MEAN

- Stack de développement javascript composée :
  - de 2 frameworks
    - ExpressJS : MVC côté serveur tournant sur NodeJS.
    - AngularJS : MVVM coté navigateur
  - d'une base de données orientée documents.
    - MongoDB

Javascript : Rappel – 1
Quelques généralités

Langage

- interprété
- orienté objet (héritage prototypal)
- Fonctionnel (les fonctions sont des objets de première classe : objets à part entières, elles peuvent être affectées à des variables, être passées en paramètres d'autres fonctions etc...)
- faiblement typé
- Mono thread
- **Asynchrone, non bloquant** : une fonction asynchrone (typiquement lorsque elle-ci consiste à récupérer des données qui se trouvent en dehors de l'environnement d'exécution: dans des fichiers, dans une base de données, sur un serveur distant etc.), rend immédiatement la main après qu'elle ait été appelée. La fonction appelante poursuit alors sont exécution sans tenir compte du résultat de l'opération.

Javascript : Rappel -2
Héritage prototypal

- **Héritage et création d'objet**

  Toute fonction en javascript est un objet qui possède une propriété « **prototype** » mise en jeu lorsque la fonction est employée comme constructeur (appel précédé du mot clef **new**).Cette propriété est un objet qui possède en règle générale l'ensemble des methodes que le constructeur s'attend à partager. Le processus de création d'un objet construit via constructeur se déroule en plusieurs étapes:

  ex : var obj = new Constr() ;

  - Avant l'exécution de la fonction Constr(), un **objet** basic (de type Object) est automatiquement créé.
  - Cet objet est affecté à la variable **this** de la fonction Constr().
  - La propriété __**proto**__ de l'objet est fixée sur la propriété **prototype** de la fonction Constr().
  - Le corps de la fonction Constr() est exécuté.
  - L'objet est finalement retourné implicitement (affectée à la variable obj dans notre exemple)

  obj possède alors les propriétés qui lui ont été définies dans le corps de la fonction Constr() **ET** a accès via sa propriété __proto__ à celles définies dans la propriété **prototype** de la fonction Constr() => on dit que obj **hérite** des propriétés définies dans le prototype de Constr(),

  Lorsque l'on tente d'accéder à une propriété de l'objet obj, l'interpréteur vérifie si la propriété est directement affectée à l'objet, et dans le cas contraire, part à sa recherche en parcourant **la chaîne des prototypes** (recherche la propriété dans l'objet pointé par __proto__, puis dans __proto.__proto__ et ainsi de suite)

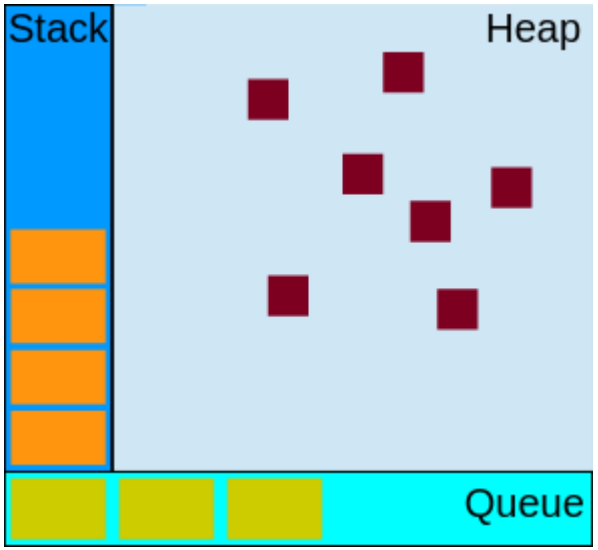
- **Héritage entre constructeur**

  Pour faire hériter un constructeur A d'un constructeur B :

  - On fait appel dans le constructeur A au constructeur B (B.call(this)).
  - On associe le prototype du constructeur A au prototype du constructeur B (A.prototype = Object.create(B.prototype)).

Javascript : Rappel – 2
Fonctionnement interne

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



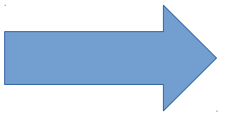
- La Stack représente la pile des appels de fonction en cours. La fonction qui se trouve au sommet de la pile est la fonction en cours d'exécution.
- La Queue représente la file des fonctions de rappel en attente d'exécution.
- La Heap (ou tas) représente l'espace mémoire ou sont stoquées les varaibles locales aux fonctions. En javascript, **chaque appel de fonction** créé un scope (on parle également de **closure**)**.** Lorsque la fonction rend la main, le scope qui lui a été associé est détruit, sauf si ses variables sont référencées par des fonctions en attente d'exécution

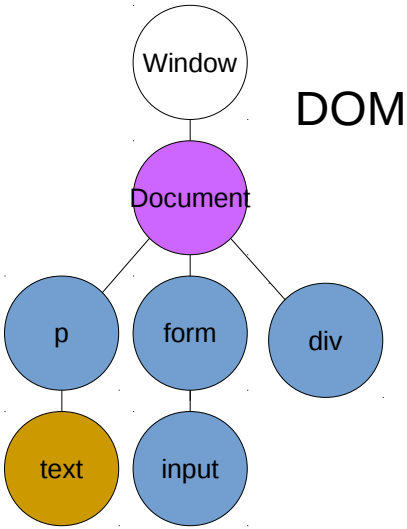
La Stack croit à mesure que les appels de fonctions s'imbriquent, et se vide lorsque les fonctions appelées rendent successivement la main. Lorsque la Stack est vide (la première fonction appelante de la série rend la main), la fonction qui se trouve en tête de file est extraite pour être empilée dans la Stack. Et le cycle recommence.

DOM - Rappel

HTML (texte)

```
<!Doctype>
	<head>
	</head>
	<body>
		<p>Hello</p>
		<form>
			<input type='input'>
		</form>
		<div></div>
		<script>
			document.querySelector('form').submit();
		</script>
	</body>
</html>
```

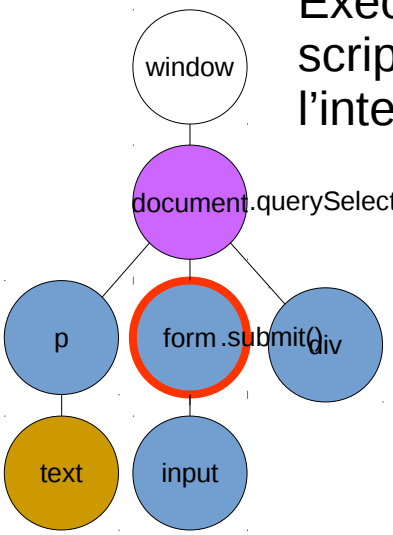




DOM (mémoire)

Le **moteur de rendu** du navigateur construit le DOM à partir de la page HTML et fait appel à l'interpréteur javascript qui exécute alors le code javascript. Le **moteur de rendu** expose à l'interpréteur javascript l'arbre DOM qu'il a construit.

Note : Le script de l'exemple précédent est « inliné », c'est à dire que son contenu apparaît intégralement dans le contenu de la page HTML. Un tel script s'exécute de manière synchrone, c'est à dire qu'il commence à s'exécuter lorsque le parseur HTML du moteur de rendu rencontre la balise <script>, et n'a accès qu'aux éléments du DOM qui l'ont précédé. Il en va autrement du chargement des scripts externes (référencés avec l'attribut src).



Exécution du script par l'interpréteur

Node - 1

Le **navigateur** est doté d'un **interpréteur** JavaScript et d'un **moteur de rendu** qui lui fourni le DOM (représentation en mémoire de la page HTML sous la forme d'un arbre).

Coté serveur, cette faculté à exécuter du code javascript est apportée par **NodeJS**. Il est composé d'un **interpréteur** Javascript et d'un ensemble de **modules** qui lui permettent d'accéder aux ressources système (socket TCP, système de fichier, information OS etc ...).

Node - 2

- Quelques modules intégrés à Node:
  - Http
  - Stream
  - Events
  - File System

  Ces modules sont des extensions intégrés à node qui permettent à l'interpréteur JavaScript de pouvoir communiquer avec l'OS sur lequel il s'exécute. Ils peuvent ainsi contenir, en plus du Javascript, des parties de code rédigés en C++ qui auront été préalablement compilés.

- Les bibliothèques sont des programmes écrits en javascript qui peuvent être basés (ou non) sur les modules fournis par node. L'installation d'une bibliothèque se fait via le gestionnaire de paquet npm qui accompagne node :

  npm install « bibliothèque »

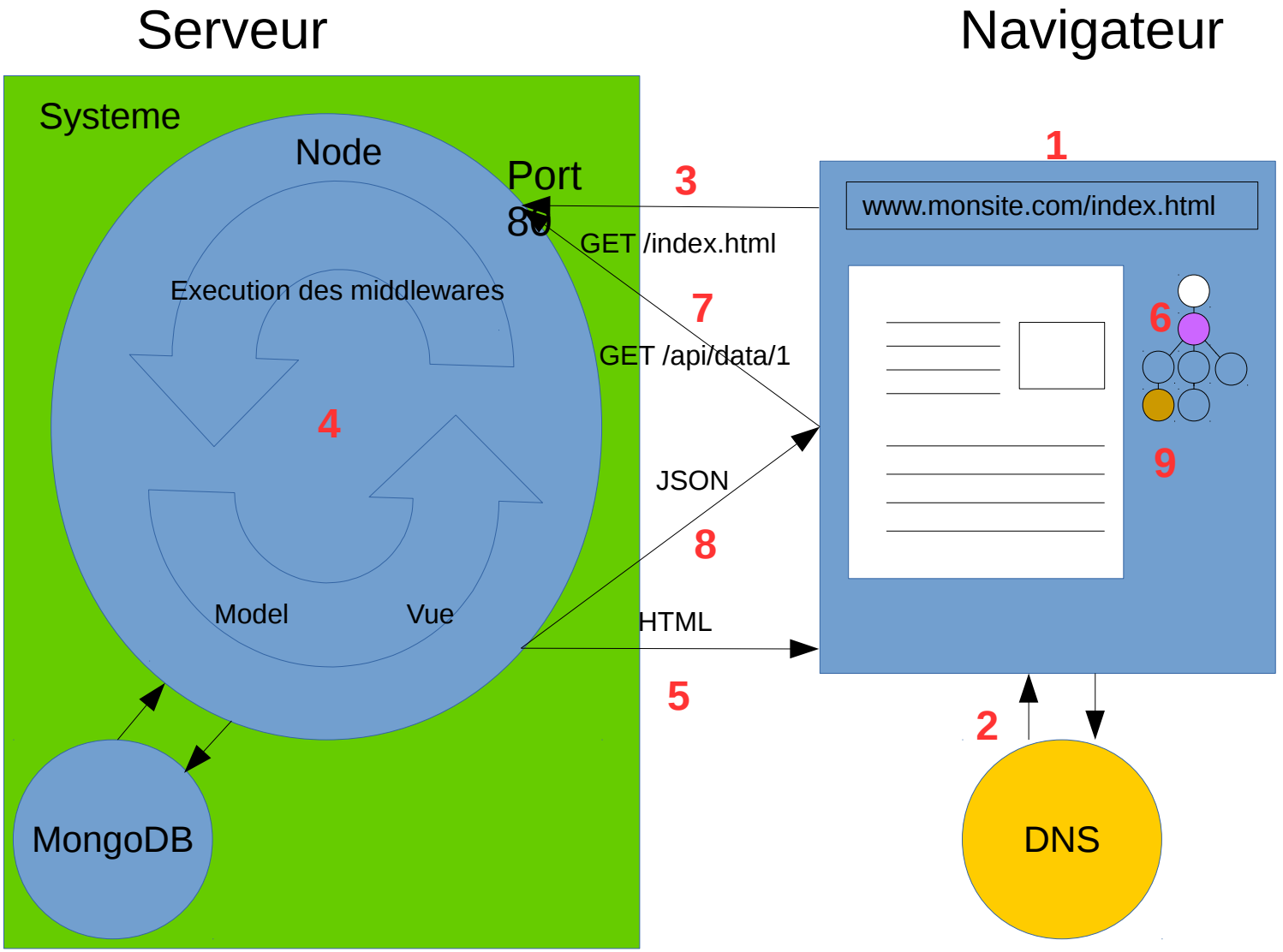
  L'exécution de la commande précédente créera un dossier node_modules dans le dossier courant. Ce dossier contiendra lui même le dossier de la bibliothèque demandée ainsi que ceux de ses dépendances.

  L'appel à la bibliothèque se fait de la façon suivante:

  var bibliotheque = require('bibliotheque')

La variable bibliothèque reçoit un objet qui possède toutes les méthodes définies par cette bibliothèque.

Stack MEAN : scénario - 1



Stack MEAN : scénario - 2

- **1** saisie de l'URL dans la barre d'adresse
- **2** requête DNS de résolution de nom : traduit www.site.com en adresse IP
- **3** requête HTTP en direction du serveur dont l'adresse IP vient d'être récupérée
- **4** Exécution de l'application basée sur Expresse. Expresse utilise le module 'http' de node qui permet d'instancier un serveur web.

  On distingue 2 temps:
  - Étape d'initialisation : exécutée qu'une seule fois au lancement
    - Paramétrage d'expresse
    - Déclaration des middlewares
    - Déclaration des routes
  - Étape de fonctionnement : A chaque requête reçue
    - Execution à la chaîne des middlewares enregistrés lors de l'étape d'initialisation suivant l'ordre dans lequel ils ont été déclarés.
    - Routage
    - Exécution du contrôleur associé à la page demandée. Peut éventuellement s'appuyer sur une BDD.
    - Execution de la vue chargée de formater les données en HTML (ou JSON au cas 8)

  Cette architecture est communément appelée MVC (pour Model Vue Controler)

Stack MEAN : scénario - 3

- **5** retour de la page formatée en HTML au navigateur
- **6** construction du DOM et chargement d'AngularJS

  AngulaJS est un framework classé parmi les frameworks MVVM (Model Vue Vue Model). Cet acronyme met en évidence l'interaction et le couplage fort qu'il existe entre la couche Model et la Vue. Une modification du model (données) se répercute instantanément sur la vue (DOM) et inversement.

  Bien souvent, les applications Front-end vont inscrire les données persistantes ou vont aller chercher les données à afficher auprès d'un WEB service accédé via une API REST. Par conséquent, une partie de la configuration des routes et des contrôleurs d'express porteront sur la définition de cette API.
- **7** requête en direction de l'API Rest
- **8** retour JSON
- **9** mise à jour du DOM par Angular

Express JS

ExpressJS est un framework qui repose sur NodeJS (et ses différents modules internes http, event, path, url, stream...) qui vise à faciliter le développement web côté serveur en fournissant à l'application une architecture MVC.

Express JS

- Configuration
  - app.set('option', valeur)
  - app.enable('option')
  - Options : 'env', 'jsonp callback name', 'json replacer', 'case sensitive routing', 'strict routing', 'view engine', 'views', 'x-powered-by', …
- middlewares
- routes/controlleurs
- vues

MongoDB

- Base de données NoSQL (non relationnelle) au sein desquelles les données sont organisées en **collections** et **documents** (terminologie comparable aux tables et aux enregistrements des SGBDRs)
- Les données dans ce système sont
  - destructurées (schemaless)
  - dénormalisées (le système admet de la redondance d'information)

Initialement prévues pour le big data, les BDDs NoSQL se sont progressivement étendues aux applications travaillant sur des données « simples ».

Se distingue par ses performances et sa capacité à passer à l'échelle très facilement (« horizontal scaling »).