleware : middleware agissant au niveau application
- Router-level middleware : middleware agissant au niveau du routing
- Error-handling middleware: middleware agissant au niveau des erreurs (HTTP)
- Built-in middleware : middleware notamment utilisé pour gérer les fichiers statiques (CSS, images)
- Third-party middleware: /

9.2 Middlewares

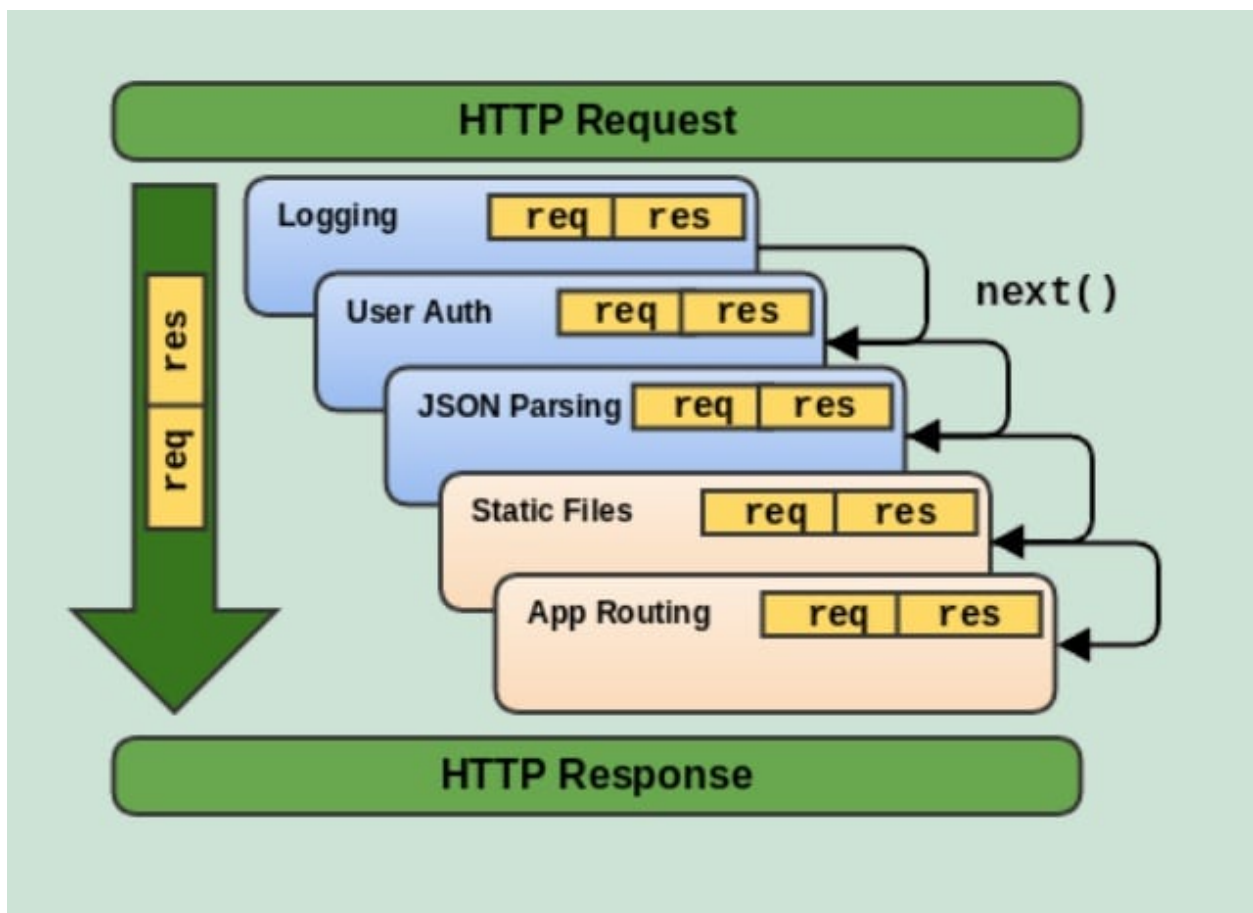


Image issue de : <https://dev.to/ghvstcode/understanding-express-middleware-a-beginners-guide-g73>

9.3 Utilisation d'un middleware

```
const express = require("express");
const app = express();
// application level middleware
app.use(function (req, res, next) {
  console.log("Time:", Date.now());
  next();
});

// router level middleware
const router = express.Router();
router.use(function (req, res, next) {
  console.log("Time:", Date.now());
  next();
});
```

9.4 Ecrire un middleware

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

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

app.listen(3000);
```

HTTP method for which the middleware function applies.

Path (route) for which the middleware function applies.

The middleware function.

Callback argument to the middleware function, called "next" by convention.

HTTP response argument to the middleware function, called "res" by convention.

HTTP request argument to the middleware function, called "req" by convention.

Image issue de : <https://expressjs.com/en/guide/writing-middleware.html>

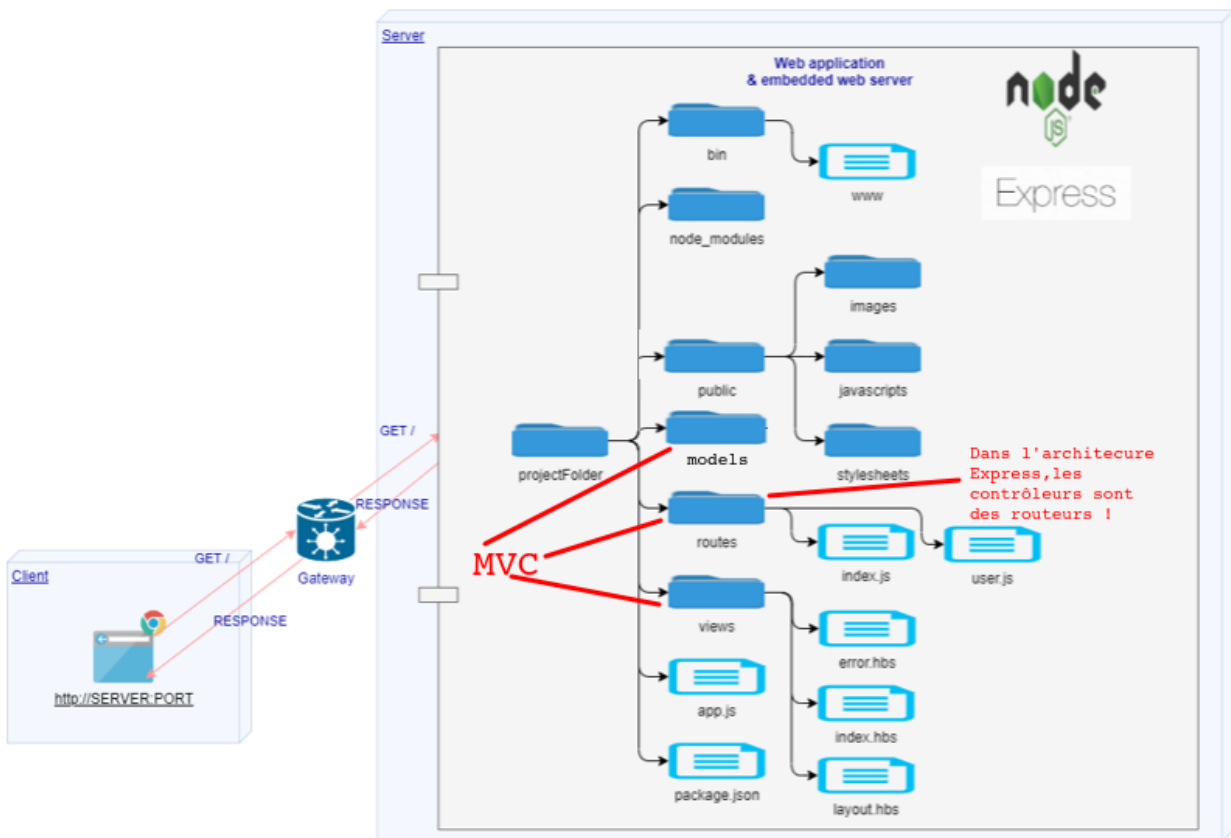
10 Semaine 3 - MVC

Model - View - Controller

10.1 MVC c'est quoi ?

- Technique de découpe du code
- Modèle == les données (persistance des données)
- Vue == affichage (HTML)
- Contrôleur == chef d'orchestre entre le modèle et la vue

10.2 Architecture Express avec MVC



10.3 Explications MVC

- **Les vues** : contiendront le code de présentation. Il s'agira dans ce cours des vues handlebars.
- **Les données** : contiendront le code d'accès aux données (SELECT, INSERT, ... dans la base de données).
- **Les contrôleurs** : contiendront le code de traitement des requêtes. Ils utiliseront le modèle et renverront vers une vue.

10.4 Les vues (views)

Concernant les vues, voir chapitre sur Vues et moteurs de templates

10.5 Les données (models) - Exemple

```
//file User.js

// the user model is simply a collection of objects users
let userList = [ { email : "laurent.leleux@vinci.be", pseudo : "lle"},
                  { email : "olivier.choquet@vinci.be", pseudo : "och"}];

// function list
// Pay attention to modules.export to give access to list function outside (in controller for example)
module.exports.list = () => {
  return userList;
}
```

```
};

module.exports.add = (data) => {
  userList.push(data);
};
```

10.6 Les données (models) - Utilisation

```
// import model file at the beginning of the file
const User = require('../models/User.js');
...

// Call list function
let lst = User.list();
...

// Call add function
User.add( {email : "stephanie.ferneeuw@vinci.be", pseudo : "sfe"});
```

10.7 Les contrôleurs (routes)

Les contrôleurs dans l'architecture Express sont des routeurs.

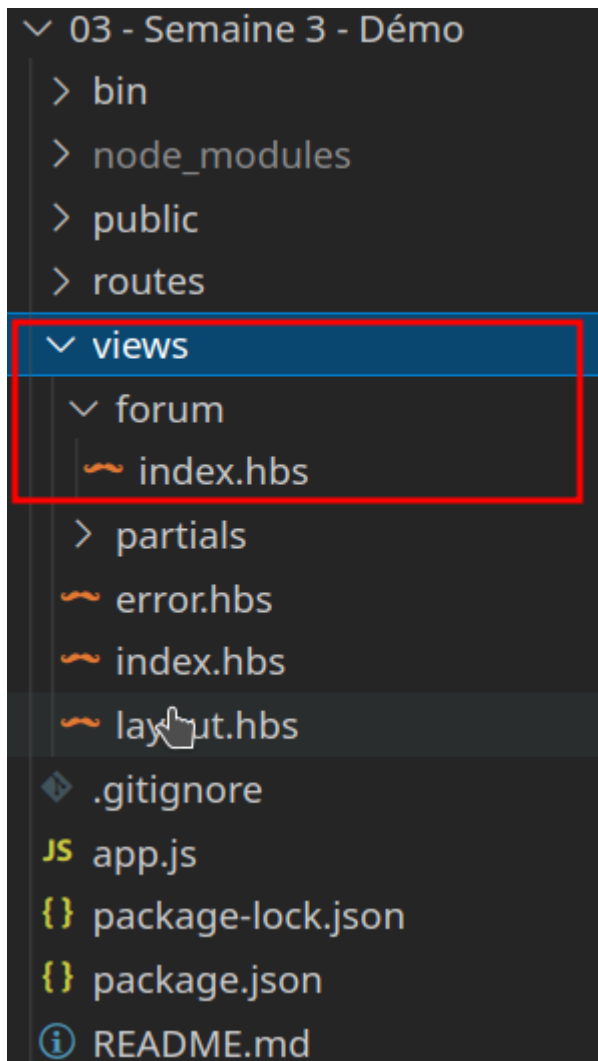
- Express a déjà créé 2 routeurs(index.js et user.js) pour vous.
- Vous pouvez créer des routeurs en regardant le code généré par Express.

Rappel Routeur Express

10.8 Conventions d'organisation

- Les fichiers modèles commencent par une majuscule et sont au singulier.
- Les fichiers routeurs portent le nom de la route. (Ex : users.js -> /users)
- Les vues spécifiques à un contrôleur peuvent être regroupées dans un dossier portant le nom du contrôleur

10.9 Conventions d'organisation - Exemple



11 Semaine 4 - Objectifs de la semaine

- Manipuler une base de données SQL via le logiciel Datagrip
- Utiliser une base de données SQL dans une application Node.js/Express

12 Base de données

12.1 Introduction

La majorité des sites Web stockent leurs informations dans une base de données. Divers types et variétés de base de données existent.

Dans le cadre de cours, nous utiliserons une base de données SQL à savoir **SQLite**.

Les slides suivants montrent comment créer une base de données SQLite et la gérer avec le logiciel Datagrip. Ensuite, nous verrons comment utiliser cette base de données avec une application Express.

12.2 SQLite

- Base de données légère (pas d'installation, ni login, ni mot passe, ...)
- Convient bien pour le développement
- Pas de gestion des utilisateurs, des accès concurrents, ...
- Autres SGBDR (Système de Gestion de Base de Données Relationnel) : MariaDB, PostgreSQL, SQL Server

12.3 Installations

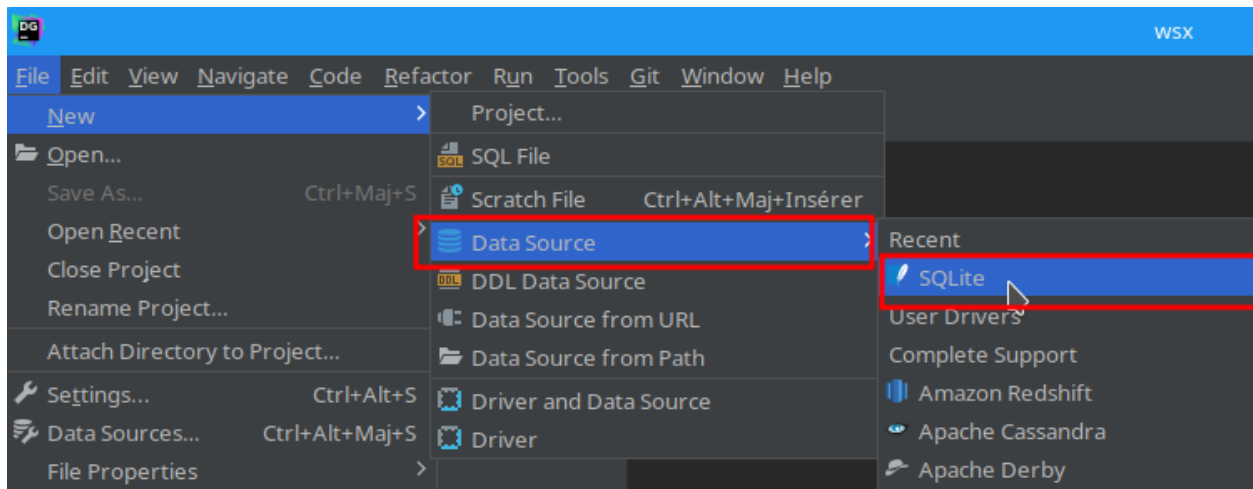
Si vous travaillez sur les machines de l'école, tout est déjà installé !

Si vous travaillez sur votre propre machine, vous devrez simplement installer Datagrip :

- Datagrip : <https://www.jetbrains.com/datagrip/download/>
 - Sous Ubuntu :

12.4 Connexion à SQLite via Datagrip (1)

- Lancez Datagrip et créez un nouveau projet.
- Créez un DataSource SQLite

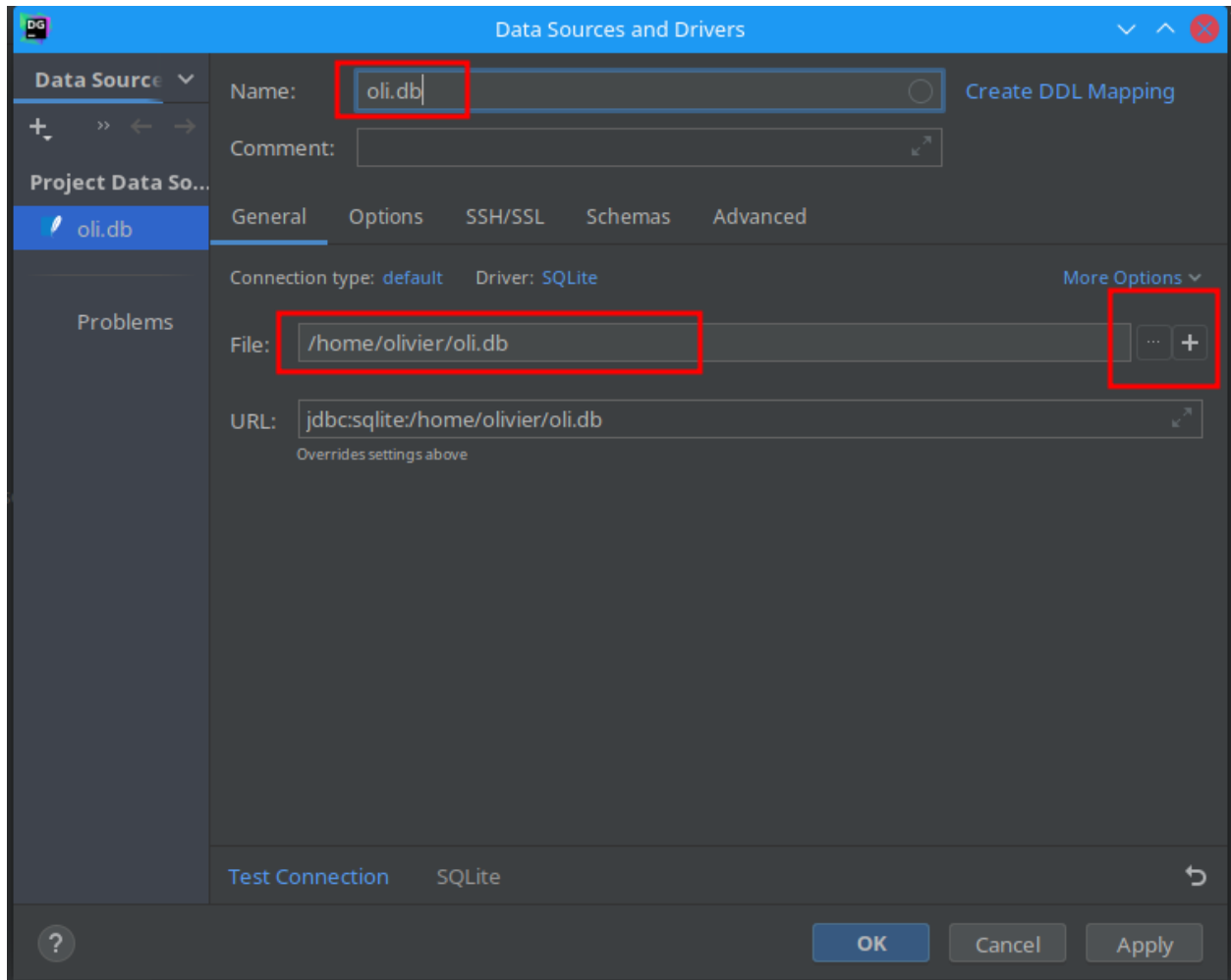


12.5 Connexion à SQLite via Datagrip (2)

Où enregistrer votre Base de données SQLite ?

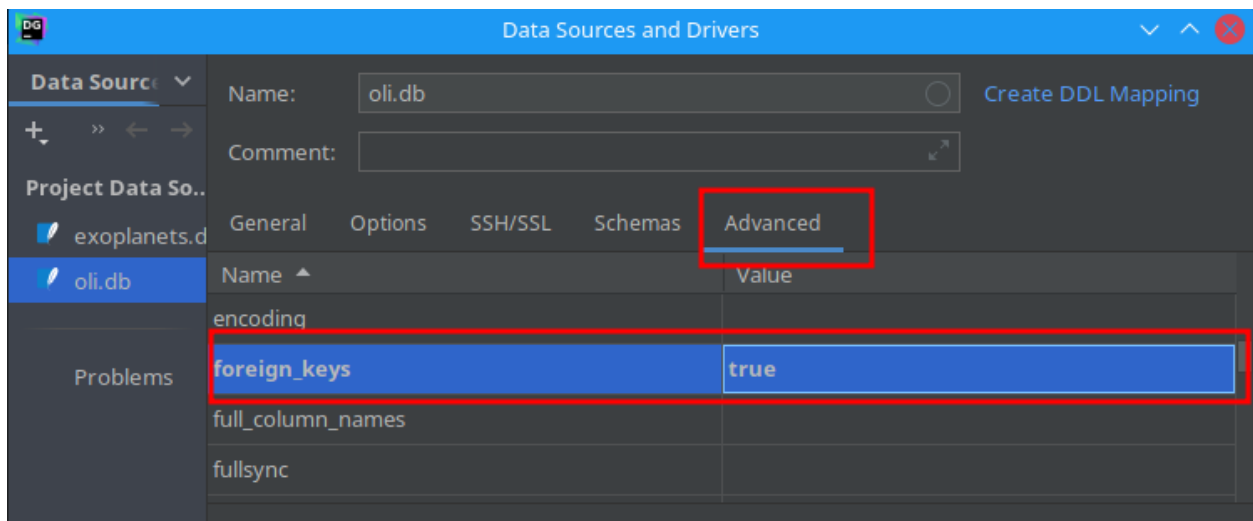
- Enregistrer votre fichier DB sur le disque U si vous travaillez sur les PC de l'école
- Enregistrer votre fichier DB où vous le souhaitez si vous travaillez sur votre propre machine
- Ce n'est pas une bonne pratique d'enregistrer votre fichier DB dans votre projet car elle sera utilisée par plusieurs exercices/projets

12.6 Connexion à SQLite via Datagrip (3)



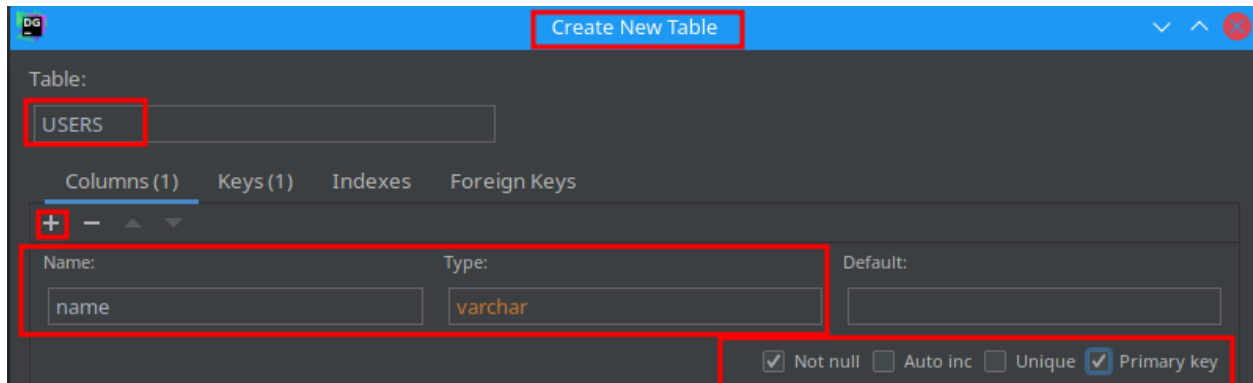
12.7 Connexion à SQLite via Datagrip (4)

Dans l'onglet avancé -> ajoutez la gestion des contraintes des clés étrangères !



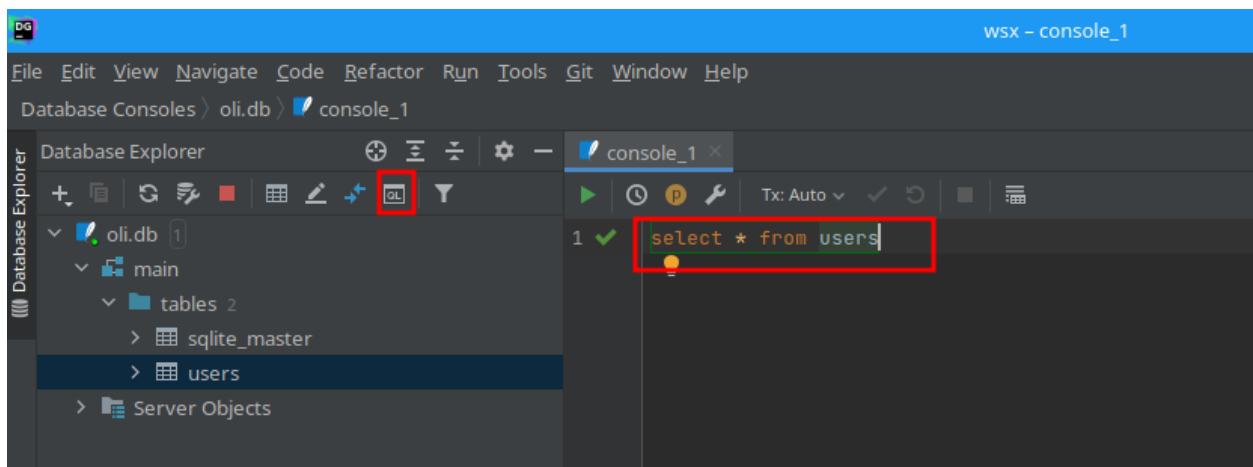
12.8 Création d'une table via Datagrip

Clic droit sur Table ...



12.9 Commandes SQL - Conseils et Astuces

- Reportez-vous à votre cours SQL pour l'utilisation des commandes SQL !
- N'oubliez pas les clés primaires, clés étrangères et NOT NULL !
- Testez vos commandes SQL dans Datagrip avant de les intégrer dans votre code !
 - Clic droit sur une table -> Edit Data
 - Clic sur l'outil query pour créer et tester une nouvelle requête



12.10 Base de données & Express

Dans une application Express MVC, la base de données SQL fera partie du modèle.

Pour réaliser ceci, les points suivants sont nécessaires :

- Installation d'un module pour communiquer avec la DB (module better-sqlite3)
- Création d'un fichier db_conf.js contenant les paramètres de connexion à la DB
- Utilisation des Prepared Statements

Ces points sont détaillés ci-après.

12.11 Module better-sqlite3 - Installation

- Utilisation du module PG pour dialoguer avec SQLite
 - Il est nécessaire d'installer ce module pour votre application

12.12 Création du fichier db.conf.js

```
// file db_conf.js in models  
  
const db = require('better-sqlite3')('/home/olivier/users.db', { verbose: console.log });  
  
module.exports = db;
```

12.13 Utilisation des prepared statements (1)

Pour envoyer une requête SQL à un SGBDR(Système de Gestion de Base de Données Relationnel), on envoie une string avec les ordres SQL.

Mais quid des paramètres ?

On peut concaténer les paramètres dans la string mais cela pose plusieurs soucis :

- les simples guillemets (') sont utilisés en SQL, le développeur doit donc être prudent lors de ces concaténations
- un problème de sécurité courant sont les injections SQL (le hacker remplit les champs d'un formulaire avec du code SQL)

12.14 Utilisation des prepared statements (2)

Pour éviter ces soucis, la majorité des langages de programmation propose des Prepared Statements.

Dans ce cas, le query se fera en 2 temps :

- l'écriture du query, les paramètres étant simplement identifiés par un ?
- l'attribution des paramètres dans un second temps (les paramètres sont convertis en string pour éviter toute injection SQL)

12.15 Méthodes better-sqlite3

- all(params) : renvoie tous les enregistrements / s'utilise avec SELECT
- get(params) : renvoie le premier enregistrement / s'utilise avec SELECT
- run(params) : exécute la commande SQL / s'utiliser avec INSERT, UPDATE,

Dans les 2 cas, on fera un prepare :

```
const stmt = db.prepare("SELECT * FROM USERS");  
// all() -> return alls rows like  
// [ {name:'user1', pseudo:'oli'}, {name:'user2', pseudo:'stef'}]  
return stmt.all();  
  
const stmt = db.prepare('INSERT INTO USERS (name, pseudo) VALUES (?, ?)');  
//run -> return infos about changes made  
const info = stmt.run(data.name, data.pseudo);
```

12.16 Modification du modèle

```
// file User.js  
const db = require('../models/db_conf');  
  
module.exports.list = () => {  
  // use of prepared statement  
  const stmt = db.prepare("SELECT * FROM USERS");
```

```

    // all() -> return alls rows like [ {name:'user1', pseudo:'oli'}, {name:'user2', pseudo:'stef'}]
    return stmt.all();
};

module.exports.save = (data) => {
    // use of prepared statement with parameters
    const stmt = db.prepare('INSERT INTO USERS (name, pseudo) VALUES (?, ?)');
    const info = stmt.run(data.name, data.pseudo);
    console.log("users model save" + info.changes);
};

```

13 Semaine 5 - Objectifs de la semaine

- Utiliser des sessions
- Crypter des informations (password)

14 Sessions

14.1 Introduction (1)

Le protocole HTTP est sans état (stateless) ce qui veut dire que chaque requête est indépendante l'une de l'autre.

Comment faire alors pour conserver le fait qu'un utilisateur soit connecté et puisse accéder à différentes pages sur un site MPA sans devoir lui redemander son login/mdp à chaque page?

Les sessions résolvent ce souci. Le serveur Web peut créer une session pour chaque nouvel utilisateur. L'id de cette session est ensuite stocké chez le client sous forme de cookie et réenvoyé avec chaque requête ce qui permet de suivre l'utilisateur.

14.2 Introduction (2)

- Un nouvel utilisateur en terme de session correspond à un même navigateur, système d'exploitation et même adresse IP.
- Les sessions constituent un moyen de conserver/transmettre des variables sur toutes les pages de votre site.

14.3 Express et Sessions (1)

Express dispose d'un module pour utiliser les sessions.

14.4 Express et Sessions (2)

Configurer le module express-sessions : **création d'une variable globale session**

```

//app.js
const createError = require('http-errors');
const express = require('express');
const path = require('path');
const cookieParser = require('cookie-parser');
const logger = require('morgan');

// Use of sessions
const session = require('express-session');

```



```
const indexRouter = require('./routes/index');
const usersRouter = require('./routes/users');

const app = express();
...
```

14.5 Express et Sessions (3)

Configurer le module express-sessions : définir quelques paramètres obligatoires

- secret : ce secret sera utilisé pour chiffrer le cookie de session stocké chez le client
- resave : forcer la sauvegarde de la session pour chaque nouvelle requête
- saveUninitialized : forcer la sauvegarde des nouvelles sessions qui n'ont pas encore été modifiées

Laissez ces paramètres comme dans le code ci-dessous

```
...
app.use(cookieParser());
app.use(express.static(path.join(__dirname, 'public')));

// use of sessions
app.use(session({secret: "Your secret key", resave: false, saveUninitialized:false}));

app.use('/', indexRouter);
app.use('/users', usersRouter);
...
```

14.6 Express et Sessions (4)

Créer/récupérer une variable dans la session : req.session.“variablename”

```
router.post('/login', (req, res, next) => {
  console.log("LOGIN");
  req.session.login = true;
});
```

14.7 Express et Sessions (5)

Détruire la session

```
router.post('/logout', (req, res, next) => {
  console.log("LOGOUT");
  req.session.destroy();
});
```

14.8 Chiffrement de données (1)

Installation module bcrypt :

14.9 Chiffrement de données (2)

Création d'une empreinte d'une donnée :

```
// import module bcrypt
const bcrypt = require('bcrypt');
// mandatory variable for bcrypt
const saltRounds = 10;

.....

const encryptedData = bcrypt.hashSync("ma donnée à protéger/encrypter", saltRounds);
```

14.10 Chiffrement de données (3)

Pour vérifier une donnée chiffrée (mot de passe par ex.), bcrypt dispose d'une méthode de comparaison du hash avec un texte en clair.

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

...

if (bcrypt.compareSync(dataInClear, hash))
{
  console.log("dataInClear == hash, OK c'est bon");
} else {
  console.log ("dataInClear != hash, KO");
}
```

15 Semaine 6 - Objectifs de la semaine (BONUS !!!)

Ne faites ceci que si vous avez terminé tous les exercices précédents !

- Envoi et traitement de documents

15.1 Envoi et traitement de documents

15.2 Introduction

Il est fort fréquent qu'un site Web permette l'envoi de documents(image, PDF, ...) . Il existe en HTML l'input suivant pour réaliser ceci :

.

Cependant il sera nécessaire de modifier le formulaire en ajoutant le support pour l'envoi de document (enctype) :

Le document sera envoyé sur le serveur dans un dossier à définir sur le serveur et son nom sera aléatoire. Cependant nous aurons accès à des informations sur le fichier tel que son nom original.

15.3 Envoi de document - formulaire

```
<form method="post" action="/users/add" enctype="multipart/form-data">
  <div>
    <label>Pseudo Utilisateur : </label>
    <input type="text" name="pseudo" />
  </div>
  <div>
    <label>Avatar Utilisateur :</label>
```

```

        <input type="file" name="avatar" />
      </div>
      <div>
        <input type="submit">
      </div>
    </form>

```

15.4 Envoi de document - Multer(1)

Nous utiliserons le module Multer pour le traitement des documents.

Installation de Multer :

15.5 Envoi de document - Multer(2)

Pour utiliser Multer, il faut lui indiquer où stocker les fichiers envoyés par l'utilisateur.

Si ces documents sont des images et que vous souhaitez pouvoir les afficher par la suite, utilisez le dossier public/images !

```

const multer = require('multer');
const upload = multer({ dest: 'public/images/' });
...

```

15.6 Envoi de document - Multer(3)

```

const fs = require('fs');
...
// avatar is the name of the input file !!!
// call of the single function of multer -> upload.single()
router.post('/add', upload.single('avatar'),
  (req, res, next) => {
    // req.file.originalname is the original name of the file on the user computer
    // req.file.path is the path of the file stored by Multer -> name is random
    console.log("MULTER INFOS : " + req.file.originalname + " " + req.file.path);

    const tempPath = req.file.path;
    const definitivePath = "public/images/" + req.file.originalname;
    const pathToDB = "images/" + req.file.originalname;

    // rename file with a proper name
    fs.renameSync(tempPath, definitivePath);
    User.save({ pseudo: req.body.pseudo, image: pathToDB });
    res.redirect('/users');
  });

```

15.7 Envoi de document - Multer(4)

Résumé sur les chemins :

- **tempPath** : chemin vers le fichier stocké par multer nommé aléatoirement
 - Ex: public/images/373042dca6068c8db14c3abc3d4bb40b
- **definitivePath** : construction d'un chemin de fichier unique et parlant
 - Ex: public/images/8-3-2022-16-22-22-masuperimage.png

- **pathToDB** : construction d'un chemin permettant l'affichage direct du document du même type que les chemins utilisés jusqu'ici (Ex : "images/logo.png")
 - Ex: images/8-3-2022-16-22-22-masuperimage.png

16 Clôture du cours JS

17 Rappels

17.1 MVC

Le MVC en Node.js/Express

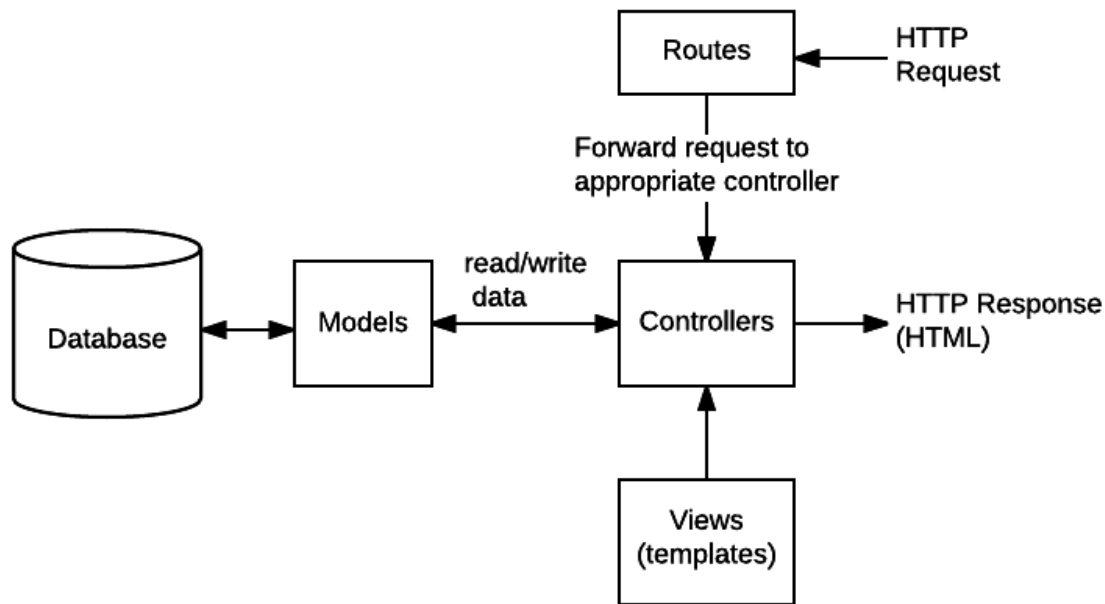


Image issue de : https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs/routes

17.2 Routes

Décortiquons une route :

`http://localhost:3000/exoplanets/search?uniqueNameExoplanet=TRA`

Partie de l'URL	Rôle
<code>http://localhost:3000</code>	Route de base du site
<code>/exoplanets</code>	Router ou Controller
<code>/search</code>	Appel d'une fonction sur le router
<code>?uniqueNameExoplanet=TRA</code>	Paramètres dans l'URL car méthode GET

Remarques :

- Les routes commencent toujours par un /
- Un router est activé via une instruction `app.use` dans `app.js`

- Un router se termine toujours par l’affichage d’une vue(render) ou la redirection(redirect) vers une route

17.3 Vues

Les vues sont des fichiers .hbs stockés dans le répertoire views.

Les vues sont affichées uniquement avec la méthode render !

La méthode render prend comme premier paramètre un fichier .hbs.

Il faut donner le chemin vers ce fichier hbs à partir du répertoire views !

Vous pouvez donner des paramètres à une vue hbs ({{login}} par ex.)

17.4 Modèles

Un modèle représente une table en base de données et contient des fonctions permettant d’interroger(SQL) le modèle.

17.5 Examen

- UE Web 1
 - 10% HTML
 - 90% JS (Examen)

Attention le projet Web a son UE propre -> Note séparée projet et Web !

17.6 Examen

Déroulement :

- sur machine de l’école (Windows)
- matière examen -> liste des exercices primordiaux
- syllabus Web1 disponible + 3 pages (recto-verso) notes manuscrites ou imprimées
- examen partira de la solution “05-3-Exo1- Sessions - Revue Inscription”
- on demandera l’ajout d’une ou plusieurs fonctionnalités

18 Solutions des exercices

Chaque vendredi, les solutions des exercices de la semaine écoulée seront disponibles.

18.1 Consulter les solutions

Consultation en ligne : Dépôt Git

18.2 Récupérer les solutions via Git

Outil pour le développement

18.3 Préliminaires

Avant de pouvoir récupérer les solutions sur votre machine, il faut :

- se connecter à Gitlab
- définir un mot de passe

Voir en image la procédure avec les slides ci-après.

18.4 Se connecter à Gitlab.vinci.be

GitLab @ Vinci



A complete DevOps platform

GitLab is a single application for the entire software development lifecycle. From project planning and source code management to CI/CD, monitoring, and security.

This is a self-managed instance of GitLab.

Username or email

Password

☐ Remember me

[Forgot your password?](#)

Sign in

Sign in with

 Azure AD


☐ Remember me

18.5 Définir un mot de passe sur Gitlab.vinci.be

GitLab

Menu

Search GitLab



User Settings

Profile

Account

Applications

Chat

Access Tokens

Emails

Password

Notifications

SSH Keys

GPG Keys

Preferences

Active Sessions

User Settings > Edit Password

Search settings

Password

After a successful password update, you will be redirected to the login page where you can log in with your new password.

Mettez ici le même mot de passe que pour votre compte @student.vinci.be !!!

Change your password or recover your current one

Current password

You must provide your current password in order to change it.

New password

Password confirmation

Save password

[I forgot my password](#)

18.6 Introduction - Git

Git est un système de contrôle de version distribué gratuit et open source conçu pour tout gérer, des petits aux très grands projets, avec rapidité et efficacité.

OK mais en clair ?

Git permet notamment de :

- partager facilement du code entre plusieurs développeurs
- intégrer les changements de plusieurs développeurs sur un même projet

Dans le cadre de ce cours, l'utilisation de Git sera limitée à la récupération des solutions des exercices.

18.7 Installation de Git

Si vous travaillez sur une machine de l'école, Git est déjà installé.

Par contre, si vous travaillez sur votre propre machine il faudra installer Git.

- Installation Git

18.8 Utilisation Git

Pour récupérer les solutions :

- Faites une copie du dépôt des solutions (à faire une seule fois)
- Récupérer les solutions (à faire chaque vendredi)
- Installer les dépendances des solutions

18.9 Vidéo Git

Video Solutions via Git