Notes : Api avec Node JS

# Qu’est-ce que Node.js ?

Runtime qui permet de créer toutes sortes d’applications/outils côté serveur, en JavaScript.

## Installation de Node JS

Téléchargement puis installation de la dernière version de Node depuis NodeJS.org (permet également d’installer npm pour l’installation des packages pour les projets)

## Les « core modules »

C’est une liste des modules de base fournis par Node.js :

* http : lancer un serveur web, envoyer des requêtes
* https : lancer un serveur SSL
* fs : fournit des méthodes pour interagir avec le système de fichiers
* path : fournit des fonctions pour interagir avec les chemins de fichiers
* os : fournit des méthodes et des propriétés pour interagir avec le système d’exploitation

## Créer un serveur web

Dans cette partie, nous allons créer un simple serveur web avec NodeJS (code démo)

Voir le cours NODEJSAPI/190722/app.js

## Le cycle de vie d’un programme Node

Cycle de vie = boucle d’évènements :

* permet d’effectuer plusieurs tâches
* permet de rendre les tâches plus rapides
* permet à Node.js d’effectuer des opérations d’E/S non bloquantes

Cycle de vie d’un programme Node :

1. exécution d’un fichier de nœud à l’aide de node app.js
2. exécution du script
3. analyse du script par l’analyseur en langage machine (toutes les fonctions/variables vont être enregistrées dans un emplacement mémoire)
4. exécution du non infini par le programme
5. exécution de la boucle d’événements

## L’objet Request

Headers : métadonnées ajoutées à une requête http

L’objet request représente la requête http et possède des propriétés pour la chaîne de requête (paramètres, corps, en-têtes http, etc).

## L’objet Response

L’objet response représente la réponse http qu’une application Node JS envoie lorsqu’elle reçoit une requête http.

Méthode d’objet de response :

* res.end(); : méthode utilisée pour terminer le processus de réponse

## Router la http

Le routage peut être effectué en important le module d’URL intégré et en analysant l’URL de la demande. Pour vérifier si c’est correct, on peut comparer l’URL de la requête à l’URL de la route.

Voir le cours NODEJSAPI/190722/app.js

On peut définir des en-têtes sur la réponse en utilisant res.setHeader()

## Comment récupérer les données des requêtes entrantes ?

Exercice 1 : Stocker contenu message dans fichier texte

Voir le cours NODEJSAPI/190722/app.js

# Express JS

Le framework Express permet de simplifier les taches de demandes entrantes et permet de déployer des API plus rapidement.

## Alternantive d’Express JS

## Installation d’Express

Pour ajouter Express à notre projet, on exécute la commande **npm install express** à partir de notre dossier racine.

Ensuite, on créer un fichier **app.js** où l’on placera notre application Express :

Une image contenant texte

Description générée automatiquement

Démo

## Notion de middleware

Une application Express est une série de fonctions appelées **middleware**.

Chacun de ses éléments reçoit des objets *request* et *response* pour pouvoir les lire, les analyser et les manipuler.

Il peut également recevoir une méthode *next* qui permet à chaque middleware de passer l’exécution au middleware suivant.

## Express Js : app.listen()

La fonction **app.listen()** est utilisée pour lier et écouter les connexions sur l’hôte et le port spécifié (équivalent à la méthode http.Server.listen() de Node).

## Express JS : routage basique

Le routage est la façon dont une application répond à un nœud final spécifique : une URL (chemin et une méthode de requête http (GET, POST, etc).

Chaque route a une ou plusieurs fonctions de gestionnaire, qui sont exécutées lorsque la route est mise en correspondance.

Pour définir une route, on utilise la structure suivante :

app.methode(chemin, gestionnaire)

où app

app.METHOD(PATH, HANDLER)

Où :

* app est une instance d’express.
* METHOD est une [méthode de demande HTTP](http://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol).
* PATH est un chemin sur le serveur.
* HANDLER est la fonction exécutée lorsque la route est mise en correspondance.

Express Js : design Pattern MVC

Les APIs REST

Découpler Frontend et Backend

Toutes les applications (UI) ne nécessitent pas forcément un rendu de code HTML côté Serveur :

* Les applications Mobiles
* Les SPAs
* Les APIs de service (Google Maps API, Poké API)

Les Apis REST : format de données

Les Apis REST : Routing et endpoints

Les APIs REST : Méthodes http

Les APIs REST : Fondamentaux

Notre API doit être :

* Prédictible
* Bien documentée
* Bien structurée

Chaque requête doit être traitée sans « historique », pas de session.  
Le serveur doit traiter chaque requête indépendamment.  
Le serveur ne se soucie pas du client.  
On parle de découpage fort en Serveur et client.

Une API REST doit pouvoir informer de la durée de validité d’une réponse http, afin que le client puisse la mettre en cache.

Création d’une API REST basique

Logique utilisateur / endpoint (métier) / upload fichier / validation post / syntaxe moderne import / structure MVC

Les APIs REST : CORS

API Rest : Projet