



Formation Développer des applications Web Full JavaScript

Romain Bohdanowicz

Twitter : @bioub

<http://formation.tech/>





- Introduction
- Rappels JavaScript
- Node.js
- Modules JavaScript
- Gestion de dépendances
- NoSQL
- Express
- Grunt
- Framework HTML/CSS/JS
- Angular
- MEAN



Introduction



- **Romain Bohdanowicz**
Ingénieur EFREI 2008, spécialité en Ingénierie Logicielle
- **Expérience**
Formateur/Développeur Freelance depuis 2006
Plus de 5000 heures de formation
- **Langages**
Expert : HTML / CSS / JavaScript / PHP / Java
Notions : C / C++ / Objective-C / C# / Python / Bash / Batch
- **Certifications**
PHP 5 / PHP 5.3 / PHP 5.5 / Zend Framework 1
- **Et vous ?**
Langages ? Expérience ? Utilité de cette formation ?



Rappels JavaScript



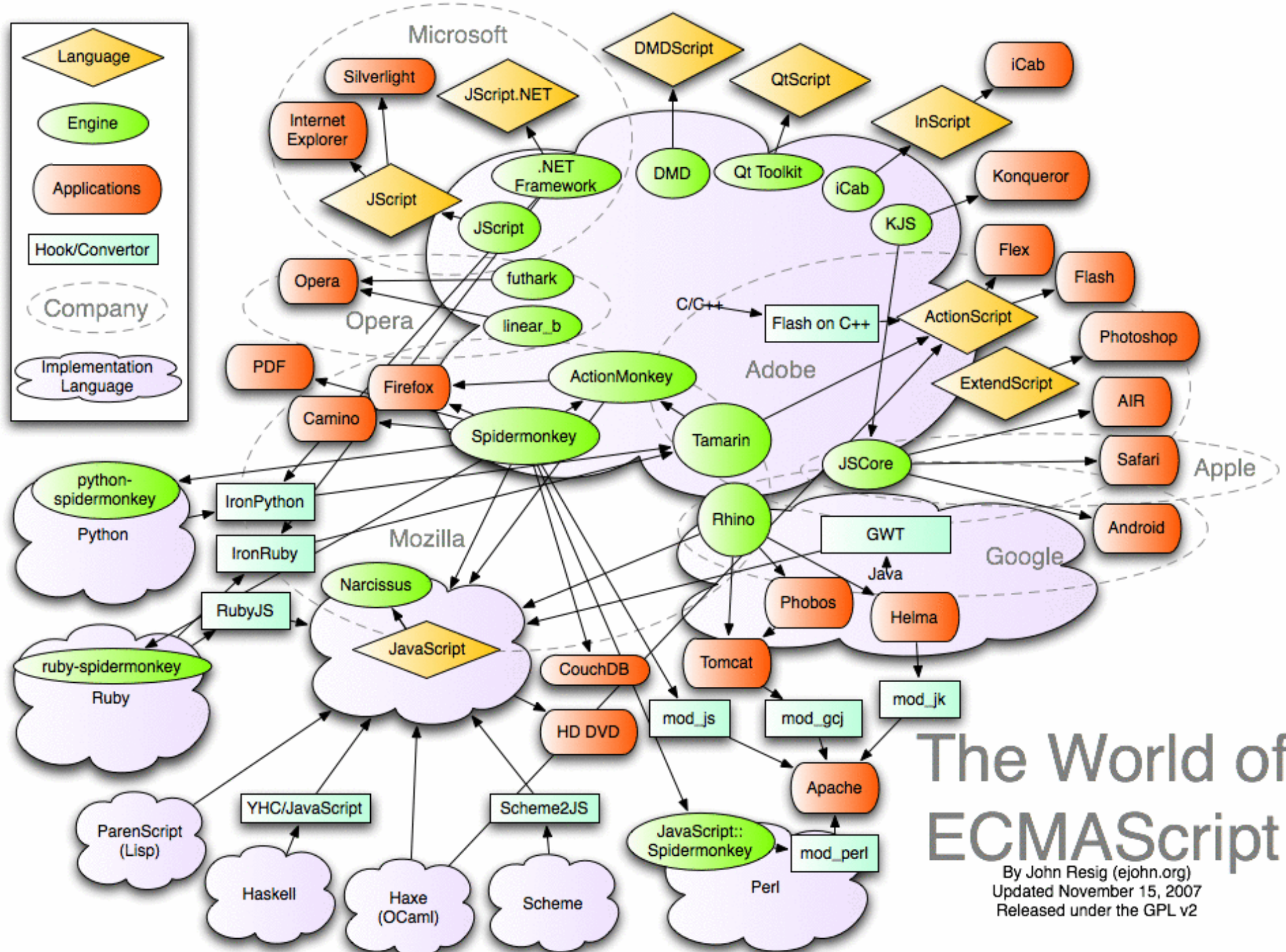
- Langage créé en 1995 par Netscape
- Objectif : permettre le développement de scripts légers qui s'exécutent une fois le chargement de la page terminé
- Exemples de l'époque :
 - Valider un formulaire
 - Permettre du rollover
- Netscape ayant un partenariat avec Sun, nomme le langage JavaScript pour qu'il soit vu comme le petit frère de Java (dont il est inspiré syntaxiquement)
- Fin 1995 Microsoft introduit JScript dans Internet Explorer
- Une norme se crée en 1997 : ECMAScript

Rappels JavaScript



- JavaScript est une implémentation de la norme ECMAScript 262
- La norme la plus récente est ECMAScript 2016, aussi appelée ECMAScript 7 ou ES7 (juin 2016)
<http://www.ecma-international.org/ecma-262/7.0/>
- Le langage a très fortement évolué avec ECMAScript 2015 / ECMAScript 6 / ES6 (juin 2015)
<http://www.ecma-international.org/ecma-262/6.0/>
- Navigateur actuels (octobre 2016) ~ 90% d'ES6
Node.js 6 ~ 90% d'ES6
Internet Explorer 11 ~ 10% d'ES6
- Pour connaître la compatibilité des moteurs JS :
<http://kangax.github.io/compat-table/>
- Pour découvrir les nouveautés d'ECMAScript 2015 / ES6
<http://es6-features.org/>
- Pour développer dès aujourd'hui en ES6 ou ES7 et exécuter le code sur des moteurs plus anciens on peut utiliser des :
 - Compilateurs ou transpileurs : Babel, Traceur, TypeScript... Transforment la syntaxe ES6 en ES5
 - Bibliothèques de polyfills : core-js, es6-shim, es7-shim... Recréent les méthodes manquante en JS

Rappels JavaScript





- La syntaxe s'inspire de Java (lui même inspiré de C)
- JavaScript est sensible à la casse, attention aux majuscules/minuscules !
- Les instructions se termine au choix par un point-virgule ou un retour à la ligne (même si les conventions incitent à la l'utilisation du point-virgule)
- 3 types de commentaires
 - `//` le commentaire s'arrête à la fin de la ligne
 - `/*` commentaire ouvrant/fermant `*/`
 - `/**` Documentation `*/`



- ▶ Les identifiants (noms de variables, de fonctions) doivent respecter les règles suivantes :
 - Contenir uniquement lettres Unicode, Chiffres, \$ et _
 - Ne commencent pas par un chiffre
- ▶ Bonnes pratiques :
 - ne pas utiliser d'accents (passage d'un éditeur à un autre)
 - séparer les mots dans l'identifiant par des majuscules (camelCase), ou des _ (snake_case)
 - les identifiants qui commencent par des \$ ou _ sont utilisés par certaines conventions
- ▶ Exemples :
 - Valides
i, maVariable, \$div, v1, prénom
 - Invalides
1var, ma-variable



▶ Attentions aux mots clés (ES7) :

- | | | | | |
|------------|-----------|--------------|----------|---------|
| ▶ break | ▶ default | ▶ for | ▶ return | ▶ var |
| ▶ case | ▶ delete | ▶ function | ▶ super | ▶ void |
| ▶ catch | ▶ do | ▶ if | ▶ switch | ▶ while |
| ▶ class | ▶ else | ▶ import | ▶ this | ▶ with |
| ▶ const | ▶ export | ▶ in | ▶ throw | ▶ yield |
| ▶ continue | ▶ extends | ▶ instanceof | ▶ try | |
| ▶ debugger | ▶ finally | ▶ new | ▶ typeof | |

▶ Mots clés en mode strict :

- | | |
|-------|----------|
| ▶ let | ▶ static |
|-------|----------|

▶ Réservés pour une utilisation future :

- | | |
|--------|---------|
| ▶ enum | ▶ await |
|--------|---------|

▶ Réservés pour une utilisation future (mode strict)

- | | | |
|--------------|-----------|-------------|
| ▶ implements | ▶ package | ▶ protected |
| ▶ interface | ▶ private | ▶ public |



▶ Voici les types primitifs en JS

- number
- boolean
- string

▶ Les types complexes

- object
- array

▶ Les types spéciaux

- undefined
- null



- ▶ Contrairement à certains langages, on ne déclare pas le type au moment de la création

```
var firstName = 'Romain';  
var lastName = 'Bohdanowicz';
```

- ▶ Historiquement on pouvait déclarer une variable sans le mot-clé var, qui devenait alors globale (mauvaise pratique que l'on peut limiter depuis ECMAScript 5)



► Affectation

Nom	Opérateur composé	Signification
Affectation	$x = y$	$x = y$
Affectation après addition	$x += y$	$x = x + y$
Affectation après soustraction	$x -= y$	$x = x - y$
Affectation après multiplication	$x *= y$	$x = x * y$
Affectation après division	$x /= y$	$x = x / y$
Affectation du reste	$x \% = y$	$x = x \% y$
Affectation après exponentiation	$x ** = y$	$x = x ** y$



► Comparaison

Opérateur	Description	Exemples qui renvoient true
Égalité (==)	Renvoie true si les opérandes sont égaux après conversion en valeurs de mêmes types.	<pre>3 == var1 "3" == var1 3 == '3'</pre>
Inégalité (!=)	Renvoie true si les opérandes sont différents.	<pre>var1 != 4 var2 != "3"</pre>
Égalité stricte (===)	Renvoie true si les opérandes sont égaux et de même type. Voir Object.is() et égalité de type en JavaScript .	<pre>3 === var1</pre>
Inégalité stricte (!==)	Renvoie true si les opérandes ne sont pas égaux ou s'ils ne sont pas de même type.	<pre>var1 !== "3" 3 !== '3'</pre>
Supériorité stricte (>)	Renvoie true si l'opérande gauche est supérieur (strictement) à l'opérande droit.	<pre>var2 > var1 "12" > 2</pre>
Supériorité ou égalité (>=)	Renvoie true si l'opérande gauche est supérieur ou égal à l'opérande droit.	<pre>var2 >= var1 var1 >= 3</pre>
Infériorité stricte (<)	Renvoie true si l'opérande gauche est inférieur (strictement) à l'opérande droit.	<pre>var1 < var2 "2" < "12"</pre>
Infériorité ou égalité (<=)	Renvoie true si l'opérande gauche est inférieur ou égal à l'opérande droit.	<pre>var1 <= var2 var2 <= 5</pre>



► Arithmétiques

En plus des opérations arithmétiques standards (+, -, *, /), JavaScript fournit également d'autres opérateurs arithmétiques, listés dans le tableau qui suit :

Opérateur	Description	Exemple
Reste (%)	Opérateur binaire. Renvoie le reste entier de la division entre les deux opérandes.	12 % 5 renvoie 2.
Incrément (++)	Opérateur unaire. Ajoute un à son opérande. S'il est utilisé en préfixe (++x), il renvoie la valeur de l'opérande après avoir ajouté un, s'il est utilisé comme opérateur de suffixe (x++), il renvoie la valeur de	Si x vaut 3, ++x incrémente x à 4 et renvoie 4, x++ renvoie 3 et seulement
Décrément (--)	Opérateur unaire. Il soustrait un à son opérande. Il fonctionne de manière analogue à l'opérateur d'incrément.	Si x vaut 3, --x décrémente x à 2 puis renvoie 2, x-- renvoie 3
Négation unaire (-)	Opérateur unaire. Renvoie la valeur opposée de l'opérande.	Si x vaut 3, alors -x renvoie -3.
Plus unaire (+)	Opérateur unaire. Si l'opérande n'est pas un nombre, il tente de le convertir en une valeur numérique.	+"3" renvoie 3. +true renvoie 1.
Opérateur d'exponentiation (**) (puissance)	Calcule un nombre (base) élevé à une puissance donnée (soit basepuissance)	2 ** 3 renvoie 8 10 ** -1 renvoie -1



► Logiques

Opérateur	Usage	Description
ET logique (&&)	expr1 && expr2	Renvoie expr1 s'il peut être converti à false, sinon renvoie expr2. Dans le cas où on utilise des opérandes booléens, && renvoie true si les deux opérandes valent true, false sinon.
OU logique ()	expr1 expr2	Renvoie expr1 s'il peut être converti à true, sinon renvoie expr2. Dans le cas où on utilise des opérandes booléens, renvoie true si l'un des opérandes vaut true, si les deux valent false, il renvoie false.
NON logique (!)	!expr	Renvoie false si son unique opérande peut être converti en true, sinon il renvoie true.



- ▶ Concaténation

```
console.log("ma " + "chaîne"); // affichera "ma chaîne" dans la console
```

- ▶ Ternaire

```
var statut = (âge >= 18) ? "adulte" : "mineur";
```

- ▶ Voir aussi

Opérateurs binaires, in, instanceof, delete, typeof...



► Priorités

Type d'opérateur	Opérateurs individuels
membre	<code>.</code> <code>[]</code>
appel/création d'instance	<code>()</code> <code>new</code>
négation/incrémentation	<code>!</code> <code>~</code> <code>-</code> <code>+</code> <code>++</code> <code>--</code> <code>typeof</code> <code>void</code> <code>delete</code>
multiplication/division	<code>*</code> <code>/</code> <code>%</code>
addition/soustraction	<code>+</code> <code>-</code>
décalage binaire	<code><<</code> <code>>></code> <code>>>></code>
relationnel	<code><</code> <code><=</code> <code>></code> <code>>=</code> <code>in</code> <code>instanceof</code>
égalité	<code>==</code> <code>!=</code> <code>===</code> <code>!==</code>
ET binaire	<code>&</code>
OU exclusif binaire	<code>^</code>
OU binaire	<code> </code>
ET logique	<code>&&</code>
OU logique	<code> </code>
conditionnel	<code>?:</code>
assignation	<code>=</code> <code>+=</code> <code>-=</code> <code>*=</code> <code>/=</code> <code>%=</code> <code><<=</code> <code>>>=</code> <code>>>>=</code> <code>&=</code> <code>^=</code> <code> =</code>
virgule	<code>,</code>



► Conversions implicites

```
console.log(3 * '3'); // 9  
console.log(3 + '3'); // '33'  
console.log(!'texte'); // false
```

► Conversions explicites

```
console.log(parseInt('33.33')); // 33  
console.log(parseFloat('33.33')); // 33.33  
console.log(Number('33.33')); // 33.33  
console.log(Boolean('texte')); // true  
console.log(String(33.33)); // '33.33'
```



- ▶ **Standard Built-in Objects**

Les objets prédéfinies par le langage, voir la doc de Mozilla pour une liste exhaustive

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects

- ▶ **Ex : String, Array, Date, Math, RegExp, JSON...**



► if ... else

```
if (typeof console === 'object') {  
    console.log('console est un objet');  
}  
else {  
    // oups  
}
```

► switch

```
switch (alea) {  
    case 0:  
        console.log('zéro');  
        break;  
    case 1:  
    case 2:  
    case 3:  
        console.log('un, deux ou trois');  
        break;  
    default:  
        console.log('entre quatre et neuf');  
}
```




► while

```
var alea = Math.floor(Math.random() * 10);

while (alea > 0) {
    console.log(alea);
    alea = parseInt(alea / 2);
}
```

► do ... while

```
do {
    var alea = Math.floor(Math.random() * 10);
}
while (alea % 2 === 1);

console.log(alea);
```

► for

```
for (var i=0; i<10; i++) {
    aleas.push(Math.floor(Math.random() * 10));
}

console.log(aleas.join(', ')); // 6, 6, 7, 0, 5, 1, 2, 8, 9, 7
```



▸ JavaScript est très consommateur de fonctions

- réutilisation
- factorisation
- récursivité
- fonction de rappel / écouteur
- closure
- module

▸ Syntaxe

- mot clé function
- nom de fonction (optionnel)
- arguments (optionnels)
- déclaration
- paramètre de retour (optionnel)



► Function declaration

```
function addition(nb1, nb2) {  
    return Number(nb1) + Number(nb2);  
}  
  
console.log(addition(2, 3)); // 5
```

► Anonymous function expression

```
var addition = function (nb1, nb2) {  
    return Number(nb1) + Number(nb2);  
}  
  
console.log(addition(2, 3)); // 5
```

► Named function expression

```
var addition = function addition(nb1, nb2) {  
    return Number(nb1) + Number(nb2);  
}  
  
console.log(addition(2, 3)); // 5
```



► Variadic function

```
function addition() {  
    var sum = 0;  
  
    for (var i=0; i<arguments.length; i++) {  
        sum += arguments[i];  
    }  
  
    return sum;  
}  
  
console.log(addition(1, 2, 3)); // 6
```



► Fonctions imbriquées

```
function addSquares(a, b) {  
    function square(x) {  
        return x * x;  
    }  
    return square(a) + square(b);  
}  
  
console.log(addSquares(2, 3)); // 13
```

► Portées

```
function addSquares(a) {  
    function square() {  
        return a * a;  
    }  
    return square() + square();  
}  
  
console.log(addSquares(2)); // 8
```



► Closure

```
function powClosure(exp) {  
    return function(base) {  
        var result = 1;  
        for(var i=0; i<exp; i++) {  
            result *= base;  
        }  
        return result;  
    }  
}  
  
var square = powClosure(2);  
var cube = powClosure(3);  
  
console.log(square(4)); // 16  
console.log(cube(4)); // 64
```



► Sans Closure

```
// affiche 4 4 4 dans 1 seconde  
for(var i = 1; i <= 3; i++) {  
    setTimeout(function() {  
        console.log(i);  
    }, 1000);  
}
```

► Avec Closure

```
// affiche 1 2 3 dans 1 seconde  
for(var i = 1; i <= 3; i++) {  
    setTimeout(function(rememberI) {  
        return function() {  
            console.log(rememberI);  
        }  
    })(i), 1000);  
}
```




► Callback

```
function onSeCallDansTroisSecondes(callback) {  
    setTimeout(callback, 3000);  
}  
  
function ditBonjour() {  
    console.log('Bonjour');  
}  
  
onSeCallDansTroisSecondes(ditBonjour); // affichera Bonjour dans 3 sec
```

- Les callbacks (fonctions de rappel) permettent la mise en place de programmation asynchrone (via une boucle d'événements)



► Object function

```
var contact = {  
  prenom: 'Romain',  
  nom: 'Bohdanowicz'  
};  
  
function saluer(prenom) {  
  return 'Bonjour ' + prenom + ' je suis ' + this.prenom;  
}  
  
console.log(saluer('Eric')); // Bonjour Eric je suis undefined  
console.log(saluer.call(contact, 'Eric')); // Bonjour Eric je suis Romain  
console.log(saluer.apply(contact, ['Eric'])); // Bonjour Eric je suis Romain  
console.log(saluer.bind(contact)('Eric')); // Bonjour Eric je suis Romain
```



- ▶ Module
Immediately Invoked Function Expression (IIFE)

```
(function($, global) {  
  'use strict';  
  
  function MonBouton(options) {  
    this.options = options || {};  
    this.value = options.value || 'Valider';  
  }  
  
  MonBouton.prototype.creer = function(container) {  
    $(container).append('<button>'+this.value+'</button>')  
  };  
  
  global.MonBouton = MonBouton;  
})(jQuery, window);
```



- Création d'un objet avec l'objet global Object :

```
var instructor = new Object();
instructor.firstName = 'Romain';
instructor.hello = function() {
    return 'Hello my name is ' + this.firstName;
};

console.log(instructor.hello()); // Hello my name is Romain
```

- Création d'un objet avec la syntaxe Object Literal (recommandé) :

```
var instructor = {
    firstName: 'Romain',
    hello: function() {
        return 'Hello my name is ' + this.firstName;
    }
};

console.log(instructor.hello()); // Hello my name is Romain
```



► Accès aux objets possible :

- Avec l'opérateur .
- Avec des crochets

```
var instructor = {  
  firstName: 'Romain',  
  hello: function() {  
    return 'Hello my name is ' + this.firstName;  
  }  
};  
  
instructor.firstName = 'Jean';  
console.log(instructor.hello()); // Hello my name is Jean  
  
instructor['firstName'] = 'Eric';  
console.log(instructor['hello']()); // Hello my name is Eric
```



- En utilisant une fonction constructeur :

```
function Person(firstName) {  
    this.firstName = firstName;  
  
    this.hello = function () {  
        return 'Hello my name is ' + this.firstName;  
    }  
}  
  
var instructor = new Person('Romain');  
  
console.log(instructor.hello()); // Hello my name is Romain  
console.log(typeof instructor); // object  
console.log(instructor instanceof Object); // true  
console.log(instructor instanceof Person); // true  
  
for (var prop in instructor) {  
    if (instructor.hasOwnProperty(prop)) {  
        console.log(prop); // firstName puis hello  
    }  
}
```



- En utilisant une fonction constructeur + son prototype :

```
function Person(firstName) {  
    this.firstName = firstName;  
}  
  
Person.prototype.hello = function () {  
    return 'Hello my name is ' + this.firstName;  
};  
  
var instructor = new Person('Romain');  
  
console.log(instructor.hello()); // Hello my name is Romain  
console.log(typeof instructor); // object  
console.log(instructor instanceof Object); // true  
console.log(instructor instanceof Person); // true  
  
for (var prop in instructor) {  
    if (instructor.hasOwnProperty(prop)) {  
        console.log(prop); // firstName  
    }  
}
```




- En utilisant une fonction constructeur + son prototype :

```
function Person(firstName) {
    this.firstName = firstName;
}

Person.prototype.hello = function () {
    return 'Hello my name is ' + this.firstName;
};

function Instructor(firstName, speciality) {
    Person.apply(this, arguments); // héritage des propriétés de l'objet (recopie dynamique)
    this.speciality = speciality;
}

Instructor.prototype = new Person; // héritage du type

// Redéfinition de méthode
Instructor.prototype.hello = function() {
    // Appel de la méthode parent
    return Person.prototype.hello.call(this) + ', my speciality is ' + this.speciality;
};

var instructor = new Instructor('Romain', 'JavaScript');

console.log(instructor.hello()); // Hello my name is Romain
console.log(typeof instructor); // object
console.log(instructor instanceof Object); // true
console.log(instructor instanceof Person); // true
console.log(instructor instanceof Instructor); // true

for (var prop in instructor) {
    if (instructor.hasOwnProperty(prop)) {
        console.log(prop); // firstName, speciality
    }
}
```



▸ Définition Wikipedia :

La programmation orientée prototype est une forme de programmation orientée objet sans classe, basée sur la notion de prototype. Un prototype est un objet à partir duquel on crée de nouveaux objets.

▸ Comparaison des modèles à classes et à prototypes

- Objets à classes :
 - Une classe définie par son code source est statique ;
 - Elle représente une définition abstraite de l'objet ;
 - Tout objet est instance d'une classe ;
 - L'héritage se situe au niveau des classes.
- Objets à prototypes :
 - Un prototype défini par son code source est mutable ;
 - Il est lui-même un objet au même titre que les autres ;
 - Il a donc une existence physique en mémoire ;
 - Il peut être modifié, appelé ;
 - Il est obligatoirement nommé ;
 - Un prototype peut être vu comme un exemplaire modèle d'une famille d'objet ;
 - Un objet hérite des propriétés (valeurs et méthodes) de son prototype ;



- ▶ En ECMAScript/JavaScript, l'écriture `foo.bar` s'interprète de la façon suivante :
 1. Le nom `foo` est recherché dans la liste des identifiants déclarés dans le contexte d'appel de fonction courant (déclarés par `var`, ou comme paramètre de la fonction) ;
 2. S'il n'est pas trouvé :
 - Continuer la recherche (retour à l'étape 1) dans le contexte de niveau supérieur (s'il existe),
 - Sinon, le contexte global est atteint, et la recherche se termine par une erreur de référence.
 3. Si la valeur associée à `foo` n'est pas un objet, il n'a pas de propriétés ; la recherche se termine par une erreur de référence.
 4. La propriété `bar` est d'abord recherchée dans l'objet lui-même ;
 5. Si la propriété ne s'y trouve pas :
 - Continuer la recherche (retour à l'étape 4) dans le prototype de cet objet (s'il existe) ;
 - Si l'objet n'a pas de prototype associé, la valeur indéfinie (`undefined`) est retournée ;
 6. Sinon, la propriété a été trouvée et sa référence est retournée.

Rappels JavaScript



- JSON, JavaScript Object Notation est la sérialisation d'un objet JavaScript
- Seuls les types string, number, boolean, array et regexp sont sérialisable, les fonctions et prototype sont perdus
- On se sert de ce format pour échanger des données entre 2 programmes ou pour créer de la config
- Le format résultant est proche de Object Literal, les clés sont obligatoirement entre guillemets "", un code JSON est une syntaxe Object Literal valide

```
{  
  "name": "My Address Book",  
  "contacts": [  
    {  
      "firstName": "Bill",  
      "lastName": "Gates"  
    },  
    {  
      "firstName": "Steve",  
      "lastName": "Jobs"  
    }  
  ]  
}
```



- ▶ JavaScript depuis EcmaScript 5 fourni l'objet global JSON qui contient 2 méthodes, `parse` (désérialiser) et `stringify` (sérialiser)

```
var contact = {  
  prenom: 'Romain',  
  nom: 'Bohdanowicz'  
};  
  
var json = JSON.stringify(contact);  
console.log(json); // {"prenom":"Romain","nom":"Bohdanowicz"}  
  
var object = JSON.parse(json);  
console.log(object.prenom); // Romain
```



► Limite des mauvaises pratiques historiques de JavaScript

```
(function($, global) {  
    'use strict';  
  
    function MonBouton(options) {  
        this.options = options || {};  
        this.value = options.value || 'Valider';  
    }  
  
    MonBouton.prototype.creer = function(container) {  
        $(container).append('<button>'+this.value+'</button>')  
    };  
  
    global.MonBouton = MonBouton;  
})(jQuery, window);
```



- Les moteurs JS sont par défaut mono-thread et mono-processus, ils ne peuvent donc exécuter qu'une seule tâche à la fois.
- Une boucle d'événements permet de passer d'un callback à l'autre de manière très performante, ex : traiter le clic d'un bouton entre 2 étapes d'une animation
- JavaScript est non-bloquant, il stocke les événements à traiter

```
setTimeout(function() {  
    console.log('1 fois dans 3 secondes');  
}, 3000);  
  
var intervalId = setInterval(function() {  
    console.log('toutes les 2 secondes');  
}, 2000);  
  
setTimeout(function() {  
    console.log('Bye bye');  
    clearInterval(intervalId);  
}, 15000);
```



- ▶ **Document Object Model**

API créé par Netscape en même temps que le JavaScript qui permet la manipulation du HTML en mémoire sous la forme d'un arbre

- ▶ **W3C**

Une norme existe depuis 1998, aujourd'hui dans sa 4e édition

DOM Level 4 : <http://www.w3.org/TR/dom/>

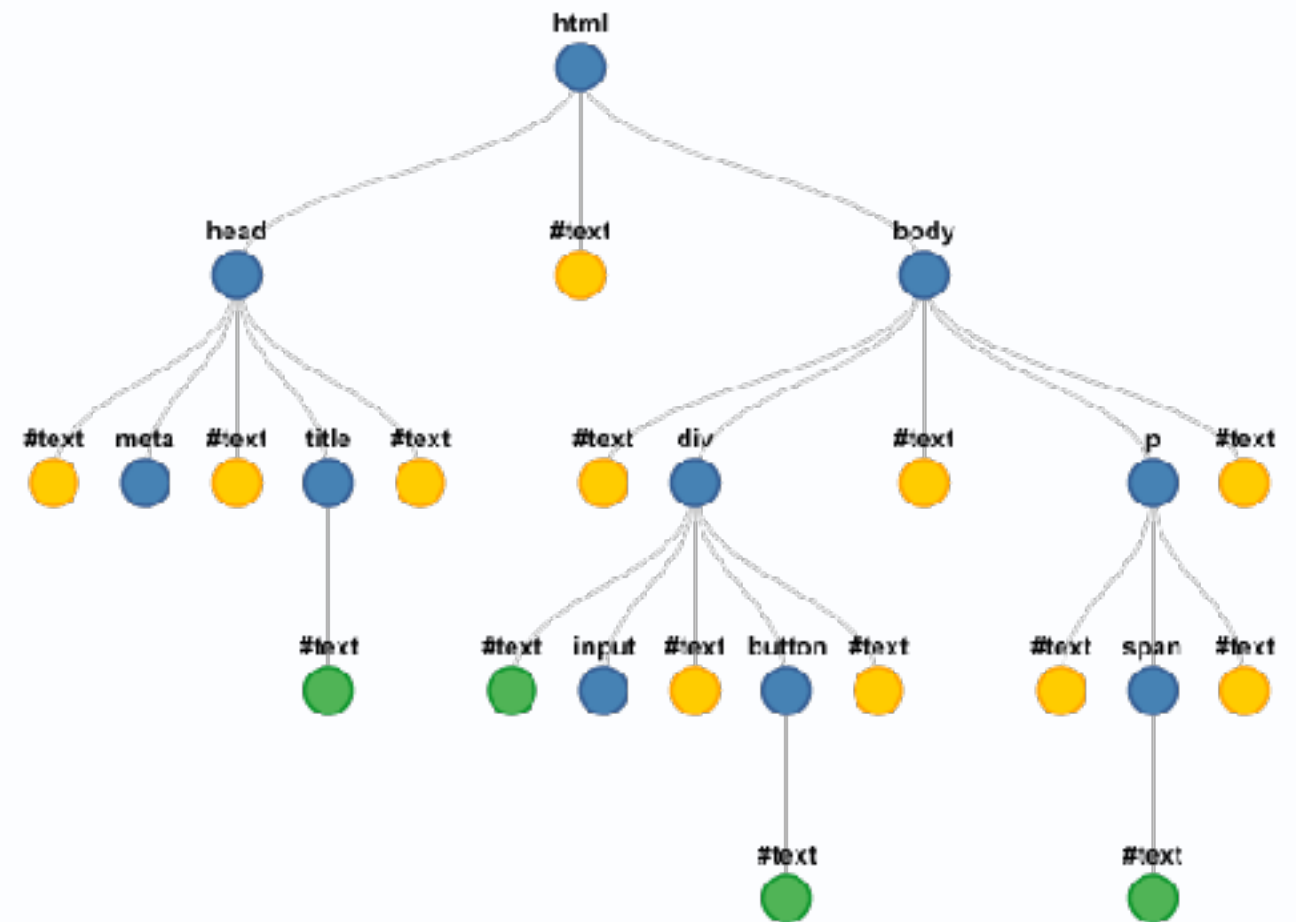
DOM Level 3 Events : <http://www.w3.org/TR/DOM-Level-3-Events/>

HTML5 : <http://www.w3.org/TR/html5/>



- Représentation sous la forme d'un arbre

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8">
  <title>Helloworld</title>
</head>
<body>
  <div>
    Prénom :
    <input name="prenom" id="prenom">
    <button id="clique">Hello</button>
  </div>
  <p>
    Bonjour <span id="result"> </span>
  </p>
</body>
</html>
```





► DOM 1 (années 1990)

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8">
  <title>Helloworld</title>
  <script>
    function helloworld() {
      var input = document.getElementsByName('prenom')[0];
      var span = document.getElementsByTagName('span')[0];

      span.firstChild.nodeValue = input.value;
    }
  </script>
</head>
<body>
<div>
  Prénom :
  <input type="text" name="prenom" id="prenom">
</div>
<p>
  Bonjour <span id="result"> </span>
</p>
</body>
</html>
```



► DOM 2 / DOM 3 (années 2000)

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8">
  <title>Helloworld</title>
  <script>
    window.addEventListener('load', function() {
      var input = document.getElementById('prenom');
      var span = document.getElementById('result');

      input.addEventListener('input', function () {
        span.textContent = input.value;
      });
    });
  </script>
</head>
<body>
<div>
  Prénom :
  <input type="text" name="prenom" id="prenom">
</div>
<p>
  Bonjour <span id="result"> </span>
</p>
</body>
</html>
```



► DOM 4 (années 2010)

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8">
  <title>Helloworld</title>
</head>
<body>
<div>
  Prénom :
  <input type="text" name="prenom" id="prenom">
</div>
<p>
  Bonjour <span id="result"> </span>
</p>
<script>
  (function() {
    'use strict';

    var input = document.querySelector('input[type=text][name=prenom]');
    var span = document.querySelector('body > p span#result');

    input.addEventListener('input', function () {
      span.innerHTML = '<b>' + input.value + '</b>';
    });
  })();
</script>
</body>
</html>
```



Gestion de dépendances



- ▶ Gestionnaire de dépendance de node.js (s'installe en même temps que node)
- ▶ Equivalent pour du code JavaScript à apt-get
- ▶ Plutôt destiné à du code console ou serveur, bien que des bibliothèques comme jQuery ou Bootstrap y soient présentes





- ▶ Trouver des packages
<https://www.npmjs.com>
- ▶ Créer un package
npm init
- ▶ Le fichier package.json
<http://browsenpm.org/package.json>



- ▶ **Installer un package**

`npm install <package>`

`npm install <package> --save`

`npm install <package>@<version> --save`

Ex : `npm install jquery@1.11.*`

- ▶ **Mettre à jour les packages installés**

`npm update`

- ▶ **Désinstaller**

`npm uninstall lodash`

`npm uninstall --save lodash`



- ▶ **Bower**

Gestionnaire de dépendance pour bibliothèques front-end (CSS/JS/Polices...). Créé par Twitter en 2012

- ▶ **Pré-requis**

Node.js

Git

- ▶ **Installation**

`npm install -g bower`

- ▶ **Créer un projet**

`bower init`

- ▶ **Trouver des packages**

<http://bower.io/search/>





- ▶ **Installer un package**

`bower install <package>`

`bower install <package>#<version>`

Ex : `bower install jquery#1.11.*`

- ▶ **Mettre à jour**

`bower update`

- ▶ **Configuration**

Fichier `.bowerrc`

<http://bower.io/docs/config/>

- ▶ **Dépôts privés :**

<https://github.com/bower/registry>



Node.js



- ▶ Créé 2009 par Ryan Dahl

A l'origine, Ryan Dahl voulait simplifier la création d'une barre d'upload.

- ▶ Sponsorisé par la société Joyent.

- ▶ Un programme en ligne de commande combinant :

- le moteur JavaScript V8 de Chrome
- une boucle d'événement
- une gestion bas niveau des entrées/sorties

- ▶ Un système en production :

- Chez des startups à la pointe : Airbnb, ...
- Dans des grands groupes : Microsoft, PayPal, Walmart, LinkedIn



- ▶ **Windows**

Exécutables : <https://nodejs.org/download/>

- ▶ **OS X**

Exécutables : <https://nodejs.org/download/>

Ou via homebrew : `brew install node`

- ▶ **Debian / Ubuntu**

`sudo apt-get update`

`sudo apt-get install nodejs npm`

- ▶ **Pensez à ajouter le répertoire de Node au Path.**



```
/* Un simple helloworld */  
  
/** @function helloworld */  
function helloworld() {  
  'use strict'; // bonne pratique  
  console.log('Helloworld');  
}  
  
setInterval(helloworld, 1000);
```

- Lancement du programme

node FILE_PATH.js

- Interruption

CTRL-C

```
LearningJS - node - 78x16  
MacBook-Pro-de-Romain:LearningJS romain$ node Node.js/Slides/helloworld.js  
Helloworld  
Helloworld  
Helloworld  
Helloworld  
Helloworld  
Helloworld  
Helloworld  
Helloworld  
Helloworld  
Helloworld  
█
```



```
var http = require('http');  
  
http.createServer(function(req, res) {  
  res.writeHead(200, {'Content-Type': 'text/plain'});  
  res.end('Hello, world !');  
}).listen(8080);
```

► Avantage du JavaScript côté serveur

- Performance : le côté non-bloquant de JavaScript le rend particulièrement performant, plus besoin de gérer les problèmes inter-thread ou inter-processus
- Réutilisation : une bibliothèque ou un composant peut être utilisé sur le client comme sur le serveur
- Apprentissage : vous connaissez déjà JavaScript
- Ecosystème : le nombre de bibliothèques open-source (langage le plus populaire sur GitHub)



```
var http = require('http');  
  
http.createServer(function(req, res) {  
  res.writeHead(200, {'Content-Type': 'text/plain'});  
  res.end('Hello, world !');  
}).listen(8080);
```

- ▶ Node.js implémente côté C++ une boucle d'événement et une gestion non-bloquante des entrées/sorties
- ▶ Ici lorsque qu'une requête HTTP arrive sur le port 8080 la fonction de rappel passée en argument de `createServer` est appelée
- ▶ Il faut quelques millisecondes pour exécuter ce callback, le reste du temps JavaScript est libre de faire autre chose (exécuter des requêtes concurrentes, se connecter à une base de données, écrire dans un fichier...)



- ▶ Node.js contient un certain nombre de modules prédéfinis
<https://nodejs.org/api/>

- | | | | |
|---------------------|---------------|------------------|----------------|
| ▶ Assertion Testing | ▶ Domain | ▶ OS | ▶ Timers |
| ▶ Buffer | ▶ Errors | ▶ Path | ▶ TLS/SSL |
| ▶ C/C++ Addons | ▶ Events | ▶ Process | ▶ TTY |
| ▶ Child Processes | ▶ File System | ▶ Punycode | ▶ UDP/Datagram |
| ▶ Cluster | ▶ Globals | ▶ Query Strings | ▶ URL |
| ▶ Console | ▶ HTTP | ▶ Readline | ▶ Utilities |
| ▶ Crypto | ▶ HTTPS | ▶ REPL | ▶ V8 |
| ▶ Debugger | ▶ Modules | ▶ Stream | ▶ VM |
| ▶ DNS | ▶ Net | ▶ String Decoder | ▶ ZLIB |



- Le module console (global) permet de logger dans la console et de réaliser des benchmarks

```
console.time('100-elements');  
for (var i = 0; i < 100; i++) {  
    console.log(i);  
}  
console.timeEnd('100-elements');  
// 100-elements: 8ms
```



- Le module `Timers` (global) contient les fonctions pour différer l'exécution de callbacks.

```
setTimeout(function() {  
    console.log('1 fois dans 3 secondes');  
}, 3000);  
  
var intervalId = setInterval(function() {  
    console.log('toutes les 2 secondes');  
}, 2000);  
  
setTimeout(function() {  
    console.log('Bye bye');  
    clearInterval(intervalId);  
}, 15000);
```



- ▶ Le module File System (require(fs)) permet les accès aux disques, fichiers, dossiers, droits, etc...

```
var fs = require('fs');

try {
  var stats = fs.statSync('./dist');
}
catch(e) {
  fs.mkdirSync('./dist');
}

var fichiers = fs.readdirSync('./src');

for (var i=0; i<fichiers.length; i++) {
  var fichier = fichiers[i];
  var src = './src/' + fichier;
  var dest = './dist/' + fichier;

  var contenu = fs.readFileSync(src);
  fs.writeFileSync(dest, contenu);
}
```



- Le module net (require(net)) permet les accès réseau

```
var net = require('net');
var server = net.createServer(function(c) { // 'connection' listener
  console.log('client connected');
  c.on('end', function() {
    console.log('client disconnected');
  });
  c.write('hello\r\n');
  c.pipe(c);
});
server.listen(8124, function() { // 'listening' listener
  console.log('server bound');
});
```



► Un chat serveur avec Net

```
var net = require('net');

var clients = {}, cpt = 0;

var server = net.createServer(function(c) { // 'connection' listener
  var me = 'c' + (++cpt);
  console.log('client connected');

  clients[me] = c;
  c.on('end', function() {
    //clients[me].end();
    delete clients[me];
  });
  c.write('Bienvenue sur le Chat !!! (telnet : taper exit pour quitter)\r\n');

  c.on('data', function(chunk) {
    for (var cid in clients) {
      if (clients.hasOwnProperty(cid)) {
        if (chunk.toString().indexOf('exit') === 0) {
          clients[me].end();
          delete clients[me];
          break;
        }

        if (cid !== me) {
          clients[cid].write(chunk.toString());
        }
      }
    }
  })
});

server.listen(8124, function() { // 'listening' listener
  console.log('server bound');
});
```



► Un chat client avec Net

```
var net = require('net');
var readline = require('readline');

var rl = readline.createInterface({
  input: process.stdin,
  output: process.stdout
});

rl.question("Quel est ton pseudo ? ", function(pseudo) {
  console.log("Bienvenue sur le chat", pseudo);

  var client = net.connect({port: 8124}, function() { // 'connect' listener
    console.log('(connecté au serveur)');
    process.stdin.on('readable', function() {
      var chunk = process.stdin.read();
      if (chunk !== null) {
        var msg = chunk.toString();
        msg = msg.substr(0, msg.length - 1); // on retire le \n
        client.write(pseudo + ': ' + msg);
      }
    });
  });

  client.on('data', function(data) {
    console.log(data.toString());
    //client.end();
  });

  client.on('end', function() {
    console.log('disconnected from server');
  });

  rl.close();
});
```



- ▶ **CreateServer**

Contrairement à d'autres technologies, l'implémentation du serveur HTTP se fait dans l'application.

- ▶ **Callback**

Une fonction de rappel est associée au serveur. Elle sera appelée à chaque requête HTTP.

- ▶ **Objets Request et Response**

Node.js abstrait la requête (IncomingMessage) et la réponse (ServerResponse), le callback doit créer une réponse valide avant la fin de son exécution.

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

http.createServer(function(req, res) {
  res.writeHead(200, {'Content-Type': 'text/plain'});
  res.end('Hello, world !');
}).listen(8080);
```




► HTTPS

Le serveur HTTPS démarre avec une clé privée et un certificat. Les serveurs HTTP et HTTPS cohabitent dans la même application.

```
var https = require('https');
var http = require('http');
var fs = require('fs');

function serveur(req, res) {
  res.writeHead(200, {'Content-Type': 'text/plain'});
  res.end('Hello, world !');
}

var options = {
  key: fs.readFileSync('./key.pem', 'utf8'),
  cert: fs.readFileSync('./server.crt', 'utf8')
};

http.createServer(serveur).listen(80);
https.createServer(options, serveur).listen(443);
```



formation.tech



GRUNT

Grunt



▸ Grunt JS

Permet l'automatisation de tâches de développement front-end.

▸ Exemples

- minifier ses fichiers JS
- compiler ses CSS
- compresser les images
- exécuter les tests
- vérifier les conventions de codage



- Installation via npm :
npm install -g grunt-cli

Gruntfile.js

```
/*global module:false*/
module.exports = function(grunt) {

  grunt.initConfig({
    copy: {
      dist: {
        src: 'index.html',
        dest: 'dist/index.html'
      }
    },
    uglify: {
      dist: {
        src: 'script.js',
        dest: 'dist/script.js'
      }
    }
  });

  grunt.loadNpmTasks('grunt-contrib-copy');
  grunt.loadNpmTasks('grunt-contrib-uglify');

  // Default task.
  grunt.registerTask('default', ['copy',
  'uglify']);
};
```

package.json

```
{
  "engines": {
    "node": ">= 0.10.0"
  },
  "devDependencies": {
    "grunt": "^0.4.5",
    "grunt-contrib-copy": "^0.8.0",
    "grunt-contrib-uglify": "^0.9.1"
  }
}
```



src/index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title></title>
</head>
<body>
  <div>
    Prénom : <input type="text" id="prenom">
  </div>
  <p>
    Bonjour <span id="output"></span>
  </p>
  <script src="script.js"></script>
</body>
</html>
```

copy
→

dist/index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title></title>
</head>
<body>
  <div>
    Prénom : <input type="text" id="prenom">
  </div>
  <p>
    Bonjour <span id="output"></span>
  </p>
  <script src="script.js"></script>
</body>
</html>
```

src/script.js

```
!function() {
  'use strict';

  var inputElt =
document.querySelector('#prenom');
  var outputElt =
document.querySelector('#output');

  inputElt.addEventListener('input', function()
{
  outputElt.innerHTML = inputElt.value;
});
}();
```

uglify
→

dist/script.js

```
!function(){"use strict";var
a=document.querySelector("#prenom"),b=document.qu
erySelector("#output");a.addEventListener("input"
,function(){b.innerHTML=a.value}})();
```



Gruntfile.js

```
/*global module:false*/
module.exports = function(grunt) {

  grunt.initConfig({
    copy: {
      dist: {
        src: 'index.html',
        dest: 'dist/index.html'
      }
    },
    uglify: {
      dist: {
        src: 'script.js',
        dest: 'dist/script.js'
      }
    }
  });

  grunt.loadNpmTasks('grunt-contrib-copy');
  grunt.loadNpmTasks('grunt-contrib-uglify');

  // Default task.
  grunt.registerTask('default', ['copy', 'uglify']);
};
```

package.json

```
{
  "devDependencies": {
    "grunt": "^0.4.5",
    "grunt-contrib-copy": "^0.8.0",
    "grunt-contrib-uglify": "^0.9.1"
  }
}
```

- package.json créé avec :
npm init
npm install grunt --save-dev
npm install grunt-contrib-copy --save-dev
npm install grunt-contrib-uglify --save-dev



- Liste des plugins pour grunt :
<http://gruntjs.com/plugins>
(4,403 plugins en juillet 2015)
- Les plugins contrib-* sont ceux des développeurs de grunt.



► jit-grunt :

Installation : `npm install jit-grunt --save-dev`

Simplifie le chargement de plugins

Avant

```
/*global module:false*/  
module.exports = function(grunt) {  
  
  grunt.loadNpmTasks('grunt-contrib-clean');  
  grunt.loadNpmTasks('grunt-contrib-concat');  
  grunt.loadNpmTasks('grunt-contrib-copy');  
  grunt.loadNpmTasks('grunt-contrib-cssmin');  
  grunt.loadNpmTasks('grunt-contrib-jshint');  
  grunt.loadNpmTasks('grunt-contrib-less');  
  grunt.loadNpmTasks('grunt-contrib-uglify');  
  grunt.loadNpmTasks('grunt-contrib-watch');  
  grunt.loadNpmTasks('grunt-google-cdn');  
  grunt.loadNpmTasks('grunt-rev');  
  grunt.loadNpmTasks('grunt-spritesmith');  
  grunt.loadNpmTasks('grunt-usemin');  
  
  grunt.initConfig({  
    // ...  
  });  
  
  // Default task.  
  grunt.registerTask('default', [  
    // ...  
  ]);  
  
};
```

Après

```
/*global module:false, require*/  
module.exports = function(grunt) {  
  'use strict';  
  
  require('jit-grunt')(grunt, {  
    useminPrepare: 'grunt-usemin',  
    cdnify: 'grunt-google-cdn',  
    sprite: 'grunt-spritesmith'  
  });  
  
  // Project configuration.  
  grunt.initConfig({  
    // ...  
  });  
  
  // Default task.  
  grunt.registerTask('default', [  
    // ...  
  ]);  
  
};
```




- ▶ **grunt-contrib-less :**
npm install grunt-contrib-less --save-dev
Compile des fichiers LESS en CSS

```
module.exports = function(grunt) {  
  
  // ...  
  
  grunt.initConfig({  
    less: {  
      dev: {  
        files: [{  
          expand: true,  
          cwd: 'less',  
          src: ['*.less'],  
          dest: 'css/',  
          ext: '.css'  
        }]  
      },  
    },  
  });  
  
  // Default task.  
  grunt.registerTask('default', [  
    // ...  
  ]);  
  
};
```



► grunt-autoprefixer :

npm install grunt-autoprefixer --save-dev

Rajoute automatiquement les préfixes -moz, -webkit, -o, -ms en fonction des versions minimales des navigateurs à supporter

```
module.exports = function(grunt) {  
  
  // ...  
  
  grunt.initConfig({  
    // ...  
    autoprefixer: {  
      options: {  
        browsers: ['last 2 versions', 'ie 8', 'ie 9']  
      },  
      dev: {  
        files: [{  
          expand: true,  
          cwd: 'css/',  
          src: '{,*/}*.css',  
          dest: 'css/'  
        }]  
      },  
    },  
  });  
  
  // Default task.  
  grunt.registerTask('default', [  
    // ...  
  ]);  
  
};
```



► grunt-contrib-watch :

npm install grunt-contrib-watch --save-dev

Surveille les modifications sur des fichiers, exécute des taches en cas de changement

```
module.exports = function(grunt) {  
  
  // ...  
  
  grunt.initConfig({  
    // ...  
    watch: {  
      less: {  
        files: ['less/**/*.less'],  
        tasks: ['less:dev', 'autoprefixer:dev']  
      }  
    },  
  });  
  
  // Default task.  
  grunt.registerTask('default', [  
    // ...  
  ]);  
};
```

grunt watch

Surveille les fichiers less, compile les fichiers CSS en ajoutant les préfixes



- ▶ **grunt-contrib-concat :**
npm install grunt-contrib-concat --save-dev
Concatène plusieurs fichiers en un. Utile pour optimiser les temps de chargement CSS/JS
- ▶ **grunt-contrib-uglify :**
npm install grunt-contrib-uglify --save-dev
Compresse les fichiers JS
- ▶ **grunt-contrib-cssmin :**
npm install grunt-contrib-cssmin --save-dev
Compresse les fichiers CSS



- ▶ **grunt-contrib-copy :**
npm install grunt-contrib-copy --save-dev
Copie des fichiers
- ▶ **grunt-contrib-clean :**
npm install grunt-contrib-clean --save-dev
Supprime des fichiers



► grunt-usemin:

npm install grunt-usemin --save-dev

Génère une configuration pour concat, uglify, cssmin à partir d'un fichier HTML

index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title></title>

  <!-- build:css css/app.css -->
  <link rel="stylesheet" href="css/body.css">
  <link rel="stylesheet" href="css/button.css">
  <!-- endbuild -->
</head>
<body>

  <!-- build:js js/app.js -->
  <script src="js/create-button.js"></script>
  <script src="js/button-listener.js"></script>
  <!-- endbuild -->
</body>
</html>
```

Gruntfile.js

```
/*global module, require*/
module.exports = function(grunt) {
  'use strict';

  // ...

  grunt.initConfig({
    // ...
    useminPrepare: {
      html: 'index.html'
    },

    usemin: {
      html: ['dist/{,*/}*.html'],
      css: ['dist/{,*/}*.css'],
      js: ['dist/{,*/}*.js'],
    },

  });

  // Default task.
  grunt.registerTask('default', [
    // ...
  ]);
};
```



config générée

```
{
  "concat": {
    "generated": {
      "files": [{
        "dest": ".tmp/concat/css/app.css",
        "src": ["css/body.css", "css/button.css"]
      }, {
        "dest": ".tmp/concat/js/app.js",
        "src": ["js/create-button.js", "js/button-
listener.js"]
      }]
    },
    "uglify": {
      "generated": {
        "files": [{
          "dest": "dist/js/app.js",
          "src": [".tmp/concat/js/app.js"]
        }]
      }
    },
    "cssmin": {
      "generated": {
        "files": [{
          "dest": "dist/css/app.css",
          "src": [".tmp/concat/css/app.css"]
        }]
      }
    }
  }
}
```

index.html généré

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title></title>

  <link rel="stylesheet" href="css/app.css">
</head>
<body>

<script src="js/app.js"></script>
</body>
</html>
```

app.css généré

```
body{background:beige}button{width:50px;height:50px}
```

app.js généré

```
!function(){"use strict";var
a=document.createElement("button");a.innerHTML=0,a.id="monBouton",document.body.appendChild(a)}(),!function()
{"use strict";var
a=document.querySelector("#monBouton");a.addEventListener("click",function(){this.innerHTML++}})();
```



- ▶ **contrib-connect :**
serveur web
- ▶ **karma :**
lancer des tests
- ▶ **concurrent :**
exécuter des tâches en parallèle
- ▶ **sass :**
compile des fichiers SASS en CSS
- ▶ **contrib-imagemin :**
compresser des images
- ▶ **contrib-htmlmin :**
minifier le HTML
- ▶ **newer :**
ne lancer les tâches que sur les nouveaux fichiers
- ▶ **rev :**
génère un nom de fichier avec hash pour le cache (avec usemin)
- ▶ **contrib-jshint, jscs :**
vérifie les conventions sur les fichiers JS
- ▶ **google-cdn :**
remplace les fichiers locaux par des CDN
- ▶ **spritesmith :**
génère des fichiers Sprite CSS



- ▶ Grunt Init

Assistant de création de projet grunt

- ▶ Installation

npm install -g grunt-init

- ▶ Création du projet

grunt-init gruntfile

```
Please answer the following:
[?] Is the DOM involved in ANY way? (Y/n) Y
[?] Will files be concatenated or minified? (Y/n) Y
[?] Will you have a package.json file? (Y/n) Y
[?] Do you need to make any changes to the above before continuing? (y/N) N

Writing Gruntfile.js...OK
Writing package.json...OK

Initialized from template "gruntfile".
```

- ▶ Créer son propre assistant/plugin :

<https://github.com/gruntjs/grunt-init-gruntplugin>



► Gulp

Equivalent de grunt, repose sur les streams Node.js (utilise la RAM plutôt que les fichiers).

Deviens très populaire, 1645 plugins contre 4403 pour grunt (juillet 2015)

► Broccoli

484 plugins

► Brunch

262 plugins

► Prepros / CodeKit

<https://prepros.io>

<https://incident57.com/codekit/>

gulpfile.js

```
var gulp = require('gulp');
var uglify = require('gulp-uglify');

gulp.task('scripts', function() {
  // Minify and copy all JavaScript (except vendor
  // scripts)
  gulp.src(['client/js/**/*.js', '!client/js/vendor/
  **'])
    .pipe(uglify())
    .pipe(gulp.dest('build/js'));

  // Copy vendor files
  gulp.src('client/js/vendor/**')
    .pipe(gulp.dest('build/js/vendor'));
});

// The default task (called when you run `gulp`)
gulp.task('default', function() {
  gulp.run('scripts');

  // Watch files and run tasks if they change
  gulp.watch('client/js/**', function(event) {
    gulp.run('scripts');
  });
});
```



Express



- ▶ **Définition d'un framework web :**
Ensemble de composants logiciels permettant d'architecturer un projet logiciel.
- ▶ **Différences par rapport à une bibliothèque :**
Le framework ne se destine pas à une tâche précise (ensemble de bibliothèques)
Le framework instaure un cadre de travail (squelettes d'application, documentation sur l'architecture...)



- ▶ **Java**

Struts (2000), Spring (2003), GWT (2006), Play (2007)...

- ▶ **Ruby**

Ruby on Rails (2005), Sinatra (2007)...

- ▶ **Python**

Django (2005)...

- ▶ **PHP**

Symfony, Zend Framework, CakePHP, CodeIgniter...



- ▶ Clients

AngularJS (2010), Ember.js (2011)

- ▶ Server

Express (2009), Hapi (2012)

- ▶ Fullstack (Client + Server)

Meteor (2012), Sails.js (2012)...



▸ Express

Framework pour Node.js le plus populaire, créé en 2009, aujourd'hui en version 4. Permet d'architecturer plus facilement le serveur web.

Très souvent utilisé pour construire des APIs REST.

▸ Avantages sur le module HTTP de Node.js

- Gestion des URLs et des méthodes HTTP
- Approche MVC
- Utilisation de middlewares qui permettent d'étendre le code
- De nombreux middleware open-source existent
- Construit comme une surcouche de HTTP, les objets Request et Response sont simplement étendus

▸ Installation

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

Express



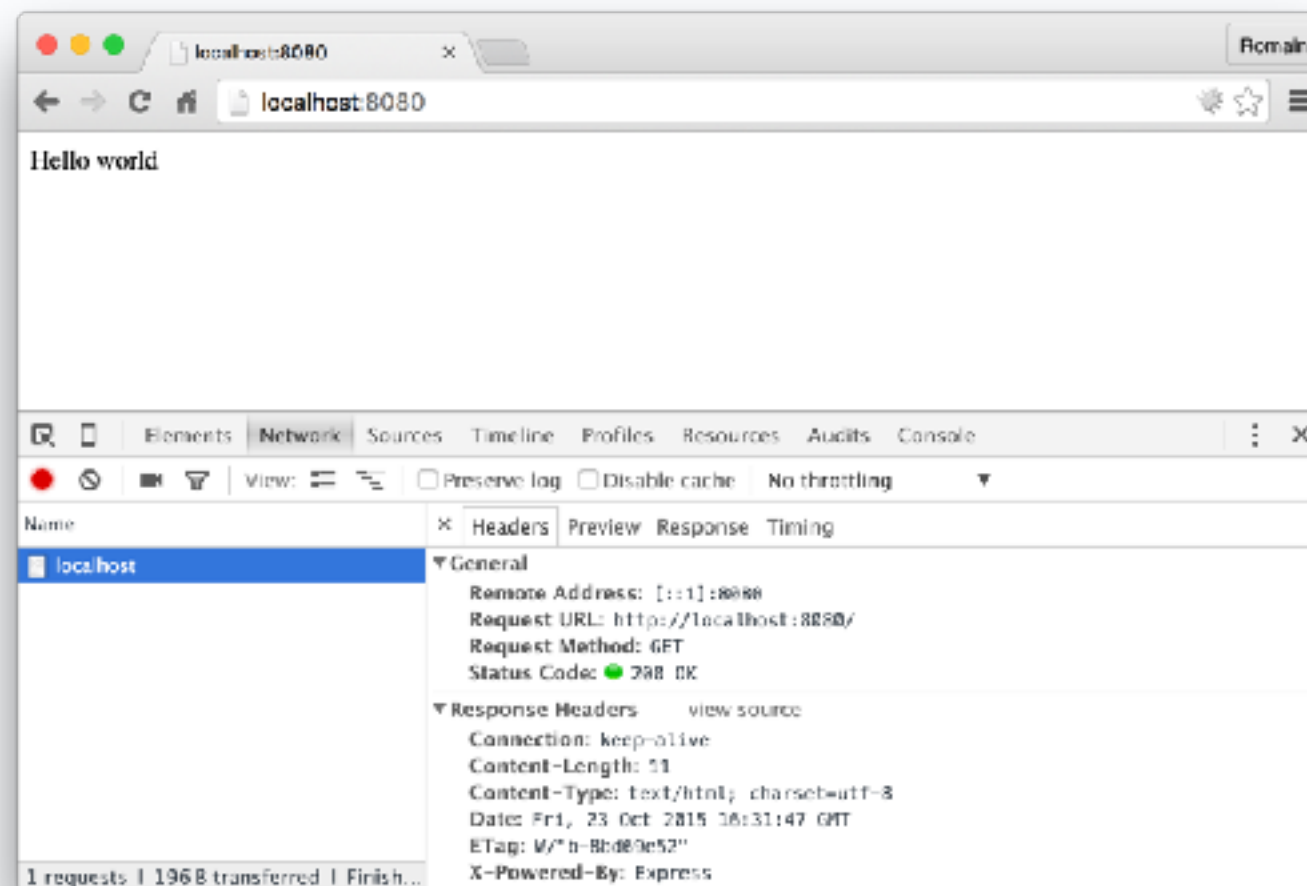
```
var express = require('express');

var app = express();

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

app.listen(8080);

console.log("URL http://localhost:8080/");
```





- ▶ **Définition**

L'architecture MVC est un Design Pattern apparu en Smalltalk et très répandu dans les frameworks web

- ▶ **Objectif**

L'objectif est de séparer les responsabilités de 3 types de composants : le Modèle (Model), la Vue (View), le Contrôleur (Controller)

- ▶ **Documentation :**

<http://martinfowler.com/eaCatalog/modelViewController.html>

<http://martinfowler.com/eaDev/uiArchs.html>

<http://fr.wikipedia.org/wiki/Modèle-vue-contrôleur>



► Modèle

Données, accès aux données, validation

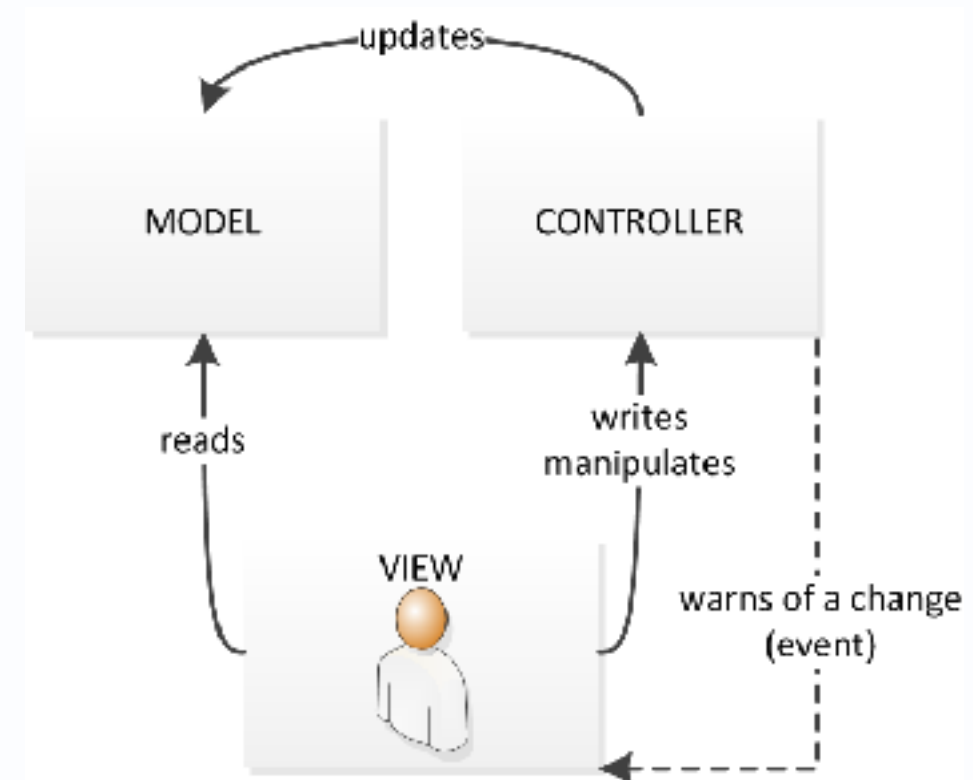
► Vue

Rendu. Se limiter à :

- affichage de variable
- bloc conditionnels if .. else if .. else (ex : afficher ou non message d'erreur, menu qui dépend d'une authentification)
- boucles foreach (uniquement foreach, ce qui impose d'avoir trié/filtré au préalable)
- appel à des fonctions de filtrage, de formatage, de rendu (parfois appelées aides de vues)

► Contrôleur

Analyse de la requête, interrogation du Modèle, transmission des données à la vue, gestion des erreurs, des redirections...





- ▶ **Définition**

Un middleware est une fonction qui va s'exécuter en amont ou en aval d'une requête dans Express pour l'étendre simplement.

- ▶ **Exemple**

Logs de requêtes, authentification, gestion des requêtes Cross-Domaines, support d'un moteur de templates...

- ▶ **Connect**

Historiquement Express utilisait le module npm connect pour la mise en place de middleware. A partir d'Express 4, les développeurs d'Express ont développé leur propre système de middleware tout en gardant la compatibilité avec Connect.



- ▶ **Introduction**

Express fourni un générateur de squelette d'application pour démarrer rapidement ses projets web (plutôt adapté aux rendus HTML)

- ▶ **Installation**

```
npm install -g express-generator
```

- ▶ **Création du squelette d'application**

```
express HelloWorld
```

- ▶ **Installation des dépendances**

```
cd HelloWorld && npm install
```

- ▶ **Lancement de l'application**

```
DEBUG=HelloWorldEJS:* npm start
```



- ▶ Autres options d'installation du squelette

```
MacBook-Pro-de-Romain:HelloWorld remain$ express -h

Usage: express [options] [dir]

Options:

  -h, --help            output usage information
  -V, --version          output the version number
  -e, --ejs              add ejs engine support (defaults to jade)
  --hbs                  add handlebars engine support
  -H, --hogan            add hogan.js engine support
  -c, --css <engine>    add stylesheet <engine> support (less|stylus|compass|sass) (defaults to plain css)
  --git                  add .gitignore
  -f, --force            force on non-empty directory

MacBook-Pro-de-Romain:HelloWorld remain$
```

- ▶ Choix du moteur de templates
Jade, EJS, Handlebars, Hogan.js
- ▶ Choix d'un préprocesseur CSS
CSS, Less, Stylus, Compass, Sass



- ▶ **app.js**
Configuration de l'application, objet principal
- ▶ **bin/www**
Démarrage du serveur
- ▶ **package.json**
Dépendances npm
- ▶ **public**
Fichiers statiques (images, scripts client, css, pdf...)
- ▶ **routes**
Contrôleurs, configuration des URLs
- ▶ **views**
Fichiers de rendus (ici au format Jade)

```
├── app.js
├── bin
│   └── www
├── node_modules
│   └── ...
├── package.json
├── public
│   ├── images
│   ├── javascripts
│   └── stylesheets
│       └── style.css
├── routes
│   ├── index.js
│   └── users.js
└── views
    ├── error.jade
    ├── index.jade
    └── layout.jade
```



► bin/www

- Dépendances
- Définition du port (variable d'env PORT ou 3000)
- Création du serveur
- Démarrage du serveur
- Listeners sur erreurs et démarrage

```
#!/usr/bin/env node

/**
 * Module dependencies.
 */

var app = require('../app');
var debug = require('debug')('Helloworld:server');
var http = require('http');

/**
 * Get port from environment and store in Express.
 */

var port = normalizePort(process.env.PORT || '3000');
app.set('port', port);

/**
 * Create HTTP server.
 */

var server = http.createServer(app);

/**
 * Listen on provided port, on all network interfaces.
 */

server.listen(port);
server.on('error', onError);
server.on('listening', onListening);

// ...
```



▸ app.js

- Dépendances
- Chargement des routes
- Création de l'objet app
- Définition du moteur de templates
- Définition des middlewares à appeler avant le contrôleur
- Définition des contrôleurs sur un préfixe d'URL
- Middleware pour les erreurs 404
- Middleware pour afficher les erreurs (avec env de dev et de prod)

```
var express = require('express');
var logger = require('morgan');
var cookieParser = require('cookie-parser');
var bodyParser = require('body-parser');

var routes = require('./routes/index');
var users = require('./routes/users');

var app = express();

// view engine setup
app.set('views', path.join(__dirname, 'views'));
app.set('view engine', 'jade');

app.use(logger('dev'));
app.use(bodyParser.json());
app.use(bodyParser.urlencoded({ extended: false }));
app.use(cookieParser());
app.use(express.static(path.join(__dirname, 'public')));

app.use('/', routes);
app.use('/users', users);

// catch 404 and forward to error handler
app.use(function(req, res, next) {
  var err = new Error('Not Found');
  err.status = 404;
  next(err);
});

// error handlers
// ...
```




- routes/index.js
 - Dépendances
 - Association d'un contrôleur à la l'URL GET /
 - Appel de la vues en transmettant la variable title (contenu 'Express')

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

/* GET home page. */
router.get('/', function(req, res, next)
{
  res.render('index', { title:
'Express' });
});

module.exports = router;
```



▸ views/index.jade

- Jade : Syntaxe très concise, l'indentation fait l'imbrication des balises, parenthèses pour les attributs
- Héritage du layout
- Remplacement du block content du layout par celui de la vue (inspiré de Django Template Engine, Twig...)
- Création de la balise h1 ayant comme contenu la variable title
- Création de la balise p qui concatène Welcome to et la variable title

```
// views/index.jade
extends layout

block content
  h1= title
  p Welcome to #{title}
```

```
// views/layout.jade
doctype html
html
  head
    title= title
    link(rel='stylesheet', href='/stylesheets/style.css')
  body
    block content
```



▸ views/index.ejs

- Syntaxe plus simple que Jade proche de PHP, ASP, JSP
- `<%= title %>` : écriture de la variable title

```
<!DOCTYPE html>
<html>
  <head>
    <title><%= title %></title>
    <link rel='stylesheet' href='/stylesheets/style.css' />
  </head>
  <body>
    <h1><%= title %></h1>
    <p>Welcome to <%= title %></p>
  </body>
</html>
```



- Réponse à toutes les méthodes HTTP

```
router.all('/api/*', requireAuthentication);
```

- Réponse aux requêtes sur certaines méthodes HTTP

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, connect

```
router.get('/', function(req, res){  
  res.send('hello world');  
});
```

- Avec une RegExp

```
router.get(/^\/commits\/(\w+)(?:\.\.(\w+))?$/, function(req, res){  
  var from = req.params[0];  
  var to = req.params[1] || 'HEAD';  
  res.send('commit range ' + from + '..' + to);  
});
```



► Route avec paramètres nommés

```
router.get('/:id', function(req, res, next) {  
  var id = req.params.id;  
  
  if (!model[id-1]) {  
    return next();  
  }  
  
  res.json({  
    data: model[id-1]  
  });  
});
```

► Ne pas confondre avec la query string

Ex : /contacts?page=1&limit=100

```
// GET /search?q=tobi+ferret  
req.query.q  
// => "tobi ferret"
```



- ▶ Middleware

Fonction qui s'exécute en amont ou en aval d'un contrôleur

- ▶ Exemple

Ajoute les entêtes à la réponse HTTP permettant d'autoriser les requêtes Cross-Domain

```
router.use(function(req, res, next) {  
  res.setHeader('Access-Control-Allow-Origin', '*');  
  res.setHeader('Access-Control-Allow-Headers', 'X-Requested-With');  
  next();  
});
```



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

app.use(function(req, res, next) {
  var err = new Error('Not Found');
  err.status = 404;
  next(err);
});

app.use(function(err, req, res, next) {
  res.status(err.status || 500);
  res.render('error', {
    message: err.message,
    error: err
  });
});
```

► 3 middlewares

1. Router qui contient toutes les URLs préfixées par /users
2. Middleware appelé si l'un des contrôleurs ou middlewares précédents appelle next(), permet de gérer simplement les erreurs 404
3. Middleware appelé si l'un des contrôleurs ou middlewares précédents appelle next(err) ou les erreurs non-interceptées



- ▶ Request

L'objet Request hérite de IncomingMessage du module HTTP

- ▶ Middleware body-parser

Le middleware body-parser ajouter la propriété body à l'objet request avec le contenu du corps de requête parsé

```
var express = require('express');
var bodyParser = require('body-parser');

var app = express();
app.use(bodyParser.urlencoded({ extended: false }));

app.get('/', function(req, res) {
  var html = '<form method="post">';
  html += '  <p>Prénom : <input name="prenom"></p>';
  html += '  <p>Nom : <input name="nom"></p>';
  html += '  <p><button type="submit">Valider</button></p>';
  html += '</form>';

  res.send(html);
})

app.post('/', function(req, res) {
  // Prénom : Romain, nom : Bohdanowicz
  res.send(`Prénom : ${req.body.prenom}, nom : ${req.body.nom}`);
})

app.listen(3000);
```




► Middleware multer

Le middleware multer ajouter la propriété file ou files (upload multiple) à l'objet request et contient des informations sur le fichier uploadé.

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

var app = express();
var upload = multer({ dest: 'uploads/' });

app.get('/', function(req, res) {
  var html = '<form method="post" enctype="multipart/form-data">';
  html += '  <p>Fichier : <input type="file" name="fichier"></p>';
  html += '  <p><button type="submit">Valider</button></p>';
  html += '</form>';

  res.send(html);
});

app.post('/', upload.single('fichier'), function(req, res){
  console.log(req.file);
  res.status(204).end();
});

app.listen(3000);
```

```
{ fieldname: 'fichier',
  originalname: '2010_Q3.pdf',
  encoding: '7bit',
  mimetype: 'application/pdf',
  destination: 'uploads/',
  filename: '799e08c05ef96ac6ec6ac5b714941161',
  path: 'uploads/799e08c05ef96ac6ec6ac5b714941161',
  size: 80108 }
```



► JSON

L'objet Response contient une méthode `json` qui sérialise un objet et renvoie les bons entêtes HTTP. Associés aux méthodes de l'objet Request et aux middlewares `body-parser` et `cors`, Express est le framework idéal pour la mise en place d'un API REST qui communique en JSON.

```
var app = express();
app.use(cors({ allowedHeaders: 'X-Requested-With' }));
app.use(bodyParser.json());
app.use(bodyParser.urlencoded({ extended: false }));

app.use('/api/v1/contacts', apiContacts);

app.listen(80);
```



```
var express = require('express');
var contacts = require('../model/contacts').slice(0);

var api = express.Router();

api.get('/', function(req, res) {
  res.json({contacts});
});

api.get('/:id', function(req, res, next) {
  var id = parseInt(req.params.id);
  var contact = contacts.find(elt => elt.id === id);

  if (!contact) return next();

  res.json({contact});
});

api.post('/', function(req, res, next) {
  var contact = req.body;

  contact.id = contacts[contacts.length-1].id + 1;
  contacts.push(contact);

  res.status(201);
  res.json(contact);
});

module.exports = api;
```



NoSQL



► NoSQL

Not Only SQL, le nom qu'on donne au mouvement depuis quelques années de ne pas tout stocker sous la forme de base de données relationnels (MySQL, SQLite, PostgreSQL, Oracle, SQL Server...)

► Intérêts

Performance, scalabilité, haute-disponibilité

► Catégories

- Clé / valeur (Redis / Memcached...)
- Orienté Colonne (HBase / Cassandra...)
- Orienté Document (MongoDB / CouchDB...)
- Orienté Graphe (Neo4j)

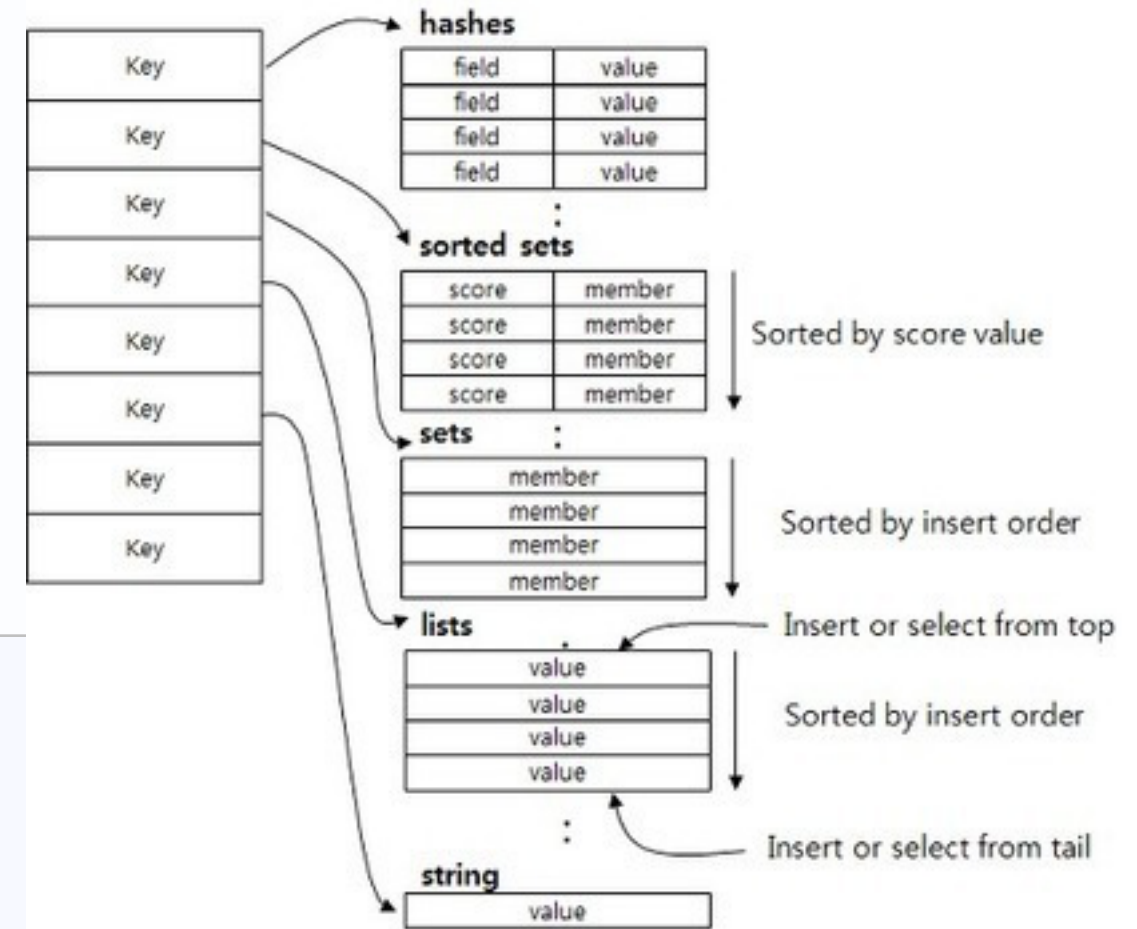


► Exemple Redis

```
var redis = require("redis"),
    client = redis.createClient();

client.on("error", function (err) {
  console.log("Error " + err);
});

client.set("string key", "string val", redis.print);
client.hset("hash key", "hashtest 1", "some value", redis.print);
client.hset(["hash key", "hashtest 2", "some other value"], redis.print);
client.hkeys("hash key", function (err, replies) {
  console.log(replies.length + " replies:");
  replies.forEach(function (reply, i) {
    console.log("    " + i + ": " + reply);
  });
  client.quit();
});
```





► Exemple Cassandra

```
var cassandra = require('cassandra-driver');
var client = new cassandra.Client({ contactPoints: ['h1', 'h2'], keyspace: 'ks1' });
var query = 'SELECT email, last_name FROM user_profiles WHERE key=?';
client.execute(query, ['guy'], function(err, result) {
  assert.ifError(err);
  console.log('got user profile with email ' + result.rows[0].email);
});
```

	A	B	C	D	E
1	foo	bar	hello		
2		Tom			
3			java	scala	cobol

Organisation d'une table dans
une BDD relationnelle

1	A foo	B bar	C hello
2	B Tom		
3	C java	D scala	E cobol

Organisation d'une table dans
une BDD orientée colonnes



► Exemple CouchDB

```
var cradle = require('cradle');
var db = new(cradle.Connection)().database('starwars');

db.get('vader', function (err, doc) {
  doc.name; // 'Darth Vader'
  assert.equal(doc.force, 'dark');
});

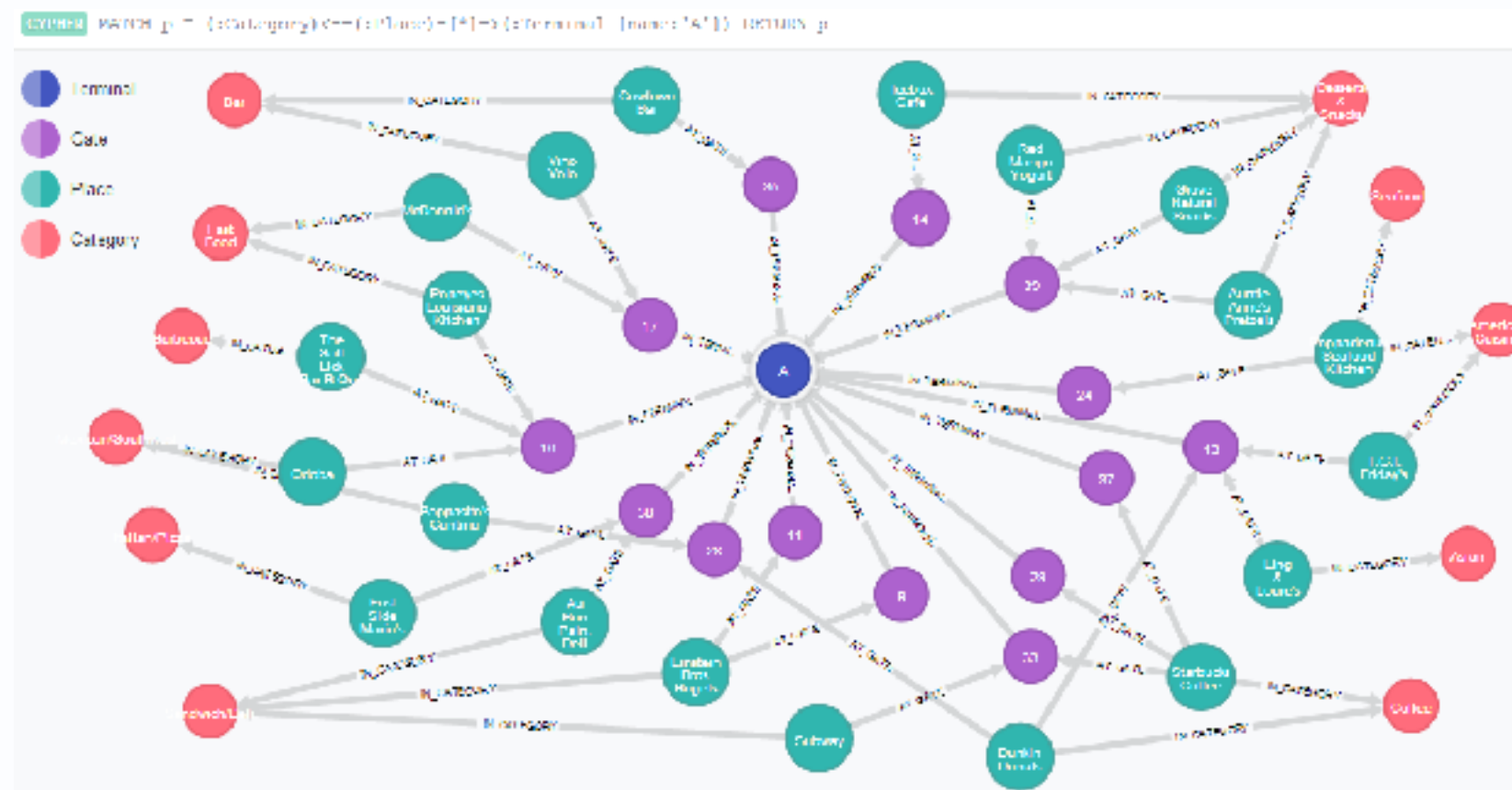
db.save('skywalker', {
  force: 'light',
  name: 'Luke Skywalker'
}, function (err, res) {
  if (err) {
    // Handle error
  } else {
    // Handle success
  }
});
```

```
db.users.insert (  ← collection
  {
    name: "sue",    ← field: value
    age: 26,        ← field: value
    status: "A"     ← field: value
  }                } document
)
```


- ▶ Exemple neo4j

```
var r = require('request');

r.post({
  uri: 'http://localhost:7474/db/data/transaction/commit',
  json: {
    statements: [{
      statement: 'MATCH (n:User) RETURN n, labels(n) as l LIMIT {limit}',
      parameters: { limit: 10 }
    }]
  },
  function(err, res) { console.log(JSON.stringify(res.body)); }
});
```





- ▶ **MongoDB**

Base de données écrite sur V8, le moteur JS de Chrome

- ▶ **Document**

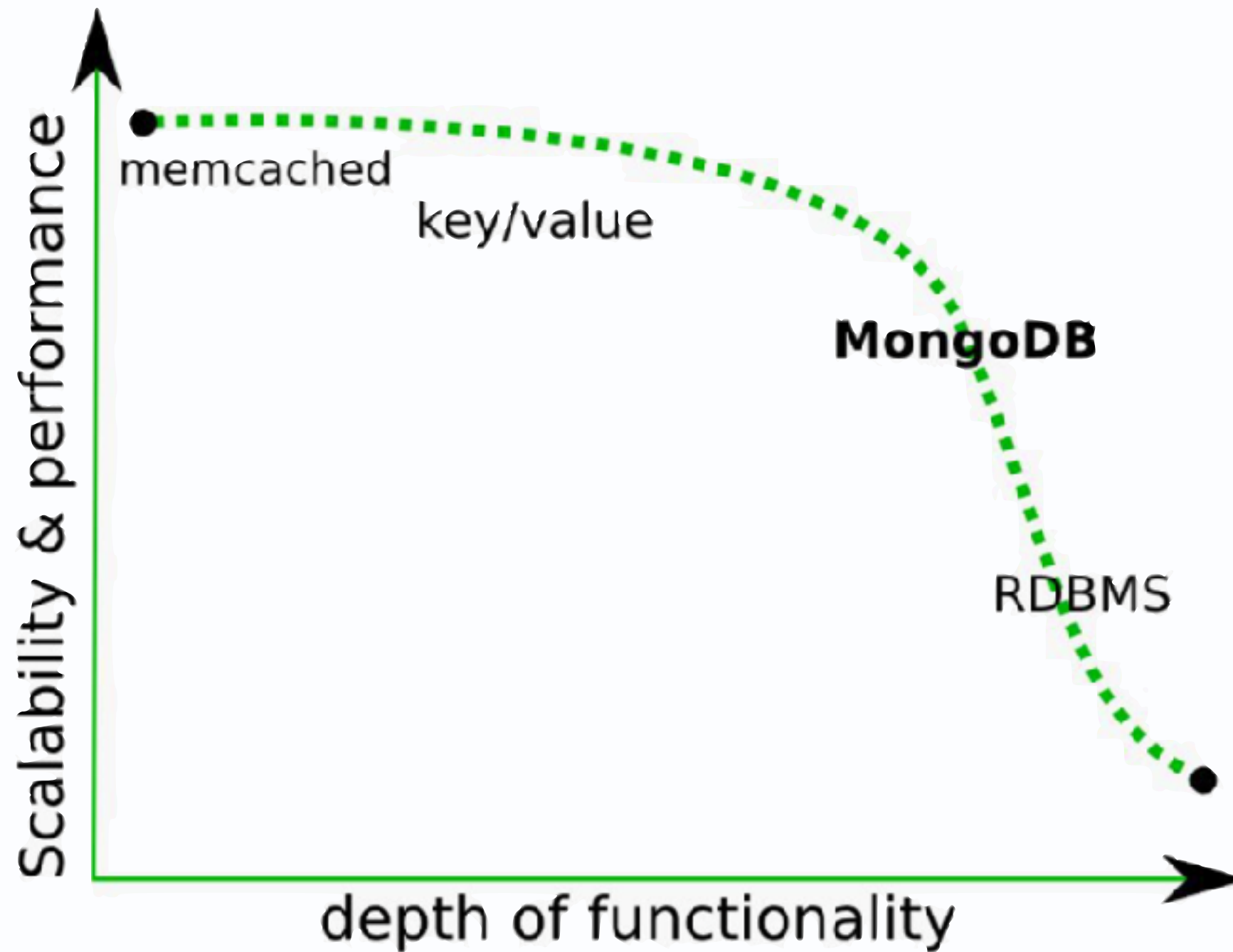
MongoDB permet de manipuler des objets structurés au format BSON (JSON binaire). Les données prennent la forme de documents enregistrés eux-mêmes dans des collections.

- ▶ **Accès aux données**

L'accès aux données se fait via un API JavaScript, pour les requêtes complexes, on utilise des objets de requêtes

- ▶ **Absence de Schéma**

Contrairement à un SGBDR, les documents stockés dans une collection peuvent avoir des formats complètement différents. Les données peuvent également être imbriquées.



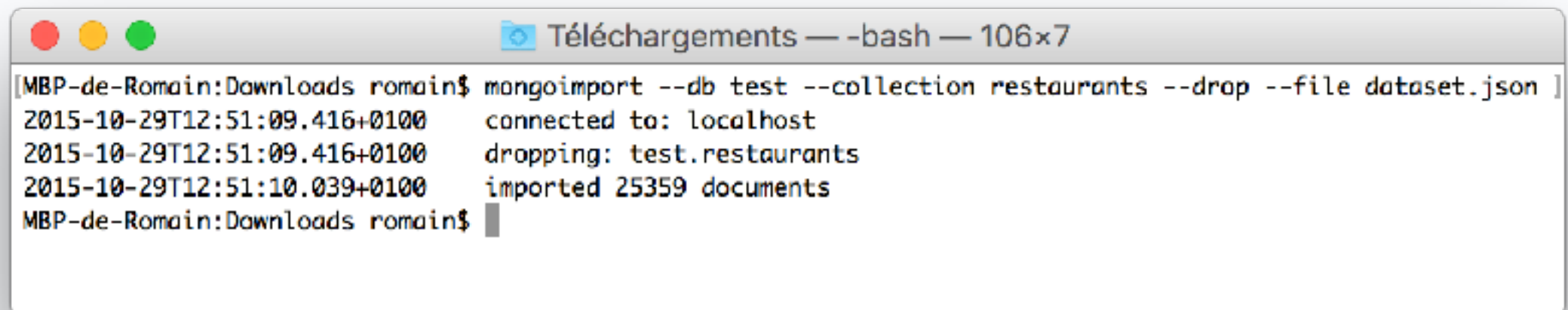


- Parallèle avec un SGBDR

MongoDB	SGBDR
Base de données	Base de données
Collection	Table
Document	Enregistrement
Pas de schéma	Schéma
API JavaScript	SQL



- ▶ Jeu de données d'exemple fourni par Mongo
<https://raw.githubusercontent.com/mongodb/docs-assets/primer-dataset/dataset.json>
- ▶ Importer un jeu de données
`mongoimport --db test --collection restaurants --drop --file dataset.json`

A screenshot of a macOS terminal window titled 'Téléchargements — -bash — 106x7'. The terminal shows the execution of the 'mongoimport' command. The output indicates a successful connection to localhost, dropping the existing 'restaurants' collection, and importing 25359 documents from the 'dataset.json' file.

```
MBP-de-Romain:Downloads romain$ mongoimport --db test --collection restaurants --drop --file dataset.json ]
2015-10-29T12:51:09.416+0100    connected to: localhost
2015-10-29T12:51:09.416+0100    dropping: test.restaurants
2015-10-29T12:51:10.039+0100    imported 25359 documents
MBP-de-Romain:Downloads romain$
```



► MongoDB

Mongo livre un programme client en ligne de commande pour accéder à la base.

```
romain — mongo — 91x17
[MBP-de-Romain:~ romain$ mongo
MongoDB shell version: 3.0.7
connecting to: test
> use address_book
switched to db address_book
> db.contact.find()
{ "_id" : ObjectId("562d4e878561c01ec2e43cfb"), "prenom" : "Steve", "nom" : "Jobs" }
{ "_id" : ObjectId("562d4e918561c01ec2e43cfc"), "prenom" : "Bill", "nom" : "Gates" }
{ "_id" : ObjectId("562d4eab8561c01ec2e43cfd"), "prenom" : "Mark", "nom" : "Zuckerberg" }
> db.contact.insert({prenom: 'Steve', nom: 'Ballmer'})
WriteResult({ "nInserted" : 1 })
> db.contact.find({prenom: 'Steve'})
{ "_id" : ObjectId("562d4e878561c01ec2e43cfb"), "prenom" : "Steve", "nom" : "Jobs" }
{ "_id" : ObjectId("562d4ee1321ac0f47f03ce9d"), "prenom" : "Steve", "nom" : "Ballmer" }
> ]
```



► Principales Commandes MongoShell

Shell Helpers	JavaScript Equivalents
show dbs, show databases	db.adminCommand('listDatabases')
use <db>	db = db.getSiblingDB('<db>')
show collections	db.getCollectionNames()
show users	db.getUsers()
show roles	db.getRoles({showBuiltinRoles: true})
show log <logname>	db.adminCommand({ 'getLog' : '<logname>' })
show logs	db.adminCommand({ 'getLog' : '*' })
it	<pre> cursor = db.collection.find() if (cursor.hasNext()){ cursor.next(); } </pre>

► Liste des méthodes MongoShell

<https://docs.mongodb.org/manual/reference/method/>



- ▶ MongoClient

API officiel fourni MongoDB pour accéder aux données sous Node.js

- ▶ Installation

`npm install mongodb --save`

- ▶ Insertion

```
var MongoClient = require('mongodb').MongoClient;
var url = 'mongodb://localhost:27017/addressbook';
MongoClient.connect(url, function(err, db) {
  if (err) {
    console.log('Erreur : ' + err);
    return;
  }
  var cursor = db.collection('contacts').insert({prenom: 'Romain', nom: 'Bohdanowicz'},
function(err, result) {
  if (err) {
    console.log('Erreur : ' + err);
    return;
  }
  console.log('Le contact a bien été inséré');
});
});
```




► Modification

```
var cursor = db.collection('contacts').update({nom: 'Bohdanowicz'}, {prenom: 'ROMAIN', nom: 'BOHDANOWICZ'}, {upsert:true}, function(err, result) {  
  if (err) {  
    console.log('Erreur : ' + err);  
    return;  
  }  
  
  console.log('Le contact a bien été mis à jour');  
});
```

► Suppression

```
var cursor = db.collection('contacts').removeOne({nom: 'BOHDANOWICZ'}, function(err, result) {  
  if (err) {  
    console.log('Erreur : ' + err);  
    return;  
  }  
  
  console.log('Le contact a bien été supprimé');  
});
```



► Recherche

```
var MongoClient = require('mongodb').MongoClient;

var url = 'mongodb://localhost:27017/addressbook';

MongoClient.connect(url, function(err, db) {

    if (err) {
        console.log('Erreur : ' + err);
        return;
    }
    var cursor = db.collection('contacts').find( );
    cursor.toArray(function(err, contacts) {
        console.log(contacts);
        db.close();
    });
});
```



► Recherche multi-critères

Exemple : Restaurants de Brooklyn, ET dont la cuisine est française OU italienne ET dont l'une des notes est supérieur à 40

```
var MongoClient = require('mongodb').MongoClient;
var url = 'mongodb://localhost:27017/test';
MongoClient.connect(url, function(err, db) {
  if (err) {
    console.log('Erreur : ' + err);
    return;
  }
  var cursor = db.collection('restaurants').find({
    borough: 'Brooklyn',
    $or: [
      { "cuisine": "Italian" },
      { "cuisine": "French" } ],
    'grades.score': { $gt: 40 }
  });
  cursor.toArray(function(err, restaurants) {
    restaurants.forEach(function(r) {
      console.log(`Nom : ${r.name}, cuisine : ${r.cuisine}, adresse : ${r.address.building} ${r.address.street}`);
    });
    db.close();
  });
});
```

```
MongoClient — -bash — 70x9
MBP-de-Romain:MongoClient romain$ node multicriteres.js
Nom : Doc Wine Bar, cuisine : Italian, adresse : 83 North 7 Street
Nom : Le Gamin, cuisine : French, adresse : 556 Vanderbilt Avenue
Nom : Peperoncino, cuisine : Italian, adresse : 72 5 Avenue
Nom : Patrizia'S, cuisine : Italian, adresse : 35 Broadway
Nom : Tutta Pasta, cuisine : Italian, adresse : 160 7 Avenue
Nom : Anella, cuisine : Italian, adresse : 222 Franklin Street
Nom : Joe'S Pizza, cuisine : Italian, adresse : 349 5 Avenue
MBP-de-Romain:MongoClient romain$
```



- ▶ **Mongoose**

ODM : Object Document Mapping, permet de communiquer avec Mongo avec des objets Entités

- ▶ **Installation**

```
npm install mongoose --save
```

- ▶ **Schema**

Mongo permet l'absence de schéma, ce qui est peu recommandable dans une utilisation sous la forme d'entité. Mongoose réintroduit ce concept.



► Création d'un Schéma

```
var mongoose = require('mongoose');

var contactSchema = mongoose.Schema({
  firstName: String,
  lastName: String,
});

var Contact = mongoose.model('contact', contactSchema);
```

► Accès aux données depuis le schéma

```
mongoose.connect('mongodb://localhost/addressbook');
var db = mongoose.connection;
db.on('error', console.error.bind(console, 'connection error:'));
db.once('open', function (callback) {
  var contacts = Contact.find(function (err, contacts) {
    if (err) return console.error(err);
    reply({data: contacts});
  });
});
```



Modules JavaScript



▶ JavaScript inventé en 1995 par Netscape

Objectif : créer des interactions côté client, après chargement de la page

Exemple de l'époque :

- Menu en rollover (image ou couleur change au survol)
- Validation de formulaire

▶ JavaScript aujourd'hui

- Permet la création d'application front-end, back-end, en ligne de commande, application de bureau
- Ces applications peuvent contenir plusieurs centaines de milliers de lignes de codes
- Il faut faciliter le travail collaboratif, en plusieurs fichiers et en limitant les risques de conflit



► Immediately-invoked function expression (IIFE)

```
// jquery-button.js
(function($, global) {
  'use strict';

  function MonBouton(options) {
    this.options = options || {};
    this.value = options.value || 'Valider';
  }

  MonBouton.prototype.creer = function(container) {
    $(container).append('<button>'+this.value+'</button>');
  };

  global.MonBouton = MonBouton;
})(jQuery, window);
```

► Une fonction anonyme appelée immédiatement

- Limite la portée des variables
- Permet de renommer localement des dépendances



► Utilisation

```
<!DOCTYPE html>
<html lang="fr">
<head>
  <meta charset="UTF-8">
  <title>Exemple</title>
</head>
<body>
  <div id="container"></div>
  <script src="http://code.jquery.com/jquery-1.11.3.min.js"></script>
  <script src="jquery-button.js"></script>
  <script>
    var button = new MonBouton({
      value: 'Cliquez ici'
    });

    button.creer('#container');
  </script>
</body>
</html>
```

► Inconvénients

- L'ordre d'inclusion des scripts doit être connu (ici jQuery avant jquery-button)
- Les modules reçoivent leur dépendances via des variables globales (jQuery, window)
- Les modules exposent leur code via des variables globales (global.MonBouton)



► Modules YUI

Yahoo User Interface library (plus maintenue depuis mi-2014)

Première bibliothèque à introduire la notion de modules

<http://yuilibrary.com/yui/docs/yui/create.html>

```
// yui-button.js
YUI().add('mon-bouton', function (Y) {
    'use strict';

    function MonBouton(options) {
        this.options = options || {};
        this.value = options.value || 'Valider';
    }

    MonBouton.prototype.creer = function(container) {
        Y.one(container).append('<button>'+this.value+'</button>')
    };

    Y.MonBouton = MonBouton;
}, '0.0.1', {
    requires: ['node']
});
```

- Un module YUI décrit ses dépendances (requires: ['node'] pour accéder aux méthodes on et append)
- Pas d'utilisation de variables globales



► Utilisation

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Exemple</title>
</head>
<body>
  <div id="container"></div>
  <script src="http://yui.yahooapis.com/3.18.1/build/yui/yui-min.js"></script>
  <script>
    YUI({
      modules: {
        'mon-bouton': 'yui-button.js'
      }
    }).use('mon-bouton', function (Y) {
      var bouton = new Y.MonBouton({
        value: 'Cliquez ici'
      });

      bouton.creer('#container');
    });
  </script>
</body>
</html>
```

- Le fichier yui-button.js est inclus automatiquement
- Pas besoin de connaître l'ordre d'inclusion des dépendances





► CommonJS

Projet visant à créer des API communs pour du développement JavaScript hors navigateur (console, GUI...)

Exemple : standardiser l'accès aux fichiers

Le projet propose une norme pour le chargement de modules utilisé entre autre par Node.js

<http://www.commonjs.org/specs/modules/1.0/>

► Création d'un module

```
// calculatrice.js
exports.ajouter = function(nb1, nb2) {
  return Number(nb1) + Number(nb2);
};
```

- Les modules communs JS exposent à l'intérieur d'un module une variable exports de type object (et qui peut être écrasée si besoin)



► Utilisation

```
// main.js  
var calc = require('./Calcullette');  
  
console.log(calc.ajouter(2, 3)); // 5
```

- CommonJS propose une méthode `require` pour le chargement de modules, dont le retour correspond à la variable `exports`
- Cependant CommonJS ne s'applique pas au navigateur où le chargement de fichiers se fait via la balise `script`



- ▶ **Browserify**

Permet de charger des modules CommonJS côté client.

- ▶ **Installation :**

`npm install -g browserify`

- ▶ **Transformation en code client :**

`browserify main.js > calculette-browser.js`

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title></title>
</head>
<body>
  <script src="calculette-browser.js"></script>
</body>
</html>
```



- ▶ **Asynchronous Module Definition**

CommonJS ne permettant pas d'exécuter de charger des modules côté client, AMD est né.

- ▶ **RequireJS**

Plusieurs bibliothèques permettent de charger des modules AMD, RequireJS est la plus connue.

<http://requirejs.org/>

- ▶ RequireJS définit 2 fonctions globales `require` et `define`. `define` permet de définir un module, `require` est le point d'entrée de l'application.



Modules JavaScript



```
// number-converter.js
define(function() {
  var exports = {};

  exports.convert = function(nb) {
    return Number(nb);
  };

  return exports;
});
```

```
// calculette.js
define(['number-converter'], function(numberConverter) {
  var exports = {};

  exports.ajouter = function(nb1, nb2) {
    return numberConverter.convert(nb1) + numberConverter.convert(nb2);
  };

  return exports;
});
```

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title></title>
</head>
<body>
  <script src="bower_components/requirejs/require.js"></script>
  <script>
    require(['calculette'], function(calc) {
      console.log(calc.ajouter(2, 3)); // 5
    });
  </script>
</body>
</html>
```




- ▶ **ECMAScript 2015 / ECMAScript 6**

La nouvelle version de JavaScript prévoit une syntaxe pour l'utilisation de module. A l'heure actuelle (juillet 2015), ni les navigateurs ni Node.js ou io.js ne supportent cette syntaxe.

- ▶ **Babel / Traceur**

Babel et Traceur sont des bibliothèques qui permettent de transpiler du code ES6 en ES5 et ainsi l'utiliser sur les moteurs actuels.

- ▶ **Installation :**

`npm install -g babel`

- ▶ **Utilisation (toutes les sources du répertoires src vers le répertoire dist) :**

`babel src --out-dir dist/`

Modules JavaScript



```
// src/number-converter.js
var exports = {};

exports.convert = function(nb) {
  return Number(nb);
};

export default exports;
```

```
// src/calcullette.js
import numberConverter from './number-converter';

var exports = {};

exports.ajouter = function(nb1, nb2) {
  return numberConverter.convert(nb1) + numberConverter.convert(nb2);
};

export default exports;
```

```
// src/main.js
import calc from './calcullette';

console.log(calc.ajouter(2, 3)); // 5
```



► Universal Module Definition

L'objectif d'UMD est de proposer des modules compatibles CommonJS, AMD ou en utilisant des variables globales si le contexte ne permet pas d'utiliser les 2 précédents.

<https://github.com/umdjs/umd>

```
// number-converter.js
(function (root, factory) {
  if (typeof exports === 'object') {
    // CommonJS
    module.exports = factory();
  } else if (typeof define === 'function' && define.amd) {
    // AMD
    define(function () {
      return (root.numberConverter = factory());
    });
  } else {
    // Global Variables
    root.numberConverter = factory();
  }
})(this, function () {
  var exports = {};

  exports.convert = function(nb) {
    return Number(nb);
  };

  return exports;
});
```

```
// calculette.js
(function (root, factory) {
  if (typeof exports === 'object') {
    // CommonJS
    module.exports = factory(require('./number-converter'));
  } else if (typeof define === 'function' && define.amd) {
    // AMD
    define(['./number-converter'], function (numberConverter) {
      return (root.calculette = factory(numberConverter));
    });
  } else {
    // Global Variables
    root.calculette = factory(root.numberConverter);
  }
})(this, function (numberConverter) {
  var exports = {};

  exports.ajouter = function(nb1, nb2) {
    return numberConverter.convert(nb1) +
    numberConverter.convert(nb2);
  };

  return exports;
});
```



- ▶ **System.js**

System.js est un loader universel qui sait charger des modules CommonJS, AMD, ES6 et IIFE dans les navigateurs et sous node.js

<https://github.com/systemjs/systemjs>



Frameworks HTML/CSS/JS



- ▶ Popularisé lorsque Twitter a proposé sa bibliothèque UI en open source sous le nom de Bootstrap en 2011
- ▶ Unifie et accélère le développement, la majeure partie du CSS est déjà développée
- ▶ Inverse les responsabilités : le HTML fait la mise en forme en s'intégrant à un CSS existant



- ▶ Bootstrap
Créé par Twitter



- ▶ Open Source depuis 2011
- ▶ Projet le plus populaire sur GitHub
Contributeurs : 658 - Watches : 5092 - Stars 83109 - Forks 33538 (juillet 2015)
- ▶ Ecrit avec jQuery, Less, QUnit, Grunt...
- ▶ Documentation
<http://getbootstrap.com>
- ▶ Support : IE8 avec HTML5 shim et Respond.js

Frameworks HTML/CSS/JS



- Téléchargement :
<https://github.com/twbs/bootstrap/archive/v3.3.5.zip>
- CDN
<https://www.bootstrapcdn.com>
- Git
git clone <https://github.com/twbs/bootstrap.git>
- Bower
bower install bootstrap
- npm
npm install bootstrap
- Meteor
meteor add twbs:bootstrap
- Composer
composer require twbs/bootstrap

Frameworks HTML/CSS/JS



```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <!-- The above 3 meta tags *must* come first in the head; any other head content must come *after* these tags -->
  <title>Bootstrap 101 Template</title>

  <!-- Bootstrap -->
  <link href="css/bootstrap.min.css" rel="stylesheet">

  <!-- HTML5 shim and Respond.js for IE8 support of HTML5 elements and media queries -->
  <!-- WARNING: Respond.js doesn't work if you view the page via file:// -->
  <!--[if lt IE 9]>
  <script src="https://oss.maxcdn.com/html5shiv/3.7.2/html5shiv.min.js"></script>
  <script src="https://oss.maxcdn.com/respond/1.4.2/respond.min.js"></script>
  <![endif]-->
</head>
<body>
<h1>Hello, world!</h1>

<!-- jQuery (necessary for Bootstrap's JavaScript plugins) -->
<script src="https://ajax.googleapis.com/ajax/libs/jquery/1.11.3/jquery.min.js"></script>
<!-- Include all compiled plugins (below), or include individual files as needed -->
<script src="js/bootstrap.min.js"></script>
</body>
</html>
```

- Nécessite jQuery + HTML5 shim et Respond.js (IE8)



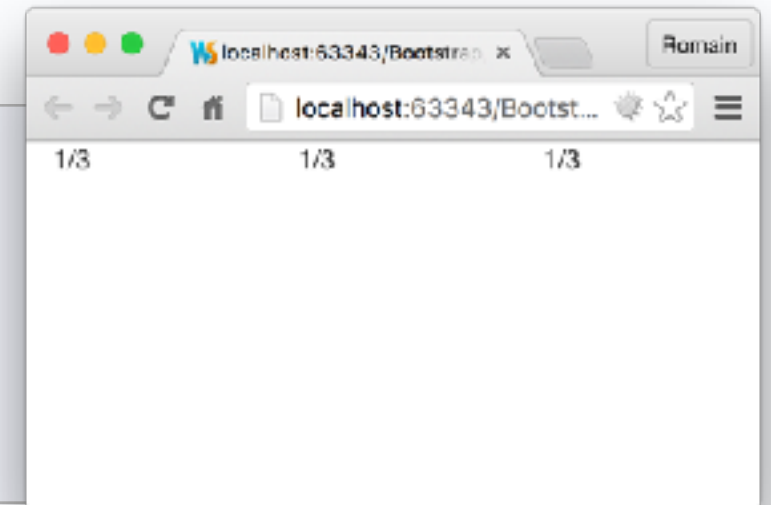
- Mise en forme de balises existantes (bouton, formulaires...)
- Inclus Normalize.css
- Composants mis en forme :
 - Container
 - Grid system
 - Typography
 - Code
 - Tables
 - Forms
 - Buttons
 - Images
 - Helper classes
 - Responsive utilities



► Le Grid System

Bootstrap contient un composant qui permet de faciliter la mise en page en colonne. Ces colonnes peuvent se transformer en ligne automatiquement si la largeur de fenêtre est insuffisante (Responsive Design)

```
<div class="container">
  <div class="row">
    <div class="col-sm-4">1/3</div>
    <div class="col-sm-4">1/3</div>
    <div class="col-sm-4">1/3</div>
  </div>
</div>
```



- Conteneur : classe container (fixed-design) ou container-fluid (fluid-design)
- Ligne : class row
- Colonnes : taille exprimée en 1/12, à partir de quelle largeur de fenêtre la colonne s'affiche, (ligne sinon) : xs (< 768px), sm (>=768px), md (>=992px), lg (>=1200px)



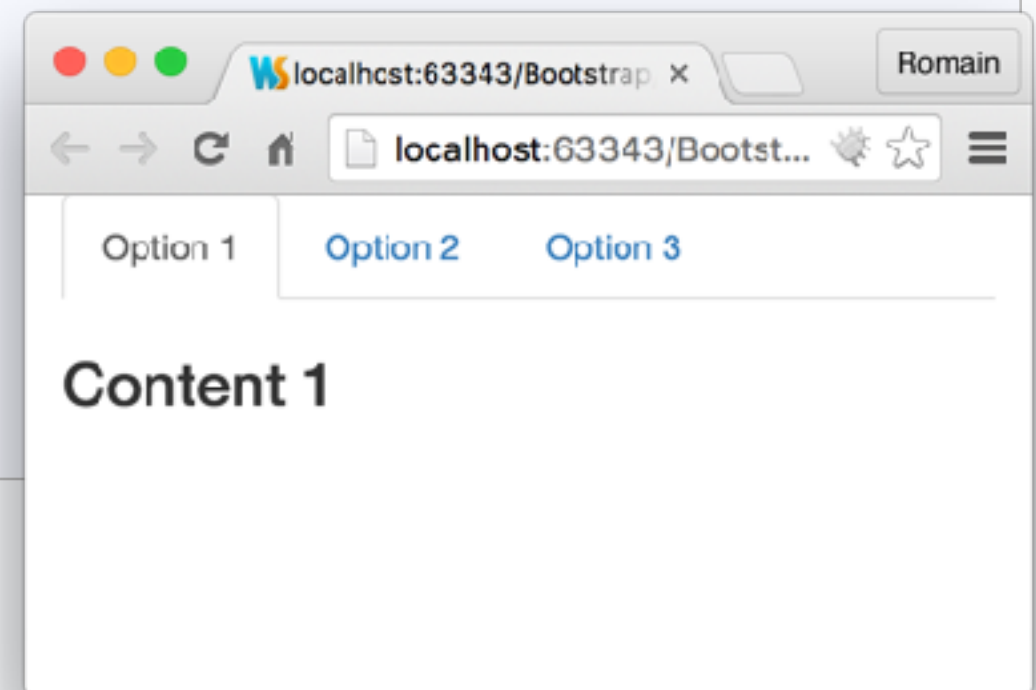
```
<form>
  <div class="form-group">
    <label for="exampleInputEmail">Email address</label>
    <input type="email" class="form-control" id="exampleInputEmail" placeholder="Email">
  </div>
  <div class="form-group">
    <label for="exampleInputPassword1">Password</label>
    <input type="password" class="form-control" id="exampleInputPassword1"
placeholder="Password">
  </div>
  <div class="form-group">
    <label for="exampleInputFile">File input</label>
    <input type="file" id="exampleInputFile">
    <p class="help-block">Example block-level help text here.</p>
  </div>
  <div class="checkbox">
    <label>
      <input type="checkbox"> Check me out
    </label>
  </div>
  <button type="submit" class="btn btn-default">Submit</button>
</form>
```

A screenshot of a web browser window displaying the rendered HTML form. The browser's address bar shows 'localhost:63343/Bootstrap'. The form contains an 'Email address' input field with the placeholder 'Email', a 'Password' input field with the placeholder 'Password', a 'File input' section with a file selection button and the text 'Aucun fichier choisi', a 'Check me out' checkbox, and a 'Submit' button. The browser window also shows a 'Romain' user profile in the top right corner.



```
<div class="container">
  <ul class="nav nav-tabs ">
    <li class="active"><a href="#option1" data-toggle="tab">Option 1</a></li>
    <li><a href="#option2" data-toggle="tab">Option 2</a></li>
    <li><a href="#option3" data-toggle="tab">Option 3</a></li>
  </ul>

  <div class="tab-content">
    <div class="tab-pane active" id="option1">
      <h3>Content 1</h3>
    </div>
    <div class="tab-pane" id="option2">
      <h3>Content 2</h3>
    </div>
    <div class="tab-pane" id="option3">
      <h3>Content 3</h3>
    </div>
  </div>
</div>
```





▸ HTML + CSS de composants plus haut niveau

▸ Composants :

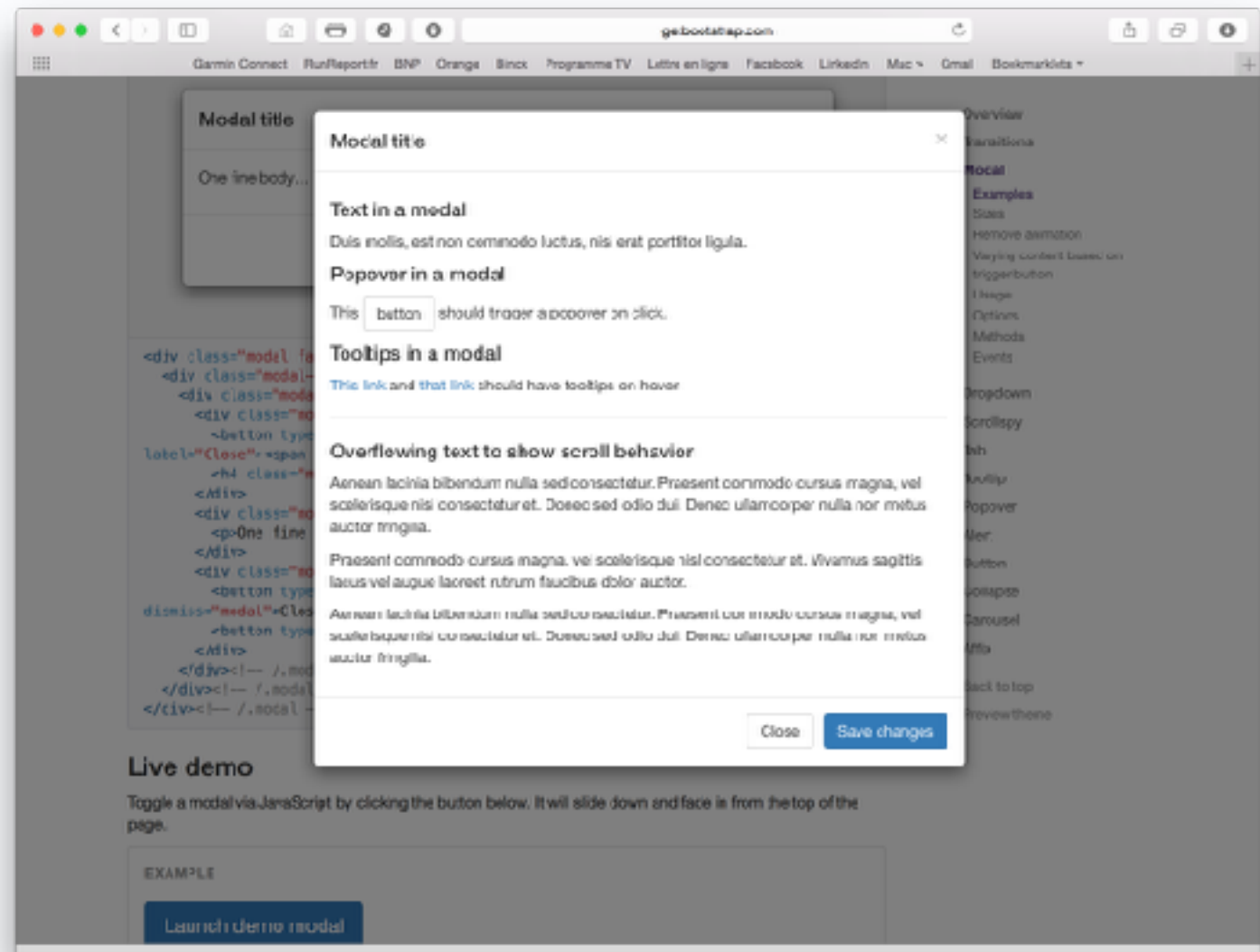
- Glyphicons
- Dropdowns
- Button groups
- Button dropdowns
- Input groups
- Navs
- Navbar
- Breadcrumbs
- Pagination
- Labels
- Badges
- Jumbotron
- Page header
- Thumbnails
- Alerts
- Progress bars
- Media object
- List group
- Panels
- Responsive embed
- Wells



► Plugins pour jQuery

► Composants :

- Transitions
- Modal
- Dropdown
- Scrollspy
- Tab
- Tooltip
- Popover
- Alert
- Button
- Collapse
- Carousel
- Affix





- ▶ Foundation
2e framework HTML/CSS/JS sur GitHub



- ▶ Documentation
<http://foundation.zurb.com>
- ▶ Stats Github :
Contributeurs : 705 - Watches : 1431 - Stars 20611 - Forks 4394 (juillet 2015)
- ▶ Ecrit avec jQuery, SASS, Jasmine, Grunt...
- ▶ Support : IE9+



- ▶ Semantic UI
3e framework HTML/CSS/JS sur GitHub
- ▶ Documentation
<http://semantic-ui.com>
- ▶ Stats Github :
Contributeurs : 111 - Watches : 994 - Stars 19191 - Forks 2129 (juillet 2015)
- ▶ Ecrit avec jQuery, LESS, Jasmine, Gulp...
- ▶ Support : Last 2 Versions FF, Chrome, IE 10+, Safari Mac





AngularJS



- ▶ Un framework pour structurer les applications web
- ▶ HTML est un langage déclaratif permettant de créer des pages statiques, AngularJS des applications dynamiques
- ▶ Imaginé pour des applications CRUD
- ▶ Créé en 2009 par Miško Hevery et Adam Abronsw puis repris par Google
- ▶ Licence MIT



- ▶ **Téléchargement**

<https://code.angularjs.org/>

- ▶ **CDN**

<https://ajax.googleapis.com/ajax/libs/angularjs/1.4.6/angular.min.js>

- ▶ **Bower**

bower install angular

- ▶ **NPM**

npm install angular



- ▶ **Live Reload**

npm install -g live-server
live-server

- ▶ **Express, Apache, nginx...**



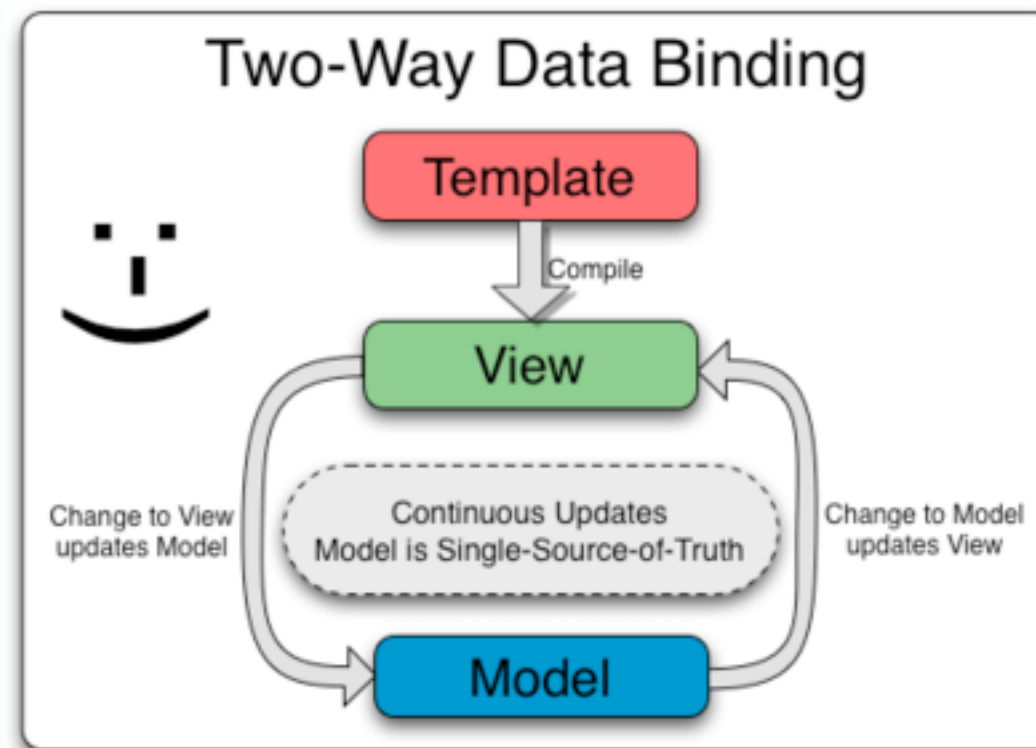
- ▶ **Model View ViewModel (MVVM)**

Design Pattern introduit par Microsoft en 2005 dans Windows Presentation Foundation et Silverlight

- ▶ **Facilite le développement d'interface graphique**

- ▶ **Two-way data binding**

Le model peut mettre à jour la vue, la vue peut mettre à jour le model.





```
<!DOCTYPE html>
<html ng-app>
<head>
  <script src="bower_components/angular/angular.js"></script>
</head>
<body>
<div>
  <label>Name:</label>
  <input type="text" ng-model="yourName" placeholder="Enter a name here">
  <h1>Hello {{yourName}}!</h1>
</div>
</body>
</html>
```

► Directives

Balises ou attributs HTML qui compilent en JS (ex : ng-app et ng-model)

► ng-app

Directive qui déclare la racine de l'application (en général <html> ou <body>)

► ng-model

Directive qui lie le contenu d'une balise input, select ou textarea à une variable

► {{yourName}}

Lie cette partie de la vue à la variable yourName (peut être une expression)



- ▶ Le contrôleur contient du code JS qui peut être rendu accessible dans la vue grâce au service \$scope
- ▶ Ici la variable contacts et la fonction ajouter deviennent disponible dans la vue

```
var addressBookModule = angular.module('addressBookModule', []);

addressBookModule.controller('AddressBookCtrl', ['$scope', function($scope) {
  $scope.contacts = [{
    prenom: 'Thierry',
    nom: 'Henry'
  }, {
    prenom: 'Zinédine',
    nom: 'Zidane'
  }];

  $scope.ajouter = function() {
    $scope.contacts.push({prenom: $scope.prenomSaisi, nom: $scope.nomSaisi});
  };
}]);
```




- Déclaration d'un Service pour accéder aux données

```
var phonecatServices = angular.module('phonecatServices', ['ngResource']);

phonecatServices.factory('Phone', ['$resource',
  function($resource){
    return $resource('data/:phoneId.json', {}, {
      query: {method: 'GET', params: {phoneId: 'phones'}, isArray: true}
    });
  }
]);
```

- Injection dans le contrôleur et utilisation du Model

```
phonecatControllers.controller('PhoneListCtrl', ['$scope', 'Phone',
  function($scope, Phone) {
    $scope.phones = Phone.query();
    $scope.orderProp = 'age';
  }
]);
```



▸ ngApp

La directive ngApp désigne la balise racine de l'application en général <body> ou <html>. Elle permet également de spécifier un module qui deviendra le module racine de l'application.

```
<html ng-app="appFilmotheque">
```

▸ ngController

La directive ngController permet d'associer le contrôleur à la vue (peut également se faire via des routes)

```
<body ng-controller="FilmController">
```

▸ ngModel

La directive ngModel lie une balise input, select ou textarea à une propriété du scope.

```
<input type="text" ng-model="film.titre">
```



- ▶ **ngIf**

Permet de créer l'élément du DOM selon une condition

```
<div class="col-sm-9" ng-if="showFilm">
  <h2>{{showFilm.titre}}</h2>
  ...
</div>
```

- ▶ **ngRepeat**

Permet de répéter l'élément du DOM pour chaque élément d'un tableau

```
<li role="presentation" ng-repeat="film in films">
  <a href="#" ng-click="show(film)">{{film.titre}}</a>
</li>
```

- ▶ **ngClick, ngChange, ngSubmit, ng...**

Permet de lier à un événement du DOM

```
<li><a href="#" ng-click="showFilm = null">Ajouter</a></li>
```



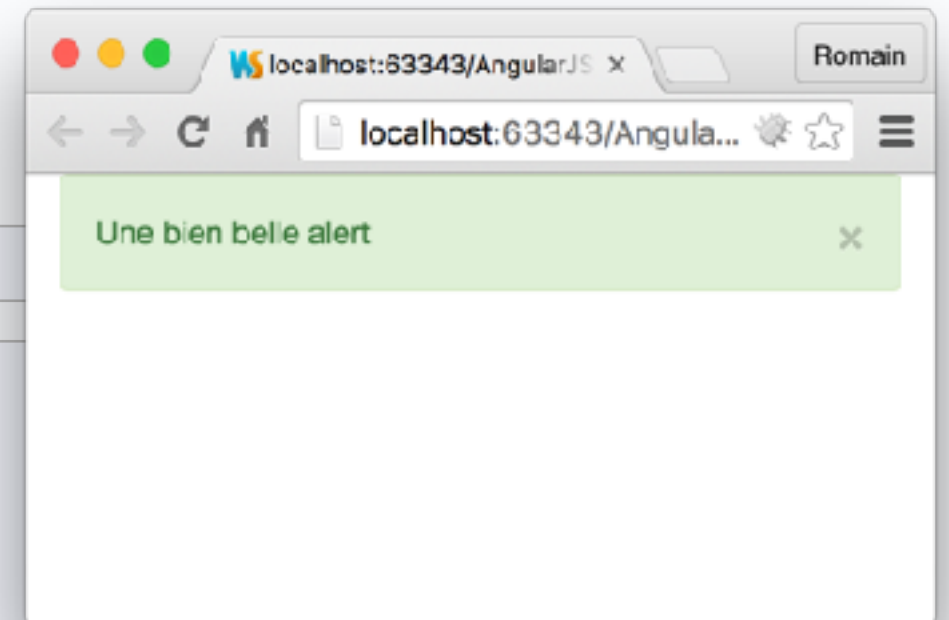
- ▶ Permet la création de balises ou d'attributs personnalisés pour simplifier le développement

Exemple : une alert bootstrap

```
<btp-alert>Une bien belle alert</btp-alert>
```

```
alertDirectives.directive('btpAlert', function() {  
    return {  
        transclude: true,  
        templateUrl: 'js/btp-alert.html'  
    };  
});
```

```
<div class="alert alert-success">  
    <button type="button" class="close" data-dismiss="alert" aria-label="Close">  
        <span aria-hidden="true">&times;</span>  
    </button>  
    <div ng-transclude></div>  
</div>
```



- ▶ ngTransclude permet de spécifier le point d'insertion du contenu de <btp-alert>



► Route

Angular permet la mise en place de routes, des URLs configurées permettront donc d'accéder à certains contrôleurs

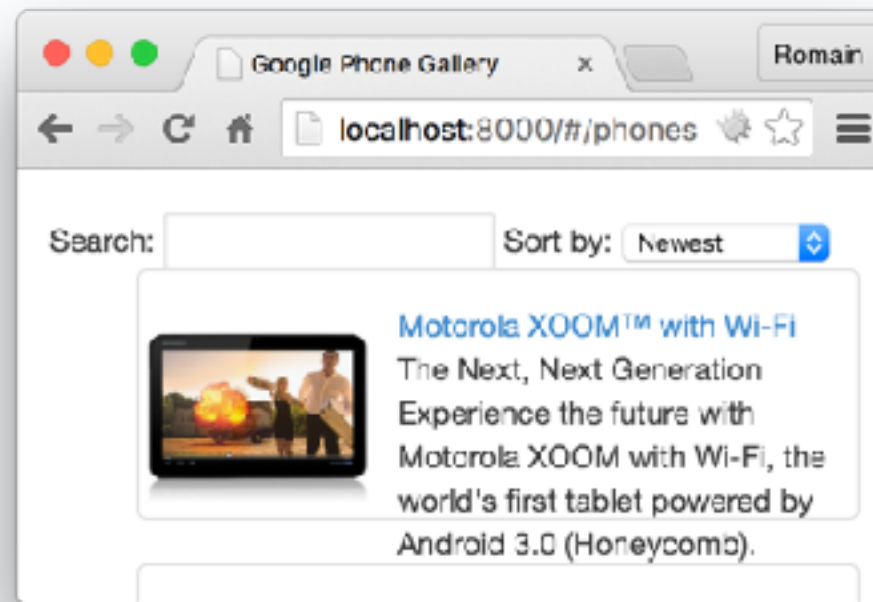
```
var phonecatApp = angular.module('phonecatApp', [  
  // ...  
]);  
  
phonecatApp.config(['$routeProvider', '$locationProvider',  
  function($routeProvider, $locationProvider) {  
    $locationProvider.html5Mode(true);  
  
    $routeProvider.  
      when('/phones', {  
        templateUrl: 'partials/phone-list.html',  
        controller: 'PhoneListCtrl'  
      }).  
      when('/phones/:phoneId', {  
        templateUrl: 'partials/phone-detail.html',  
        controller: 'PhoneDetailCtrl'  
      }).  
      otherwise({  
        redirectTo: '/phones'  
      });  
  });  
});
```

```
<!doctype html>  
<html lang="en" ng-app="phonecatApp">  
<head>  
  <meta charset="utf-8">  
  <title>Google Phone Gallery</title>  
  <base href="/">  
  ...  
</head>  
<body>  
  <div class="view-container">  
    <div ng-view></div>  
  </div>  
</body>  
</html>
```

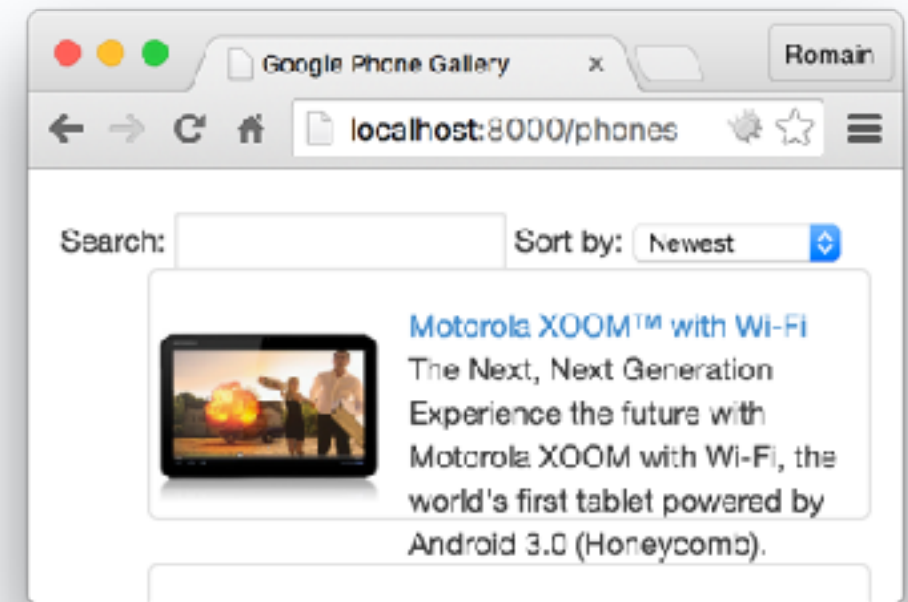


► 2 modes

Hash



HTML5



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

app.use('/bower_components', express.static(__dirname + '/app/bower_components'));
app.use('/css', express.static(__dirname + '/app/css'));
app.use('/img', express.static(__dirname + '/app/img'));
app.use('/js', express.static(__dirname + '/app/js'));
app.use('/partials', express.static(__dirname + '/app/partials'));
app.use('/data', express.static(__dirname + '/app/data'));

app.all('/*', function(req, res, next) {
  // Just send the index.html for other files to support HTML5Mode
  res.sendFile('app/index.html', { root: __dirname });
});

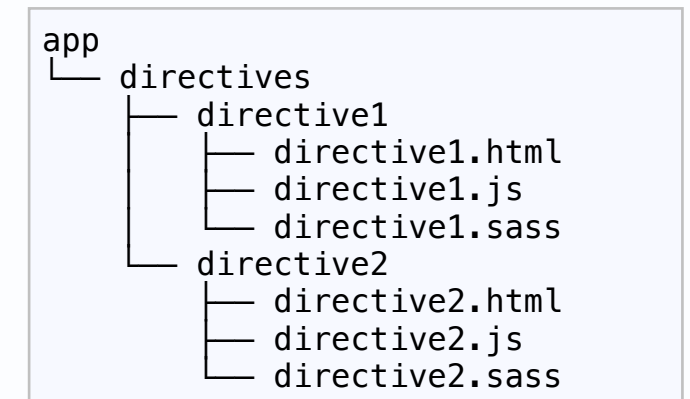
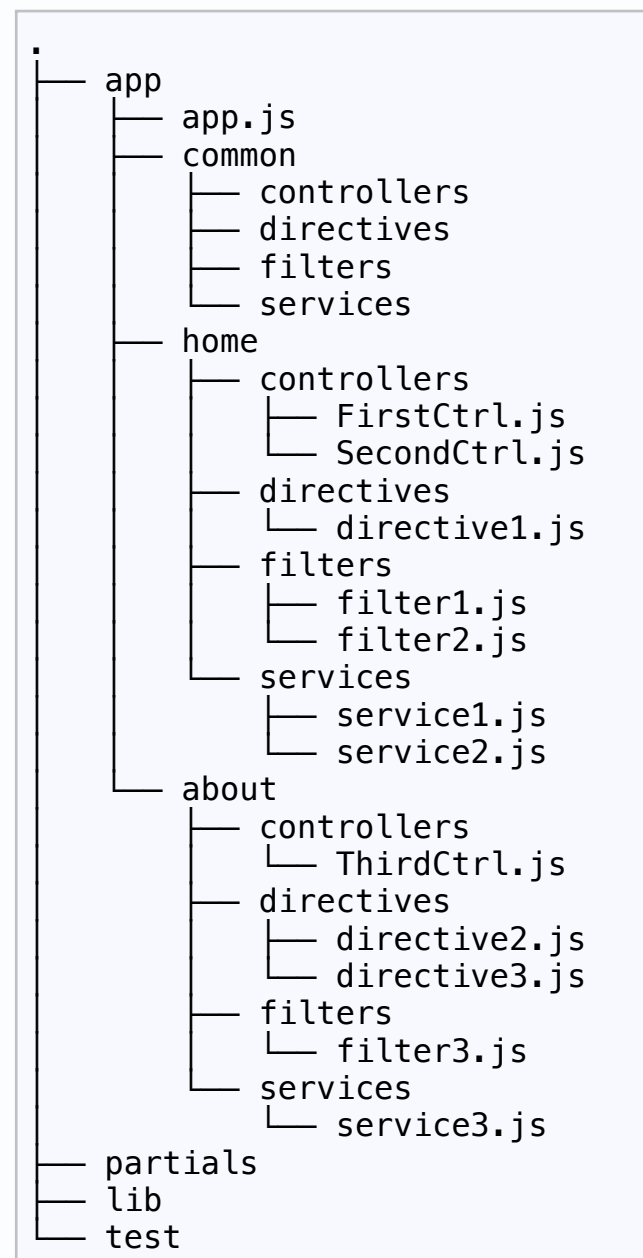
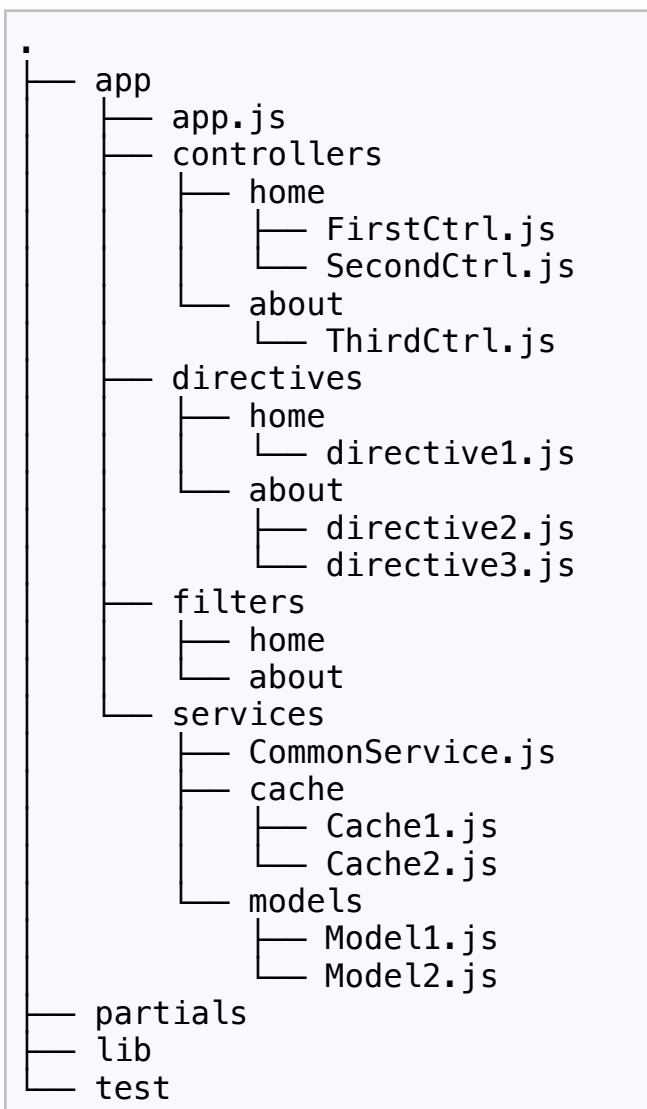
app.listen(8000); //the port you want to use
```



- ▶ Extraits de :

<https://github.com/mgechev/angularjs-style-guide/blob/master/README-fr-fr.md>

- ▶ Arborescence :





- ▶ **MEAN**

MEAN signifie Mongo Express Angular Node.

Des développeurs ont développé un framework qui utilise ces technologies de manière orientée.

- ▶ **Installation**

```
npm install -g mean-cli
```

- ▶ **Création d'un squelette d'application**

```
mean init Helloworld
```

```
cd Helloworld
```

```
npm install
```




- **Les packages**

Le code de MEAN s'organise en packages, qui contiennent à la fois du code client (Angular) et serveur (Express).

- **System**

Package principal, création des pages, du layout, du menu. Du moteur de template, des fichiers statiques, les routes client et serveur.

- **Users**

Model utilisateur, pages de login et d'inscription.

- **Theme**

Theme de l'application



- ▶ **Package System**

Le package principal, à éditer avec le code de votre applica

- ▶ **app.js**

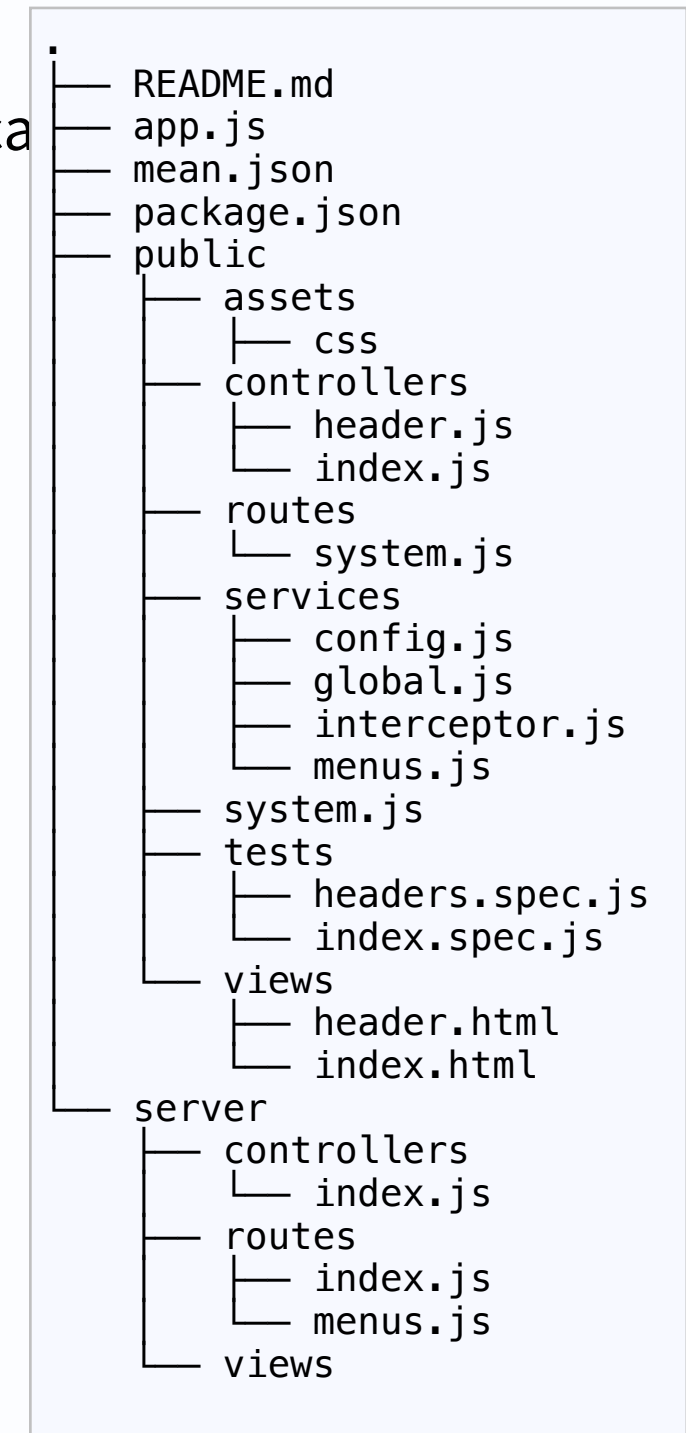
Configuration du package

- ▶ **public/**

Application cliente AngularJS

- ▶ **server/**

Application serveur Express





- ▶ Injection de dépendance

Les packages peuvent s'imbriquer les uns dans les autres simplement en déclarant l'un dans le callback de l'autre (MEAN fait de l'introspection de méthode)

- ▶ Exemple

users/app.js

```
MeanUser.auth = require('./authorization');  
require('./passport')(passport);  
  
mean.register('auth', MeanUser.auth);
```

system/app.js

```
SystemPackage.register(function(app, auth, database) {  
    // ...  
    return SystemPackage;  
});
```



▸ Dépendances

Les dépendances de module AngularJS se définissent au niveau du package

```
SystemPackage.angularDependencies(['mean-factory-interceptor']);
```

▸ Module

Les modules portent le nom du package et se trouve dans le répertoire public

```
'use strict';

angular.module('mean.system', ['ui.router', 'mean-factory-interceptor'])
  .run(['$rootScope', function($rootScope) {
    $rootScope.$on('$stateChangeSuccess', function(event, toState, toParams, fromState,
fromParams){
      var toPath = toState.url;
      toPath = toPath.replace(new RegExp('/', 'g'), '');
      toPath = toPath.replace(new RegExp(':', 'g'), '-');
      $rootScope.state = toPath;
      if($rootScope.state === '' ) {
        $rootScope.state = 'firstPage';
      }
    });
  }]);
;

```



- ▶ CSS

Les fichiers CSS d'un package s'ajoute aux autres avec la méthode

```
SystemPackage.aggregateAsset('css', 'common.css');
```

- ▶ JS

Idem pour les fichiers JavaScript

```
MeanUser.aggregateAsset('js', '../lib/angular-jwt/dist/angular-jwt.min.js', {  
    absolute: false,  
    global: true  
});
```



- ▶ Côté serveur

Elles se définissent dans server/routes

```
app.route('/api/logout')  
  .get(users.signout);  
app.route('/api/users/me')  
  .get(users.me);
```

- ▶ Côté client

Elles se définissent avec ui-route dans public/routes

```
// states for my app  
$meanstateProvider  
  .state('auth', {  
    url: '/auth',  
    abstract: true,  
    templateUrl: 'users/views/index.html'  
  })  
  .state('auth.login', {  
    url: '/login',  
    templateUrl: 'users/views/login.html',  
    resolve: {  
      loggedIn: function(MeanUser) {  
        return MeanUser.checkLoggedOut();  
      }  
    }  
  })  
})
```