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** est automatiquement créé, dont la propriété **__proto__** fait référence à la propriété **prototype** de la fonction Constr().
  - Cet objet est affecté à la variable **this** de la fonction Constr().
  - Le corps de la fonction Constr() est exécuté.
  - L'objet est finalement retourné implicitement (affecté 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 récursif basée sur la référence __proto__ : recherche dans l'objet pointé par __proto__ puis dans __proto.__proto__ etc)

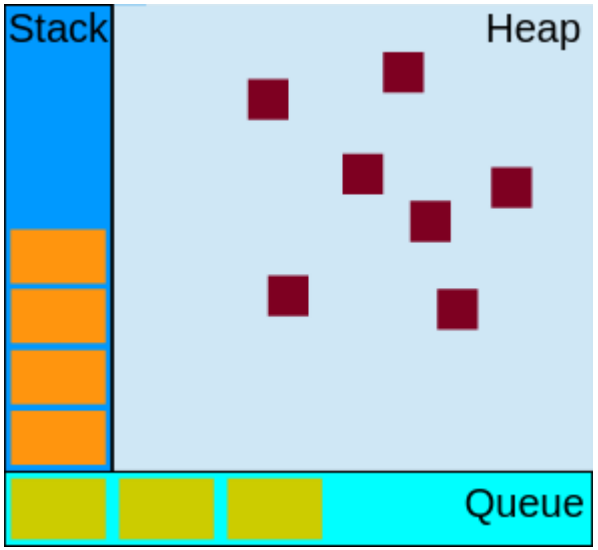
- **Héritage entre constructeur**

  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



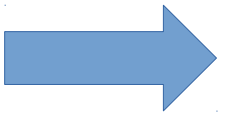
- La Stack représente la pile des appels de fonction en cours. La fonction qui se trouve au sommet de la pile est la fonction en cours d'exécution.
- La Queue représente la file des fonctions de rappel en attente d'exécution.
- La Heap (ou tas) représente l'espace mémoire ou sont stoquées les varaibles 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

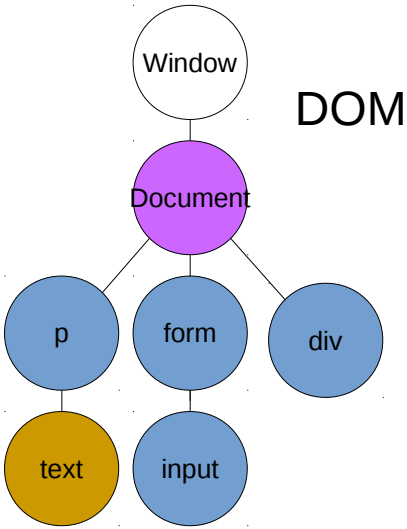
La Stack croit à mesure que les appels de fonctions s'imbriquent, et se vide lorsque les fonctions appelées rendent successivement la main. Lorsque la Stack est vide (la première fonction appelante de la série rend la main), la fonction qui se trouve en tête de file est extraite pour être empilée dans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

DOM - Rappel

HTML (texte)

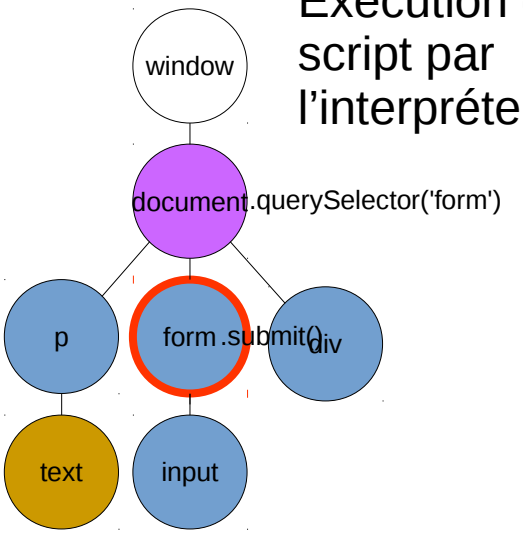
```
<!Doctype>
	<head>
	</head>
	<body>
		<p>Hello</p>
		<form>
			<input type='input'>
		</form>
		<div></div>
		<script>
			document.querySelector('form').submit();
		</script>
	</body>
</html>
```





DOM (mémoire)

Exécution du script par l'interpréteur



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 corps 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).

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 sur les modules fournis par node ou codé en JavaScript pure. L'installation d'une bibliothèque se fait via le gestionnaire de paquet npm qui accompagne node :

  npm install « bibliothèque »

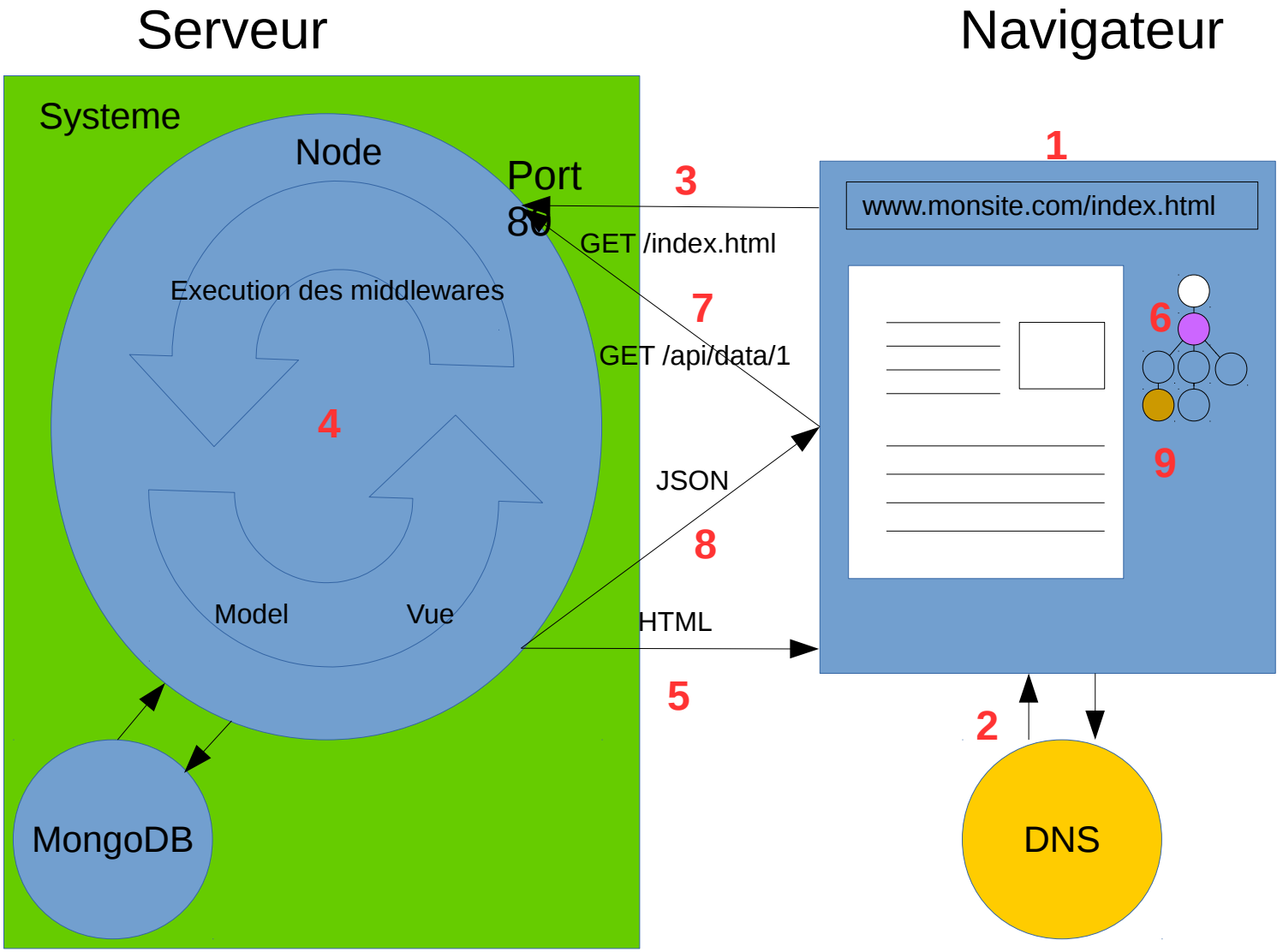
  L'exécution de la commande précédente créera un dossier node_modules dans le dossier courant. Ce dossier contiendra lui même le dossier de la bibliothèque demandée ainsi que ceux de ses dépendances.

  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 HTTP. 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 HTTP
- **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 : les middlewares (aussi appelés pre et post hook) sont des fonctions appelées en série durant l’exécution d'une fonction asynchrone
- 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 »).

Un ensemble de programmes

- Outils d'export :
  - mongoexport : au format JSON (texte)
  - mongodump : au format BSON (binaire)
- Outils d'import :
  - mongoimport : depuis un fichier au format json
  - Mongorestore : depuis un fichier au format binaire
- Outils statistiques
  - mongostat
- Convertisseur :
  - bsondump : convertit du BSON en JSON
- Serveur
  - mongod
- Client :
  - mongo : connu également sous le nom de mongoshell. C'est un interpréteur javascript capable de communiquer avec le serveur de base de données. Toutes les opérations en base de données sont réalisées à partir de l'objet **db**.

Travailler avec Node

- Mongoshell et NodeJS possèdent un point commun : ils fournissent tout deux un interpréteur javascript en ligne de commande (REPL) qu'il ne faut pas confondre. Mongo shell est uniquement conçu pour administrer une base de données mongodb.
- Travailler depuis node pour interagir le serveur mongod requiert l'installation d'un driver disponible sous forme de bibliothèque.
  - npm install mongodb
- En dehors des quelques raccourcis que propose le Mongo Shell, l'API de la bibliothèque mongodb sous Node est la même que celle de Mongo Shell pour manipuler les données.
- Pour des raisons pratiques, la présentation de MongoDB se fera avec Mongo Shell.

API

- Les fonctions de l'API
  - Requêter :
    - find(queryObject, filterObject)
      - Variante : findeOne
  - Enregistrer :
    - save(object) (efffectue une insertion ou une mise à jour en se basant sur l'identifiant de l'objet)
    - Insert(object)
    - Update(queryObject)
      - Par insertion ou par remplacement :
    - Update(queryObject, replacement)
    - findAndModify(queryObject, modificationObject, beforeAfter, all)
  - Supprimer :
    - remove(objectQuery)
- Les opérateurs présents dans la queryObject sont reconnaissables au $ qui les prefixe: $in, $all, $or, $exists, $mod, $not, $elemmatch, $where
- Opérateurs de l'objet modificationObject : $incr, $set, $unset, $push, $pushAll, $addToSet, $pull, $pullAll, $pop, $rename

ObjectId

- inséré automatiquement dans la propriété _id lors de la création d'un document (une opération de sauvegarde)
- immuable et unique
- basé sur le timestamp, le hostanme, le pid et un nombre aléatoire
- La méthode getTimestamp vous permet de récupérer la date de création du document
- La fonction de génération de cet objet peut être personnalisée et affectée à une propriété autre que _id.

Mongoose

- ODM : Object Relational Mapper

  => l'objectif est d'abstraire la base de données pour en faciliter l'usage : en n'ayant recours qu'à des éléments du langage. Les collections sont alors représentées par des fonctions constructeur (dit également models) et les documents par des objets javascript instanciés par ces models. L'ODM gère la synchronisation entre les objets ainsi créés et documents en base.
- => apporte tout un ensemble de mécanisme de contrôle sur les données, de validation absent dans mongoDB.

Le Schema

Un schema est un objet qui définit le mapping vers une Collection MongoDB. Le model est compilé à partir du schéma.

Le Schéma

- définit la structure des documents que peut contenir une collection
- des methodes d'instance :
  - Les documents possèdent leur propres methodes (get, set, toJSON, update, validate etc.). Les nouvelles methodes qu'on souhaite leur définir s'inscrivent au niveau du schéma à travers sa propriété **methods**
- des methodes static (de Model)
  - Ce type de methode ne s'appliquent au niveau du model (la collection) ex : count, findOne, find, findOneAndRemove, findOneAndUpdate, update etc.. L'ajout d'une methode static se fait au niveau du schema à travers sa propriété **statics**
- des propriétés virtuelles
  - Les propriétés statiques sont des propriétés qui sont calculées. Derrière une propriété virtuelle se cache un appel de fonction. Elles peuvent être très utiles au programme, mais ces propriétés, qui n'existent pas en tant que tel, ne sont pas sauvegardées en base.
- des middlewares (dont le système de validation fait partie :
  - Il existe 2 types de middlwares : ceux concernant les documents (et qui touchent aux fonctions init, validate, save, remove) et ceux concernant les requetes (count, findOne, find, findOneAndRemove, findOneAndUpdate, update ...)
  - ces 2 types de middlewares supportent des middleware pre et post

Le Model

Un model est l'équivalent d'un constructeur qui permet d'instancier des objets **représentants** des documents en base. Cette fonction particulière s'obtient à partir du schema une fois compilé.

- Le model contient toutes les methodes statiques (qui s'appliquent à la collection) permettant la création, la suppression, la récupération ou la mise à jour de documents au sein de la collection à laquelle elle se rapporte.

Les Documents

Une instance du Model. Appelé document par commodité, il désigne en fait l'objet javascript qui est lié à un document en base. On a souvent tendance à l'assimiler au document lui même.

API Mongoose

- L'API de mongoose est fidèle à celle de l'API de mongoDB. On y retrouve les même methodes et sous le même nom.
- Mongoose introduit un Query Builder qui offre, via son API fluide, la possibilité de construire une requête petit bout par petit bout, par appel de méthode successive.