



RÉSUMÉ THÉORIQUE

FIILIÈRE DÉVELOPPEMENT DIGITAL – OPTION WEB FULL STACK M214 – Créer une application cloud native


90 heures





Equipe de rédaction et de lecture

Equipe de rédaction :

Mme BOURDUS Imane: Formatrice en développement digital option Web Full Stack
Mme YOUSSEF Amel: Formatrice en développement digital option Web Full Stack





SOMMAIRE

1. INTRODUIRE LE CLOUD NATIVE

- Définir le cloud
- Définir l'approche cloud native

2. CRÉER DES APIS REST SIMPLES EN NODE JS ET EXPRESS JS

- Introduire Express et Node.js
- Créer des APIs REST
- Authentifier une API REST avec JWT

3. CRÉER UNE APPLICATION MICROSERVICE

- S'initier aux architectures microservices
- Créer une application microservices

4. MANIPULER LES CONTENEURS

- Apprendre la notion du conteneur
- Prendre en main Docker

DÉPLOYER UNE APPLICATION CLOUD NATIVE EN AZURE CLOUD

- Introduire Azure Cloud
- Déployer en Azure App service





Partie 1

Introduire le cloud native

Dans cette partie, vous allez :

- Définir le cloud
- Définir l'approche cloud native







CHAPITRE 1

Définir le cloud

Ce que vous allez apprendre dans ce chapitre :

- Concept du cloud et ses avantages ;
- Exemple des fournisseurs cloud ;
- Différence entre cloud privé, public et hybride ;
- Services du cloud (IaaS, PaaS, SaaS).







CHAPITRE 1


Définir le cloud

1. Concept du cloud et ses avantages ;
2. Exemple des fournisseurs cloud ;
3. Différence entre cloud privé, public et hybride ;
4. Services du cloud (IaaS, PaaS, SaaS).



1. Définir le cloud
Concept du cloud et ses avantages

- Le terme « cloud » désigne les serveurs accessibles sur Internet, ainsi que les logiciels et bases de données qui fonctionnent sur ces serveurs.
- Les serveurs situés dans le cloud sont hébergés au sein de **datacenters** répartis dans le monde entier.
- L'utilisation du cloud computing (informatique cloud) permet aux utilisateurs et aux entreprises de se libérer de la nécessité de gérer des serveurs physiques eux-mêmes ou d'exécuter des applications logicielles sur leurs propres équipements.



1. Définir le cloud
Concept du cloud et ses avantages

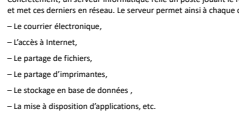
- Le cloud permet aux utilisateurs d'accéder aux mêmes fichiers et aux mêmes applications à partir de presque n'importe quel appareil, car les processus informatiques et le stockage ont lieu sur des serveurs dans un **datacenter** et non localement sur la machine utilisateur.
- C'est pourquoi vous pouvez vous connecter votre compte Instagram à partir de n'importe quel appareil, avec toutes vos photos, vidéos et l'historique de vos conversations. Il en va de même avec les fournisseurs de messagerie cloud comme Gmail ou Microsoft Office 365 et les fournisseurs de stockage cloud comme Dropbox ou Google Drive.
- Pour les entreprises, le passage au cloud computing supprime certains coûts et frais informatiques : par exemple, les sociétés n'ont plus besoin de mettre à jour et d'entretenir leurs propres serveurs, c'est le fournisseur de cloud qui s'en charge.



1. Définir le cloud
Concept du cloud et ses avantages

Serveur informatique vs cloud privé : quelle solution de stockage de données choisir pour une entreprise ?
La question du stockage des données se pose pour toute entreprise. Le volume des données numériques à gérer ne cesse d'augmenter. Optimiser la gestion des documents et le traitement des informations permet aux entreprises de rester concurrentielles.
Concrètement, un serveur informatique relie un poste jouant le rôle de serveur à différents postes utilisateurs (postes clients) et met ces derniers en réseau. Le serveur permet ainsi à chaque client de bénéficier de services divers :


- Le courrier électronique,
- L'accès à Internet,
- Le partage de fichiers,
- Le partage d'imprimantes,
- Le stockage en base de données ,
- La mise à disposition d'applications, etc.



1. Définir le cloud
Concept du cloud et ses avantages

Le client se connecte au réseau de l'entreprise et accède à ses documents. Le partage de documents entre les différents membres d'une équipe est également possible mais uniquement sur les postes installés en interne au sein de l'entreprise.
Les limites du serveur informatique:
⇒ **La sécurité des données en question**
L'utilisation d'un support de stockage expose les entreprises à d'autres risques :

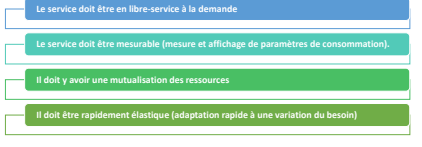
- pannes matérielles pouvant rendre les systèmes de gestion inséparables ;
- infiltration des données (introduction d'un malware dans les systèmes informatiques) ou piratage des données.
- Une capacité de stockage limitée
- Des coûts élevés pour l'entreprise



1. Définir le cloud
Concept du cloud et ses avantages

le cloud computing doit posséder 4 caractéristiques essentielles :

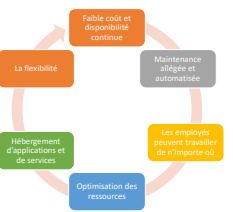
- Le service doit être en libre-service à la demande
- Le service doit être mesurable (mesure et affichage de paramètres de consommation).
- Il doit y avoir une mutualisation des ressources
- Il doit être rapidement élastique (adaptation rapide à une variation du besoin)




1. Définir le cloud
Concept du cloud et ses avantages

Les avantages du Cloud

- La flexibilité
- Hebergement d'applications et de services
- Optimisation des ressources
- Les employés peuvent travailler de n'importe où
- Maintenance allégée et automatisée
- Faible coût et disponibilité continue






CHAPITRE 1

Définir le cloud

1. Concept du cloud et ses avantages ;
2. Exemple des fournisseurs cloud ;
3. Différence entre cloud privé, public et hybride ;
4. Services du cloud (IaaS, PaaS, SaaS).

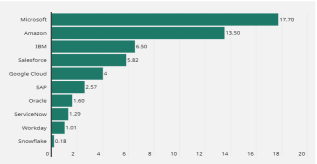


1. Définir le cloud
Exemple des fournisseurs cloud

- Les 10 premiers fournisseurs mondiaux de cloud en termes de revenus totaux pour le trimestre fiscal se terminant le 31 mars 2021 (en milliards de dollars américains)

Source: Statista, cloudwars.co

Fournisseur	Revenus (milliards de \$)
Microsoft	11.70
Amazon	10.50
IBM	8.40
Salesforce	7.40
Google Cloud	6.40
SAP	2.57
Oracle	1.00
ServiceNow	1.00
Workday	1.00
Zoom	1.00





CHAPITRE 1

Définir le cloud


1. Concept du cloud et ses avantages ;
2. Exemple des fournisseurs cloud ;
3. Différence entre cloud privé, public et hybride ;
4. Services du cloud (IaaS, PaaS, SaaS).




1. Définir le cloud
Différence entre cloud privé, public et hybride

Cloud public

- Les clouds **publics** sont généralement des environnements cloud créés à partir d'une infrastructure informatique qui n'appartient pas à l'utilisateur final.
- Alibaba Cloud, Microsoft Azure, Google Cloud, Amazon Web Services (AWS) et IBM Cloud sont les principaux fournisseurs de cloud public.
- Les clouds **publics** étaient habituellement exécutés hors site, mais les fournisseurs de cloud public proposent désormais des services cloud dans les datacenters de leurs clients, ce qui rend les notions d'emplacement et de propriété obsolètes.






1. Définir le cloud

Différence entre cloud privé, public et hybride

Cloud privé



Les clouds **privés** sont généralement définis comme des environnements cloud spécifiques à un utilisateur final ou à un groupe, et sont habituellement exécutés derrière le pare-feu de l'utilisateur ou du groupe.

Tous les clouds deviennent des clouds **privés** lorsque l'infrastructure informatique sous-jacente est spécifique à un **client unique**, avec un accès entièrement isolé.

1. Définir le cloud

Différence entre cloud privé, public et hybride

Cloud privé

Toutefois, les clouds **privés** ne reposent désormais plus forcément sur une infrastructure informatique sur site. Aujourd'hui, les entreprises créent des clouds privés dans des **datacenters hors site** et loués à des fournisseurs, ce qui rend les règles relatives à l'emplacement et à la propriété obsolètes.

Cette tendance a fait naître différents sous-types de clouds privés, notamment :

- Clouds **privés gérés** : Ce type de cloud est créé et utilisé par les clients, tandis qu'il est déployé, configuré et géré par un fournisseur tiers.
- Clouds **dédiés** : Il s'agit d'un cloud au sein d'un autre cloud. Vous pouvez déployer un cloud spécialisé dans un cloud public.

1. Définir le cloud

Différence entre cloud privé, public et hybride

Cloud hybride

Un cloud **hybride** fonctionne comme un environnement informatique unique créé à partir de plusieurs environnements connectés via des réseaux locaux (LAN), des réseaux étendus (WAN), des réseaux privés virtuels (VPN) et/ou des API.

Les caractéristiques des clouds hybrides sont complètes et les exigences associées peuvent varier selon l'utilisateur qui les définit. Par exemple, un cloud hybride peut inclure :

- Au moins un cloud privé et au moins un cloud public
- Au moins deux clouds privés
- Au moins deux clouds publics
- Un environnement virtuel connecté à au moins un cloud privé ou public

CHAPITRE 1

Définir le cloud

- Concept du cloud et ses avantages ;
- Exemple des fournisseurs cloud ;
- Différence entre cloud privé, public et hybride ;
- Services du cloud (IaaS, PaaS, SaaS).

1. Définir le cloud

Services du cloud (IaaS, PaaS, SAAS)

As-a-Service : définition

L'expression « **aaS** » ou « as-a-Service » signifie généralement qu'un tiers se charge de vous fournir un service de cloud computing, afin que vous puissiez vous concentrer sur des aspects plus importants, tels que votre code et les relations avec vos clients.

Chaque type de cloud computing allège la gestion de votre infrastructure sur site.

Il existe trois principaux types de cloud computing « **as-a-Service** », chacun offrant un certain degré de gestion :

- IaaS (Infrastructure as a Service)
- PaaS (Platform as a Service)
- SaaS (Software as a Service).

On-site	IaaS	PaaS	SaaS
Applications	Applications	Applications	Applications
Données	Données	Données	Données
Runtime	Runtime	Runtime	Runtime
Middleware	Middleware	Middleware	Middleware
Système d'exploitation	Système d'exploitation	Système d'exploitation	Système d'exploitation
Virtualisation	Virtualisation	Virtualisation	Virtualisation
Serveurs	Serveurs	Serveurs	Serveurs
Stockage	Stockage	Stockage	Stockage
Network	Network	Network	Network

■ Vous gérez
■ Gérer par le fournisseur de services

1. Définir le cloud

Services du cloud (IaaS, PaaS, SAAS)

IaaS : Infrastructure as a Service

Pour ce type de service le fournisseur de solution fournit les fonctions de virtualisation le système de stockage les réseaux et les serveurs et vous y donnez accès en fonction de vos besoins;

Ainsi, l'utilisateur ne contrôle pas l'infrastructure Cloud sous-jacente et il n'a pas à s'inquiéter des mises à jour physiques ou de la maintenance de ces composants.

Par contre et en tant qu'utilisateur, vous êtes responsable du **système d'exploitation** ainsi que des **données applications**, solutions de middleware et environnements d'exécution.

On-Premise

IaaS

Vous gérez
Gérer par le fournisseur de services

1. Définir le cloud

Services du cloud (IaaS, PaaS, SAAS)

IaaS : Infrastructure as a Service

L'IaaS est le modèle Cloud « as a Service » le plus flexible et libre, il apporte aux utilisateurs tous les avantages des ressources informatiques sur site, sans les actions et frais de gestion de l'infrastructure

En effet, il facilite la mise à l'échelle, la mise à niveau et permet d'ajouter des ressources, par exemple le stockage dans le Cloud

Exemples

Fournisseurs	AWS	Google Cloud	Azure
IaaS Services	Elastic ComputeCloud (EC2)	Compute Engine	Virtual Machine

1. Définir le cloud

Services du cloud (IaaS, PaaS, SAAS)

PaaS : Platform-as-a-Service

Le type de service **PaaS** est semblable à du IaaS, sauf que votre fournisseur de services Cloud fournit également le système d'exploitation et les environnements d'exécution.

Ainsi, l'utilisateur ne contrôle pas l'infrastructure Cloud sous-jacente et il n'a pas à s'inquiéter des mises à jour physiques ou de la maintenance de ces composants y compris le réseau, les serveurs, les systèmes d'exploitation ou de stockage.

Par contre et en tant qu'utilisateur, vous avez le contrôle pour le déploiement et configuration d'applications créés à l'aide de langages de programmation, de bibliothèques, de services et d'outils pris en charge par le fournisseur.

On-Premise

PaaS

Vous gérez
Gérer par le fournisseur de services

1. Définir le cloud

Services du cloud (IaaS, PaaS, SAAS)

PaaS : Platform-as-a-Service

Idéalement destiné aux développeurs et aux programmeurs, le PaaS fournit une plateforme simple et évolutive permettant aux utilisateurs d'exécuter et gérer leurs propres applications, sans avoir à créer ni entretenir l'infrastructure ou la plateforme généralement associée au processus.

Exemples

Fournisseurs	AWS	Google Cloud	Azure
PaaS services	AWS Elastic Beanstalk	Google App Engine	Azure App Service Azure Function App

Un service de gestion base de données géré par le fournisseur et accessible via le Cloud est considéré comme du PaaS. Exemple : Azure SQL DB, Azure Cosmos DB ...

1. Définir le cloud

Services du cloud (IaaS, PaaS, SAAS)

SaaS Software as a Service

Le SaaS (ou services d'applications Cloud, est le type le plus complet qui utilise le plus des services sur le marché du Cloud

Pour ce type de service le fournisseur fournit et gère une application complète accessible par les utilisateurs via un navigateur Web ou un client lourd

Ainsi, l'utilisateur ne contrôle pas la plateforme Cloud sous-jacente et il n'a pas à s'inquiéter des mises à jour logicielles ou l'application des correctifs et les autres tâches de maintenance logicielle

On-Premise

SaaS

Vous gérez
Gérer par le fournisseur de services

1. Définir le cloud

Services du cloud (IaaS, PaaS, SAAS)

SaaS Software as a Service

Le SaaS constitue une option intéressante pour les PME qui n'ont pas les ressources humaines pour gérer l'installation et le suivi de l'installation des mises à jour de sécurité et logicielles.

Par ailleurs, il est à noter que le modèle SaaS réduit le niveau de contrôle et peut nuire à la sécurité et aux performances -> il convient donc de choisir soigneusement votre fournisseur Cloud

Exemples

Fournisseurs	AWS	Google Cloud	Azure
SaaS services	Zoom	Google Apps	Microsoft Office 365

CHAPITRE 2

Définir l'approche cloud native

- Définition ;
- Avantages ;
- Vue générale sur les caractéristiques du cloud natif ;

2. Définir l'approche cloud native


Définition

Cloud Native : le Cloud Native décrit une approche de développement logiciel dans laquelle les applications sont dès le début conçues pour une utilisation sur le Cloud.

Il en résulte des applications Cloud Native (NCA) capables de pleinement exploiter les atouts de l'architecture du **Cloud Computing**.

Cette approche se concentre sur le **développement d'applications sous la forme de microservices individuels**, qui ne sont pas exécutés « On-Premises » (localement), mais sur des **plateformes agiles** basées sur des **conteneurs**.

Cette approche accélère le développement de logiciels et favorise la création d'applications **résilientes et évolutives**.




2. Définir l'approche cloud native

Fonctionnement

L'approche Cloud Native repose sur **quatre piliers** qui sont liés et interdépendants.

- Du côté **technique**, on trouve les **microservices** et les technologies de **conteneurs** développées spécialement pour l'environnement Cloud qui constituent des éléments fondamentaux du concept Cloud Native. Les différents microservices remplissent une fonction précise et peuvent être rassemblés dans un conteneur avec tout ce qui est nécessaire à leur exécution. Ces conteneurs sont portables et offrent aux équipes de développement un haut degré de flexibilité, par exemple lorsqu'il s'agit de tester de nouveaux services.
- Du côté de la **stratégie**, les **processus de développement** et la **Continuous Delivery** sont bien établis. Lors de la conception d'une architecture Cloud Native efficace, les équipes de développeurs (Developers = Dev), mais aussi l'entreprise (Operations = Ops) sont directement impliquées. Dans le cadre d'un échange constant, l'équipe de développeurs ajoute à un microservice certaines fonctionnalités livrées automatiquement par des processus de **Continuous-Delivery**.



CHAPITRE 2

Définir l'approche cloud native

- Définition ;
- Avantages ;
- Vue générale sur les caractéristiques du cloud natif ;

2. Définir l'approche cloud native
Avantages

Avantages

- ✓ **Flexibilité**: Comme tous les services sont exécutés indépendamment de leur environnement les développeurs disposent d'une **grande liberté**. Les **modifications apportées au code n'ont pas d'impact sur le logiciel dans son ensemble**. Le déploiement de nouvelles versions du logiciel présente donc un risque plus faible.
- ✓ **Évolutivité** des applications à proprement parler, qui permet aux entreprises de ne pas devoir procéder à une **mise à niveau coûteuse du matériel** en cas d'augmentation des exigences pour un service.
- ✓ Le haut **niveau d'automatisation** réduit par ailleurs à un minimum les erreurs humaines de configuration et d'utilisation.

2. Définir l'approche cloud native
Avantages

Avantages

- ✓ Voici quelques entreprises qui ont implémenté des techniques natives Cloud et qui ont obtenu, par conséquence, la vitesse, l'agilité et la scalabilité.
- ✓ Netflix, Uber et WeChat exposent des systèmes natifs Cloud qui se composent de nombreux services indépendants. Ce style architectural leur permet de répondre rapidement aux conditions du marché. Elles mettent instantanément à jour de petites zones d'une application complexe en service, sans redéploiement complet. Elles mettent à l'échelle individuellement les services en fonction des besoins.


Entreprise	Expérience
 NETFLIX	Dispose de plus de 600 services en production. Effectue des déploiements 100 fois par jour.
 Uber	Dispose de plus de 1 000 services en production. Effectue des déploiements plusieurs milliers de fois par semaine.
 WeChat	Dispose de plus de 3 000 services en production. Effectue des déploiements 1 000 fois par jour.

CHAPITRE 2
Définir l'approche cloud native

1. Définition ;
2. Avantages ;
3. **Vue générale sur les caractéristiques du cloud natif :**

2. Définir l'approche cloud native
Vue générale sur les caractéristiques du cloud natif

L'approche Cloud Native, se caractérise par l'utilisation d'architectures en **microservices**, de la technologie de **conteneurs**, de **livraisons en continu**, de **pipelines** de développement et d'infrastructure exprimés sous forme de code (Infrastructure As a Code), une pratique importante de la culture DevOps.



2. Définir l'approche cloud native
Vue générale sur les caractéristiques du cloud natif

Automatisation des processus du développement et de déploiement:

Comme l'approche DevOps, le **Cloud Native** cherche à rassembler les équipes Dev et Ops autour d'un objectif commun long terme : celui de la création de valeur business par les applications.

L'approche DevOps permet de converger vers une **approche Cloud Native** avec l'automatisation des processus et des technologies entre les équipes, de façon à intégrer plus rapidement les innovations dans les cycles de développement et de déploiement d'une **application Cloud Native**.

En parallèle du **Cloud Native**, l'adoption des méthodes Agiles va permettre d'intégrer les équipes métier dans cette collaboration avec les équipes techniques et de développement. L'idée est de collaborer pour délivrer une itération en améliorant le produit à chaque livraison de façon continue.

2. Définir l'approche cloud native
Vue générale sur les caractéristiques du cloud natif

Les microservices

Les **microservices** désignent à la fois une architecture et une approche de développement logiciel qui consiste à décomposer les applications en éléments les plus simples, indépendants les uns des autres. Contrairement à une approche monolithique classique, selon laquelle tous les composants forment une entité indissociable, les microservices fonctionnent en synergie pour accomplir les mêmes tâches, tout en étant séparés.

Pour communiquer entre eux, les **microservices** d'une application utilisent le modèle de communication requête-réponse. L'implémentation typique utilise des appels API REST basés sur le protocole HTTP. Les procédures internes (appels de fonctions) facilitent la communication entre les composants de l'application.

les microservices sont beaucoup plus faciles à créer, tester, déployer et mettre à jour


2. Définir l'approche cloud native
Vue générale sur les caractéristiques du cloud natif

Les Conteneurs

Tout comme le secteur du transport utilise des conteneurs pour isoler les différentes marchandises à transporter à bord des navires, des trains, des camions et des avions, le développement logiciel a de plus en plus recours au concept de **conteneurisation**.

Un package logiciel unique, appelé « **conteneur** », regroupe le code d'une application avec les fichiers de configuration, les bibliothèques et les dépendances requises pour que l'application puisse s'exécuter.

Ceci permet aux développeurs et aux professionnels de l'informatique de déployer les applications de façon **transparente dans tous les environnements**.



Partie 2
CRÉER DES APIS REST SIMPLES EN
NODE JS ET EXPRESS JS

Dans cette partie, vous allez :

- Introduire Express et Node js
- Créer des APIs REST
- Authentifier et autoriser une API REST avec JWT



CHAPITRE 1
Introduire Express et Node js

Ce que vous allez apprendre dans ce chapitre :

- Rappel du concept des APIs REST ;
- Rappel des méthodes du protocole http ;
- Définition de l'écosystème Node JS ;
- Configuration de l'environnement de développement;
- L'essentiel du Node js

5 heures

CHAPITRE 1
Introduire Express et Node js

1. **Rappel du concept des APIs REST ;**
2. Rappel des méthodes du protocole http ;
3. Définition de l'écosystème Node JS ;
4. Configuration de l'environnement de développement;
5. L'essentiel du Node js

Introduire Express et Node js
Rappel du concept des APIs REST

Une **API (interface de programmation d'application)** est un ensemble de définitions et de protocoles qui facilite la création et l'intégration de logiciels d'applications.

L'API est l'**intermédiaire** permettant à deux systèmes informatiques totalement indépendants d'interagir entre eux, de manière automatique, sans intervention humaine.

Elle est parfois considérée comme un **contrat** entre un **fournisseur** d'informations et un **utilisateur** d'informations, qui permet de définir le contenu demandé au consommateur (l'appel) et le contenu demandé au producteur (la réponse).


Par exemple, l'API conçue pour un service de météo peut demander à l'utilisateur de fournir un code postal et au producteur de renvoyer une réponse en deux parties : la première concernant la température maximale et la seconde la température minimale.



Introduire Express et Node js
Rappel du concept des APIs REST

Avantages des APIs

- ✓ Elle permet de pouvoir interagir avec un système sans se soucier de sa complexité et de son fonctionnement. ...
- ✓ Une API est souvent spécialisée dans un domaine et sur un use case particulier ce qui simplifie son utilisation, sa compréhension et sa sécurisation.
- ✓ Les APIs constituent un moyen simplifié de connecter votre propre infrastructure au travers du développement d'applications cloud-native.
- ✓ Elles vous permettent également de partager vos données avec vos clients et d'autres utilisateurs externes.
- ✓ Les APIs publiques offrent une valeur métier unique, car elles peuvent simplifier et développer vos relations avec vos partenaires, et éventuellement monétiser vos données (l'API Google Maps en est un parfait exemple)



Introduire Express et Node js
Rappel du concept des APIs REST


L'API REST ?

Roy Fielding a défini REST comme un style architectural et une méthodologie fréquemment utilisés dans le développement de services Internet, tels que les systèmes hypermédiés distribués.

La forme complète de l'API REST est l'interface de programmation de programmation d'applications de transfert d'état représentationnelle, plus communément appelée service Web API REST.

Par exemple, lorsqu'un développeur demande à l'API Twitter de récupérer l'objet d'un utilisateur (une ressource), l'API renvoie l'état de cet utilisateur, son nom, ses abonnés et les publications partagées sur Twitter. Cela est possible grâce aux projets d'intégration d'API.

Cette représentation d'état peut être au format JSON, XML ou HTML.



Introduire Express et Node js
Rappel du concept des APIs REST

Comment fonctionne une API REST ?

REST détermine la structure d'un API. Les développeurs s'obligent à un ensemble de règles spécifiques lors de la conception d'une API. Par exemple, une loi stipule qu'un lien vers une URL doit renvoyer certaines informations.

Chaque URL est connue sous le nom de demande (request), tandis que les données renvoyées sont appelées réponse (response).

L'API REST décompose une transaction pour générer une séquence de petits composants. Chaque composant aborde un aspect fondamental spécifique d'une transaction. Cette modularité en fait une approche de développement flexible.

Une API REST exploite les méthodes HTTP décrites par le Protocole RFC 2616



CHAPITRE 1
Introduire Express et Node js

1. Rappel du concept des APIs REST ;
2. **Rappel des méthodes du protocole http ;**
3. Définition de l'écosystème Node JS ;
4. Configuration de l'environnement de développement;
5. L'essentiel du Node js

Introduire Express et Node js
Rappel des méthodes du protocole HTTP

HTTP (Hypertext Transfer Protocol) est créé pour fournir la communication entre les clients et le serveur.

Il fonctionne en tant qu'une requête et une réponse.

Il existe deux méthodes HTTP principalement utilisées: GET et POST:


La méthode GET

La méthode GET de HTTP demande des données d'une source spécifiée. Les demandes GET peuvent être mises en cache et rester dans l'historique du navigateur. Il peut également être marqué.

Il ne doit jamais être utilisé lorsque vous travaillez sur des données sensibles. Les requêtes GET ont des restrictions de longueur et ne doivent être utilisées que pour obtenir des données.

La méthode POST

La méthode POST envoie les données à traiter à une source spécifiée. Contrairement à la méthode GET, les requêtes POST ne sont jamais paramétrées, elles ne restent pas dans l'historique du navigateur et nous ne pouvons pas les mettre en signet. De plus, les requêtes POST n'ont aucune restriction de longueur de données.



Comparaison des méthodes GET et POST

	GET	POST
Peut être marqué	Oui	Non
Peut être mise en cache	Oui	Non
Historique	Reste dans l'historique de navigateur.	Ni reste pas dans l'historique de navigateur.
Restrictions de type de données	Seulement les caractères ASCII sont autorisés.	N'a aucune restriction. Les données binaires sont également autorisées.
Sécurité	Il est moins sécurisé que le POST car les données envoyées font partie de l'URL.	Le POST est un peu plus sûr que GET car il ne reste pas dans l'historique du navigateur ni dans les journaux du serveur Web.
Visibilité	Les données sont visibles à tous dans l'URL.	N'affiche pas les données dans l'URL.

Exemple: Thak Abdineval - 2020



Outre les méthodes GET et POST, il existe d'autres méthodes.

Méthode	Descriptions
HEAD	Il en va de même avec la méthode GET mais elle ne renvoie que des lecteurs HTTP, pas de corps de document.
PUT	Il télécharge la représentation de l'URL spécifié.
DELETE	Il supprime la ressource spécifiée.
OPTIONS	Il retourne les méthodes HTTP supportées par le serveur.
CONNECT	Il convertit la connexion de demande en tunnel TCP / IP transparent.

Exemple: Thak Abdineval - 2020



CHAPITRE 1

Introduire Express et Node.js

1. Rappel du concept des APIs REST ;
2. Rappel des méthodes du protocole http ;
3. Définition de l'écosystème Node JS ;
4. Configuration de l'environnement de développement;
5. L'essentiel de Node.js



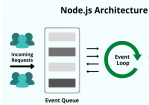
Exemple: Thak Abdineval - 2020

- ✓ **Node.js** est un environnement pour développer et déployer des applications web à base du Javascript.
- ✓ Plusieurs frameworks Javascript permettent de développer la partie frontale tel que Angular, VueJS , React
- ✓ **Node.js** est un environnement d'exécution single-thread, open-source et multi-plateforme permettant de créer des applications rapides et évolutives côté serveur et en réseau.
- ✓ Il fonctionne avec le moteur d'exécution Javascript V8 et utilise une architecture d'E / S non bloquante et pilotée par les événements, ce qui le rend efficace et adapté aux applications en temps réel.



Exemple: Thak Abdineval - 2020

- ✓ Node.js utilise l'architecture « Single Threaded Event Loop » pour gérer plusieurs clients en même temps.
- ✓ Pour comprendre en quoi cela est différent des autres runtimes, nous devons comprendre comment les clients concurrents multi-threads sont gérés dans des langages comme Java.
- ✓ Dans un modèle requête-réponse multi-thread, plusieurs clients envoient une requête, et le serveur traite chacune d'entre elles avant de renvoyer la réponse. Cependant, plusieurs threads sont utilisés pour traiter les appels simultanés. Ces threads sont définis dans un pool de threads, et chaque fois qu'une requête arrive, un thread individuel est affecté à son traitement.



Exemple: Thak Abdineval - 2020



Caractéristiques de Node.js

Node.js a connu une croissance rapide au cours des dernières années. Cela est dû à la vaste liste de fonctionnalités qu'il offre :

- ✓ **Facile** : Node.js est assez facile à prendre en main.
- ✓ **Évolutif** – Il offre une grande évolutivité aux applications. Node.js, étant single-thread, est capable de gérer un grand nombre de connexions simultanées avec un débit élevé.
- ✓ **Vitesse** – L'exécution non bloquante des threads rend Node.js encore plus rapide et plus efficace.
- ✓ **Paquets** – Un vaste ensemble de paquets Node.js open source est disponible et peut simplifier votre travail. Aujourd'hui, il y a plus d'un million de paquets dans l'écosystème NPM.
- ✓ **Backend solide** – Node.js est écrit en C et C++, ce qui le rend rapide et ajoute des fonctionnalités comme le support réseau.
- ✓ **Multi plateforme** – La prise en charge multi-plateforme vous permet de créer des sites web SaaS, des applications de bureau et même des applications mobiles, le tout en utilisant Node.js.
- ✓ **Maintenable** – Node.js est un choix facile pour les développeurs, car le frontend et le backend peuvent être gérés avec JavaScript comme un seul langage

Exemple: Thak Abdineval - 2020



Quelques entreprises des plus populaires qui utilisent Node.js aujourd'hui : Twitter, Spotify, eBay, LinkedIn

Applications de Node.js



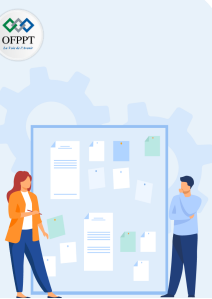
Exemple: Thak Abdineval - 2020



Express est un framework Node pour développer la partie Backend
Ainsi il est désormais possible de développer des applications fullstackJS
Dans ce module, on va développer des API Rest via Express. Dans un premier temps on va utiliser des données dans des fichiers JSON, ensuite on va traiter le cas MongoDB et MySQL.



Exemple: Thak Abdineval - 2020



CHAPITRE 1

Introduire Express et Node.js

1. Rappel du concept des APIs REST ;
2. Rappel des méthodes du protocole http ;
3. Définition de l'écosystème Node JS ;
4. Configuration de l'environnement de développement.
5. L'essentiel de Node.js

Installation de Node.js

Tout d'abord, nous devons télécharger le fichier Windows Installer (.msi) depuis le site officiel de Node.js.

Ce fichier d'installation MSI contient une collection de fichiers d'installation essentiels pour installer, mettre à jour ou modifier la version existante de Node.js.

Le programme d'installation contient également le gestionnaire de paquets Node.js (npm). Cela signifie que vous n'avez pas besoin d'installer npm séparément.

Visiter le site officiel : <https://nodejs.org/en/>

Node.js is an open-source, cross-platform JavaScript runtime environment.

Node.js assessment of OpenSSL 3.0.7 security advisory

Download for Windows (x64)

18.12.1 LTS (Long Term Support)

19.3.0 Current (Latest Release)

For information about supported releases, see the release schedule.

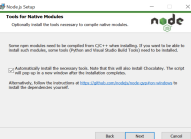
Exemple: Thak Abdineval - 2020



Installation de Node.js

Commencer le processus d'installation: Une fois que vous avez ouvert et exécuté le fichier.msi, le processus d'installation commence.

Node.js vous offre des options pour installer des outils pour les modules natifs. Si vous êtes intéressé par ces derniers, cliquez sur la case à cocher pour marquer vos préférences, ou cliquez sur **Suivant** pour continuer avec la valeur par défaut :

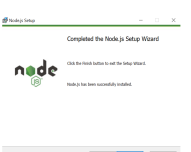


Exemple: Thak Abdineval - 2020



Installation de Node.js

Le système terminera l'installation en quelques secondes ou minutes et vous montrera un message de réussite. Cliquez sur le bouton Terminer pour fermer le programme d'installation de Node.js.



Exemple: Thak Abdineval - 2020

Installation de Node.js

Vérifier l'installation de Node.js

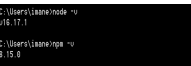
Pour vérifier l'installation et confirmer que la bonne version a été installée, ouvrez l'invite de commande votre PC et saisissez la commande suivante :

node -v

Avec Node, le gestionnaire de paquet NPM sera automatiquement installé

npm -v

npm permet de gérer vos dépendances et de lancer vos scripts



Exemple: Thak Abdineval - 2020



Généralement vous aurez besoin des commandes suivantes :

- ✓ **npm install** : installe le projet sur votre machine.
- ✓ **npm start** : démarre le projet.
- ✓ **npm test** : lance les tests du projet.
- ✓ **npm run dev** : s'occupe de lancer un environnement de développement agréable.

Exemple: Thak Abdineval - 2020



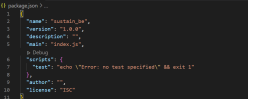
Etape 1 : Création du fichier package.json

1- Commencer par créer un nouveau répertoire qui sera le nom du projet

2-Taper la commande **npm init**

3-Créer un fichier index.js

A la fin de cette étape, nous aurons un fichier package.json contenant les paramètres du projet ainsi que les dépendances future.



Exemple: Thak Abdineval - 2020



Etape 2 : Installation du serveur Express

Pour installer le framework Express dans le projet, il faut exécuter la commande suivante :

npm install Express

Cette commande a pour but de télécharger depuis NPM remote repository l'ailibrairie Express ainsi que l'ensemble des librairies dont Express a besoin pour fonctionner dans votre répertoire de travail, dans le répertoire node_modules.

NPM va également l'ajouter dans votre package.json dans l'objet dépendances.



Exemple: Thak Abdineval - 2020

Introduire Express et Node Js

Configuration de l'environnement de développement

Remarque : Dans certains cours en ligne, on peut trouver l'option --save ou -s après la commande npm install Sachez qu'avant la version 5.0 de NPM, il fallait passer cette option pour retrouver la dépendance ajoutée dans le package.json.

Depuis la version 5.0 dès que vous passez la commande npm install la librairie est par défaut ajoutée au package.json Il n'est plus nécessaire de passer l'option -- ou -save

Pourquoi est ce qu'on a besoin d'ajouter la dépendance dans package.json?

Pour qu'un projet d'API Node JS ou tout autre projet Node puisse être repris par un autre développeur ou être déployé sur un serveur à distance, le package.json DOIT référencer toutes les librairies dont l'application a besoin pour bien fonctionner

Vous n'uploaderez pas toutes votre application avec le répertoire node_modules mais simplement votre code et le package.json

Le serveur sera en charge de faire un npm install pour récupérer toutes les dépendances

Introduire Express et Node Js

Configuration de l'environnement de développement

STEP 3 : Lancement du serveur Express

=> Dans cette étape, la librairie est installée dans notre projet, l'étape suivante consiste à appeler Express dans le fichier index.js

Pour cela on va utiliser l'éditeur Visual Studio Code

=> l'étape suivante consiste à lancer le serveur et le mettre à l'écoute sur un port donné

Ajoutons le bout de code suivant:

```
index.js
1 const express = require('express')
2 const app = express()
3
4 app.listen(3000, () => {
5   console.log('LISTENING API v1.0 ExpressJS')
6 })
```

Pour lancer le Serveur, taper sur Un terminal la Commande : node index.js

CHAPITRE 1

Introduire Express et Node js

- Rappel du concept des API REST ;
- Rappel des méthodes du protocole http ;
- Définition de l'écosystème Node JS ;
- Configuration de l'environnement de développement;
- L'essentiel du Node js

Introduire Express et Node Js

L'essentiel du Node js : Les modules : création, exports et import

En Node.js, un module est simplement un fichier Javascript contenant des fonctions, des objets ou des variables que vous souhaitez partager entre plusieurs fichiers. Les modules vous permettent de modulariser votre code, de le rendre plus facile à comprendre et de le réutiliser facilement.

Pour créer un module en Node.js, voici les étapes à suivre :

Etape 1 : Créer un fichier Javascript : Vous devez créer un fichier Javascript contenant le code que vous souhaitez exporter en tant que module. Par exemple, si vous souhaitez créer un module pour calculer la somme de deux nombres, vous pouvez créer un fichier nommé "sum.js"

Etape 2 : Définir les fonctions, les objets ou les variables que vous souhaitez exporter : Dans votre fichier Javascript, définissez les fonctions, les objets ou les variables que vous souhaitez exporter en utilisant la syntaxe module.exports. Par exemple, voici un exemple de module qui exporte une fonction de somme :

```
function sum(a, b) {
  return a + b;
}

module.exports = sum;
```

Introduire Express et Node Js

L'essentiel du Node js : Les modules : création, exports et import

Etape 3 : Importer le module : Pour utiliser le module dans un autre fichier Javascript, vous devez l'importer en utilisant la syntaxe require. Par exemple, pour utiliser le module "sum.js" que nous avons créé ci-dessus, vous pouvez l'importer dans un autre fichier Javascript en écrivant :

```
const sum = require('./sum');
console.log(sum(14, 3)); // résultat: 17
```

Dans cet exemple, nous avons importé le module "sum.js" à l'aide de l'instruction **require** en passant le chemin relatif vers le fichier "sum.js". Nous avons ensuite utilisé la fonction sum exportée par le module pour calculer la somme de deux nombres.

En résumé, pour créer un module en Node.js, vous devez définir les fonctions, les objets ou les variables que vous souhaitez exporter dans un fichier Javascript, utiliser la syntaxe module.exports pour exporter le code, puis importer le module dans d'autres fichiers Javascript à l'aide de l'instruction require.

Introduire Express et Node Js

L'essentiel du Node js : Les modules : création, exports et import

Depuis l'arrivée de la norme ES6 du langage, une nouvelle directive est apparue : il s'agit de la directive "import". Cette directive apporte de nouveaux avantages mais nécessite que le module que vous importiez la prenne en charge.

La directive "require" indique à Javascript d'importer la totalité du module demandée. Si le module en question est lourd, cela peut allonger le délai d'affichage d'une page. La directive "import" permet de n'importer qu'une partie spécifique d'un module. On gagne ainsi en légèreté et en rapidité de traitement. De plus, la directive import peut être utilisée de manière asynchrone. On peut continuer à exécuter du code ou à effectuer d'autres imports en parallèle de votre import initial. Ce n'est pas le cas de "require", qui fonctionne de manière synchrone et doit attendre la fin de l'import précédent avant de s'exécuter. Pour qu'un module puisse être utilisé avec la directive "import", celui-ci doit utiliser dans son code la directive "export".

Introduire Express et Node Js

L'essentiel du Node js : Les modules : création, exports et import

Dans NodeJS, il peut y avoir une confusion entre les deux directives. Par exemple, pour la plateforme Express JS, on trouve sur internet des morceaux de code avec les deux directives.

```
// Import avec require
const express = require('express');

// Import avec import
import express from 'express';
```

Pourtant, la plateforme Express ne gère pas la directive "import". Si vous essayez cette directive sans avoir installé d'autres modules, vous obtiendrez le message d'erreur "express has no default export". La raison pour laquelle la directive "import" va fonctionner dans certains codes est la présence d'une autre librairie, Babel.

Babel est un transpilateur qui peut convertir plusieurs types de code différents dans du Javascript. Il gère entre autres le langage TypeScript. Babel détecte si un module supporte la directive "import" et, si ce n'est pas le cas, convertit les instructions en directives "require". Si vous souhaitez ne plus vous poser la question sur la directive à utiliser dans votre code, installez Babel sur votre serveur Node.js.

Introduire Express et Node Js

L'essentiel du Node js : Les modules : création, exports et import

Les bibliothèques standard Node.js : Voici la liste des bibliothèques contenues dans Node.js considérées comme stables:

- ✓ REPL : c'est l'interpréteur que vous avez quand vous tapez node dans votre console.
- ✓ assert : pour faire des tests.
- ✓ console : pour les logs.
- ✓ debugger : point d'arrêt, step, ...
- ✓ dns : les noms de domaines.
- ✓ event : tout sur la gestion des événements.
- ✓ fs : tout sur le système de fichiers.
- ✓ global : tout ce qui est tout le temps disponible.
- ✓ http : un serveur, un client, requête, réponse, ...
- ✓ net : wrapper réseau asynchrone.
- ✓ path : gestion des chemins sur un système de fichier.

Introduire Express et Node Js

CHAPITRE 2

Créer des APIs REST

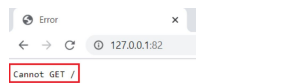
Ce que vous allez apprendre dans ce chapitre :

- API REST exposant des opérations CRUD sur un fichier Json
- Test de l'API REST avec Postman
- API REST manipulant une base de données MongoDB
- Opérations CRUD ;

Création des API Rest : API REST exposant des opérations CRUD sur un fichier Json

Etape 1 : Présentation de notre source de données

Etant donné que nous n'avons aucune route configurée, sur le navigateur, si on tape 127.0.0.1:3000 on aura le message d'erreur suivant:



Une fois que le serveur est lancé, on pourra développer nos API Rest, deux étapes sont nécessaires :

1-Les ressources (fichiers json, bases de données MongoDB ou mysql...)

2-Les routes : (les chemins pour récupérer , ajouter, modifier et supprimer les données disponibles dans nos ressources)

Création des API Rest : API REST exposant des opérations CRUD sur un fichier Json

Etape 1 : Présentation de notre source de données

Nous allons considérer un fichier equipes.json contenant un ensemble d'équipes de foot. L'objectif de cette section est d'exposer les opérations CRUD(Create, Remove, Update,Delete) sur cette source de données

Ainsi les routes que nous allons considérer sont les suivantes:

- ✓ GET /equipes(display)
- ✓ GET /equipes/{id} (display)
- ✓ POST /equipes(create)
- ✓ PUT /equipes/{id} (update)
- ✓ DELETE /equipes/{id} (remove)

CHAPITRE 2

Créer des APIs REST

Ce que vous allez apprendre dans ce chapitre :


- API REST exposant des opérations CRUD sur un fichier json
- Test de l'API REST avec Postman
- API REST manipulant une base de données MongoDB
- Opérations CRUD ;

Test de l'API REST avec Postman

Notre source de données est le fichier equipes.json contenant des équipes. Chaque équipe dispose des champs id, name, country

=>Placer ce fichier dans la racine de votre projet.

Afin de tester nos api rest, nous allons installer l'outil Postman



```
1 {
2   "id": 1,
3   "name": "Psgo",
4   "country": "France"
5 }
6
7 {
8   "id": 2,
9   "name": "Barcelone",
10  "country": "Espagne"
11 }
12
13 {
14   "id": 3,
15   "name": "Real Madrid",
16   "country": "Espagne"
17 }
18
19 {
20   "id": 4,
21   "name": "Milan",
22   "country": "Italie"
23 }
24
25 {
26   "id": 5,
27   "name": "Bayern",
28   "country": "France"
29 }
```

Test de l'API REST avec Postman

Postman, c'est quoi ?

Postman est officiellement présentée comme une plateforme API pour la création et l'utilisation d'API. D'une manière générale, Postman est une plateforme qui permet de simplifier chaque étape du cycle de vie des API et de rationaliser la collaboration, afin de créer, plus facilement et plus rapidement, de meilleures API.

Pourquoi utiliser Postman ?

La plupart des utilisateurs de Postman recourent à cette plateforme pour la construction et la formulation de requêtes, afin de tester des API sans avoir à réécrire de code. Parmi les nombreux points forts de Postman, on relève :

- ✓ la possibilité d'utiliser la plateforme, quel que soit le langage utilisé pour la programmation des API ;
- ✓ une interface utilisateur assez simple et facile à prendre en main ;
- ✓ l'absence de compétences nécessaires en codage.

Test de l'API REST avec Postman

Comment marche Postman ?

Le fonctionnement de Postman se résume le plus souvent à formuler une requête en suivant la structure spécifique (Verbe http + URI + Version http + Headers + Body) puis à obtenir une réponse.


Le code de réponse HTTP livré par la plateforme informe ensuite le développeur du statut de la réponse : "200 OK" pour une requête réussie, "404 Not Found" pour un échec, etc.

Comment télécharger Postman ?

Postman est compatible avec les différents systèmes d'exploitation (Linux, Windows et OS X). Pour télécharger Postman, il suffit de se rendre sur le site internet officiel de la plateforme.

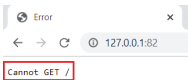
<https://www.postman.com/download/>

Download Postman



Test de l'API REST avec Postman

Étant donné que nous n'avons aucune route configuré, sur le navigateur, si on tape 127.0.0.1:30 on aura le message d'erreur suivant:



Une fois que le serveur est lancé, on pourra développer nos API Rest, deux étapes sont nécessaires :

- 1-Les ressources (fichiers json, bases de données MongoDB ou mysql...)
- 2-Les routes : (les chemins pour récupérer , ajouter, modifier et supprimer les données disponibles dans nos ressources)

Test de l'API REST avec Postman

Étape 2 : Route GET /equipes : Ajoutons le bout de code suivant:

```

1 const express = require('express')
2 const app = express()
3
4 const equipes = require('./equipes.json')
5
6 app.listen(30, () => {
7   console.log('REST API via ExpressJS')
8 })
9
10 app.get('/equipes', (req,res) => {
11   res.send('Liste des Equipes')
12 })

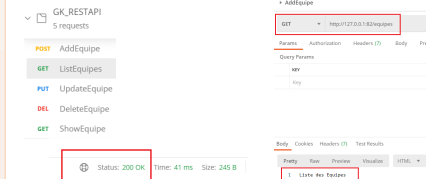
```

Le framework Express Nous propose des méthode get,post,put delete pour manipuler Les data
→ faites ctrl + c : pour Annuler le serveur puis relancer nodeindex.js

Test de l'API REST avec Postman

Étape 2 : Route GET / equipes

Avec POSTMAN, on va créer une collection pour nos 4 requêtes



Test de l'API REST avec Postman

Étape 2 : Route GET / equipes

Afin de récupérer les données, ajoutons la ligne de code suivante

```

app.get('/equipes', (req,res) => {
  //res.send('Liste des Equipes')
  res.status(200).json(equipes)
})

```

Nous avons remplacé la méthode send par la méthode json
En effet notre API REST va retourner un fichier JSON au client
Et non pas du texte ou un fichier html Nous avons également Ajouté le statut 200 qui correspond au code réponse http Indiquant au client que sa requête s'est terminée avec succès

Test de l'API REST avec Postman

Étape 2 : Route GET / equipes/id

La requête suivante est aussi une requêteGET avec un paramètre vide

```

app.get('/equipes/:id', (req,res) => {
  const id = parseInt(req.params.id)
  const equipe = equipes.find(equipe=> equipe.id === id)
  res.status(200).json(equipe)
})

```



Test de l'API REST avec Postman

Rappel des Middlewares :Les middlewares sont des fonctions qui s'exécutent lors de la requête au serveur. Ces fonctions ont accès aux paramètres de la requête et de la réponse et peuvent donc effectuer beaucoup de choses pour améliorer/automatiser les fonctionnalités de l'API

Le middleware se situe entre la requête et la réponse : user request -> middleware -> response

La requête suivante est POST permettant de poster des Data vers le serveur

Pour récupérer les données passées dans la requête POST, nous devons ajouter un middleware à notre Node JS API afin qu'elle soit capable d'interpréter le body de la requête Ce middleware va se placer entre l'arrivée de la requête et nos routes et exécuter son code, rendant possible l'accès au body

```

const express = require('express')
const app = express()

// Middleware
app.use(express.json())

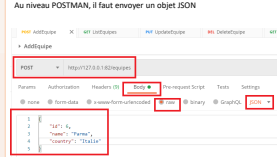
app.post('/equipes', (req,res) => {
  equipes.push(req.body)
  res.status(200).json(equipes)
})

```

Test de l'API REST avec Postman

Route POST /equipes/

Au niveau POSTMAN, il faut envoyer un objet JSON



Test de l'API REST avec Postman

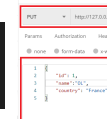
Route PUT /equipes/id

Pour la requête PUT on doit spécifier comme paramètre l'id de l'objet à modifier

```

app.put('/equipes/:id', (req,res) => {
  const id = parseInt(req.params.id)
  let equipe = equipes.find(equipe=> equipe.id === id)
  equipe.name = req.body.name,
  equipe.country = req.body.country,
  res.status(200).json(equipe)
})

```



Test de l'API REST avec Postman

Route PUT / equipes /id

On peut vérifier que la liste a été mis à jour



Test de l'API REST avec Postman

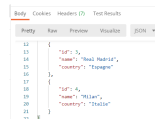
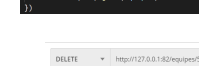
Route DELETE / equipes /id

Enfin, la requête Delete permettant de supprimer un objet de la liste

```

app.delete('/equipes/:id', (req,res) => {
  const id = parseInt(req.params.id)
  let equipe = equipes.find(equipe => equipe.id === id)
  equipes.splice(equipes.indexOf(equipe),1)
  res.status(200).json(equipes)
})

```



Test de l'API REST avec Postman

Exercice

Considérer la ressource joueurs.json

Chaque joueur dispose des champs(id, idEquipe,nom,numero,poste)

1-Développer les opérations crud pour l'entité joueur(4 requetes)

2-Développer la route permettant d'afficher les joueurs d'une équipe via son id(de l'équipe).

3-Développer la route permettant d'afficher l'équipe d'un joueur donné via son id.

4-Développer la route permettant de chercher un jour à partir de son nom