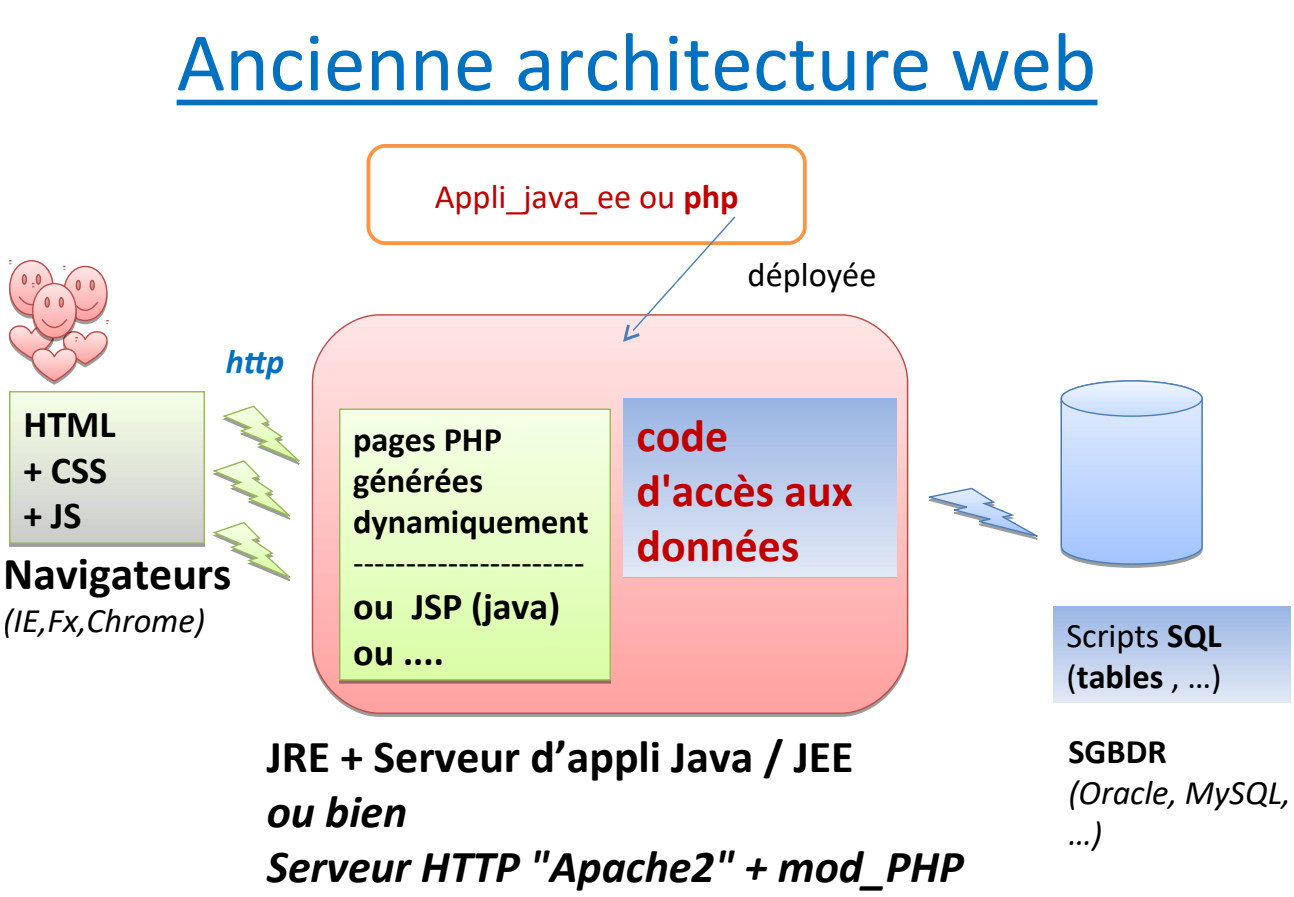
nodeJs , npm , express, ...

Table des matières

I - NodeJs : utilisations, principes

1. Principaux cas d'utilisation de nodeJs

1.1. Limitations de l'ancienne architecture logicielle "LAMP"



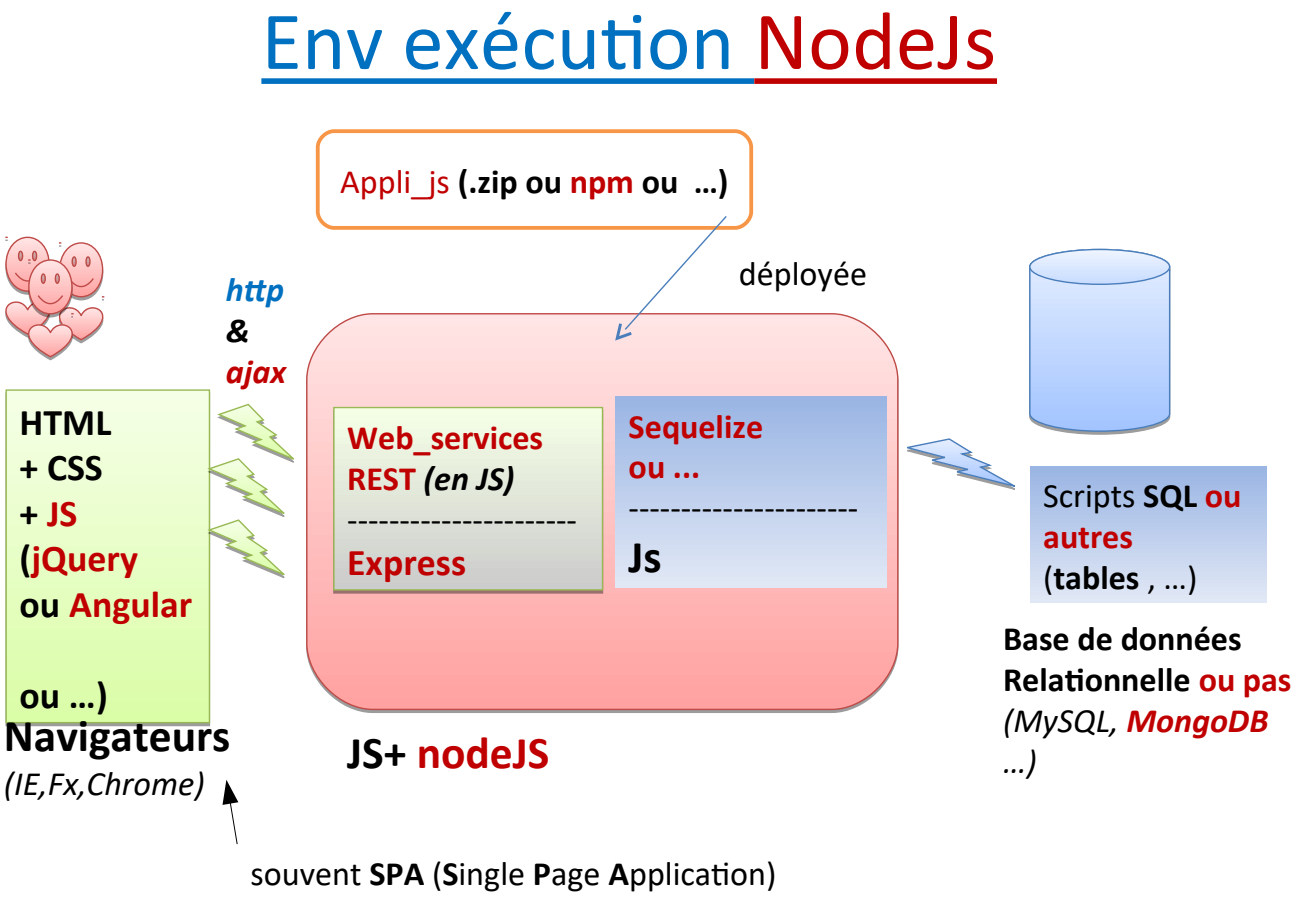
Dans ancienne architecture :

- presque tout coté serveur , le navigateur interprète et affiche des pages entièrement générées (de manière dynamique) coté serveur en fonction des valeurs en base de données
- le navigateur utilise "javascript" essentiellement pour dynamiser les pages (effets , ...)
- coté serveur : beaucoup de multi-threading (ex : Apache 2 ou JEE/Java)
- exemple classique : architecture **LAMP** = **L**inux / **A**pache / **M**ysql / **P**HP

Principaux défauts :

- problème de performance lorsque très nombreux utilisateurs simultanés
- IHM moyennement réactive
- dispersion des technologies "IHM/WEB/serveur" :PHP , JSP , ...(au cas par cas)

1.2. Plate-forme d'exécution des api REST nécessaires en arrière plan des applications "Single Page"



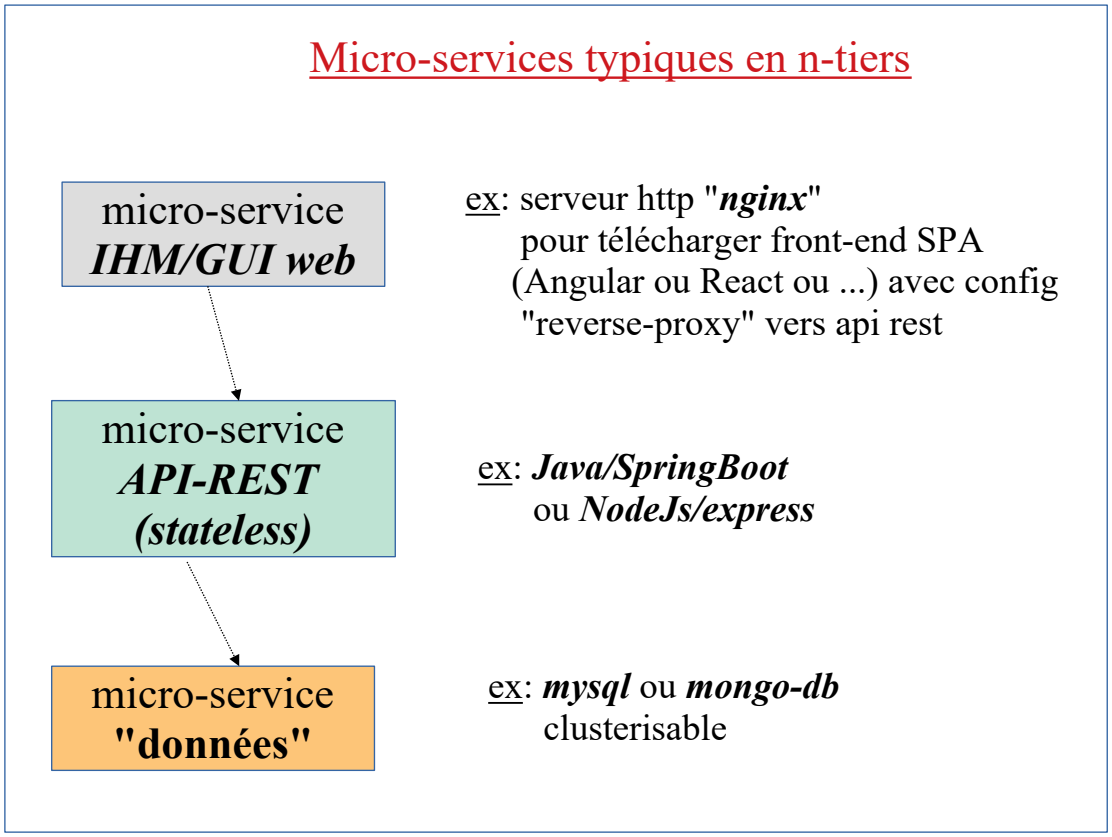
Dans nouvelle architecture :

- le coté serveur est moins sollicité : juste besoin gérer (retourner ou accepter) des données JSON à travers un paquet de WebServices "REST" , plus besoin de générer des pages HTML complètes, plus besoin de gérer des "sessions utilisateurs coté serveur" , plus besoin de gérer la navigation entre les pages
- le navigateur utilise beaucoup plus "javascript" (appels HTTP/ajax + Api DOM , ...)
- coté serveur : beaucoup moins de thread mais api asynchrones (ex : Node / express, ...)
- exemple classique : architecture **MEAN** = **M**ongo / **E**xpress / **A**ngular / **N**ode

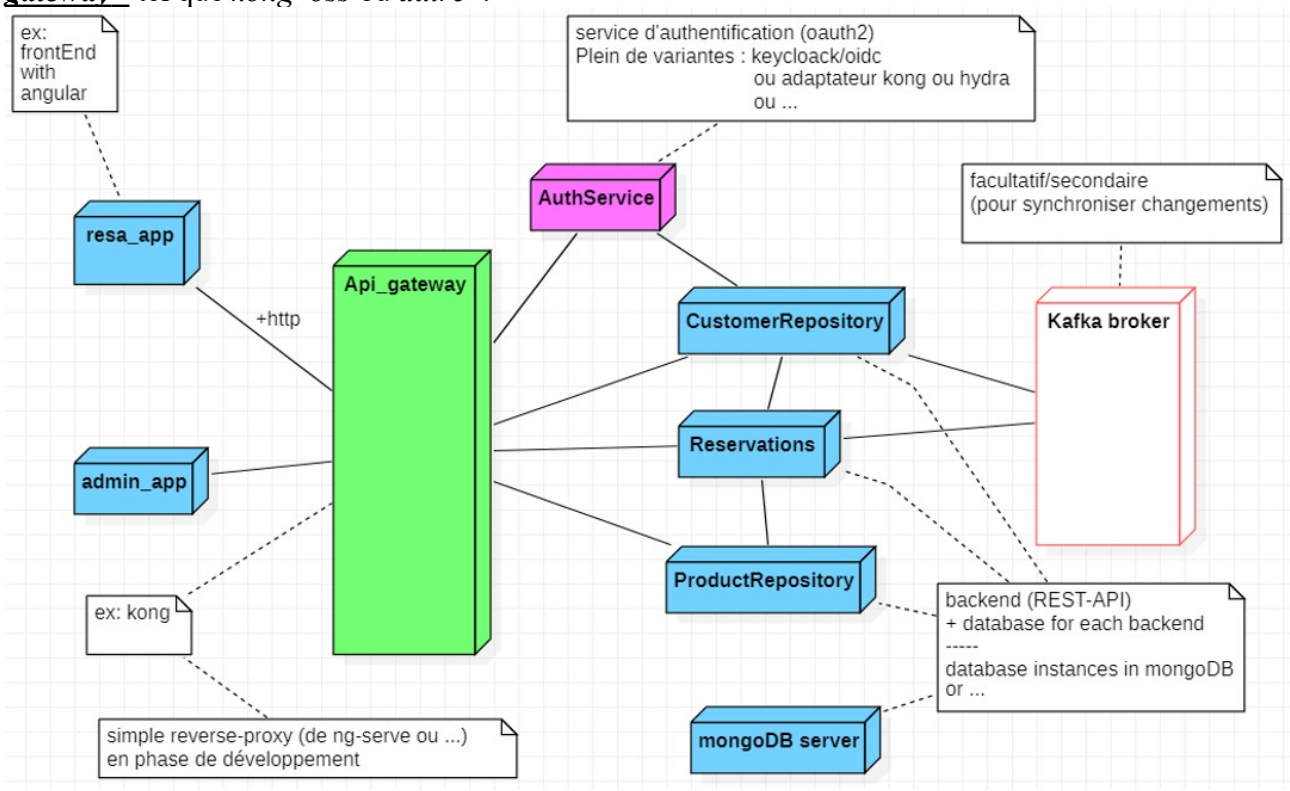
Principaux avantages:

- meilleurs performances lorsque très nombreux utilisateurs simultanés
- IHM plus réactive (meilleurs temps de réaction)
- meilleur séparation "frontEnd (HTML/CSS/JS)" et "backend (Api_REST + accès DB)

1.3. Bonne solution pour mettre en oeuvre un micro service



NB : il est fortement conseillé (en production) d'utliliser **nodeJs/express** en **arrière plan d'un "*api gateway*"** tel que *kong oss* ou *autre* .



2. Vue d'ensemble sur le fonctionnement de nodeJs

2.1. Présentation de nodeJs

node Js

node (nodeJs) est un environnement d'exécution javascript basé sur un moteur V8 (*"machine virtuelle" ou "moteur d'interprétation" javascript*).

Très souvent utilisé coté serveur , cet environnement d'exécution est basé sur une logique d'exécution asynchrone très légère et très efficace.

Pouvant charger du code modulaire compartimenté en modules téléchargeables (ex : "*express*") , nodeJs est essentiellement utilisé pour implémenter des *web services REST* (invoqués via HTTP).

2.2. Historique de nodeJs

Historique de nodeJs

NodeJs a été créé par **Ryan Dahl** en 2009. Son développement et sa maintenance sont effectués par l'entreprise Joyent .

Dès **2009** : NodeJs combine le moteur javascript V8 de Google, une boucle événementielle et une api I/O asynchrone .

En **2010** , un gestionnaire de modules téléchargeable "**npm**" est intégré à nodeJs (publish, install, ...) ce qui permet de partager le code efficacement.

De 2011 à 2015 : plein de fork, de join , ...
et parallèlement un très grand nombre de modules mis au point ou améliorés (express, ...)

Aujourd'hui : site officiel de nodeJs : **https://nodejs.org**

NB : NodeJs a petit à petit intégré les éléments du standard es2015+ .

- ***Les vielles versions de nodeJs et les anciennes applications*** *utilisaient principalement des* ***modules (non normalisés)*** *au format* ***"cjs / common-js"*** *(avec* ***module.exports. et require()*** *)* .

- **Les applications nodeJs modernes** peuvent maintenant s'appuyer à fond sur les **modules standards de es2015+** (avec **export** et **import … from ...**) .

2.3. Principes de fonctionnement de nodeJs

REPL (Read–Eval–Print Loop)

Un langage interprété (ex : "lisp" , "basic" ou "javascript") peut (à un haut niveau , quelque fois proche d'une interaction utilisateur) effectuer une boucle dite "REPL" de ce type :

loop({
- read expression *(ex : **read** "2*4+5" in variable **expr**)*
- eval result *(ex : res=**eval**(expr) = 13)*
- print result *(ex : console.log("res="+res) ;)*
})

vision fonctionnelle :

```
(loop (print (eval (read()))))
```

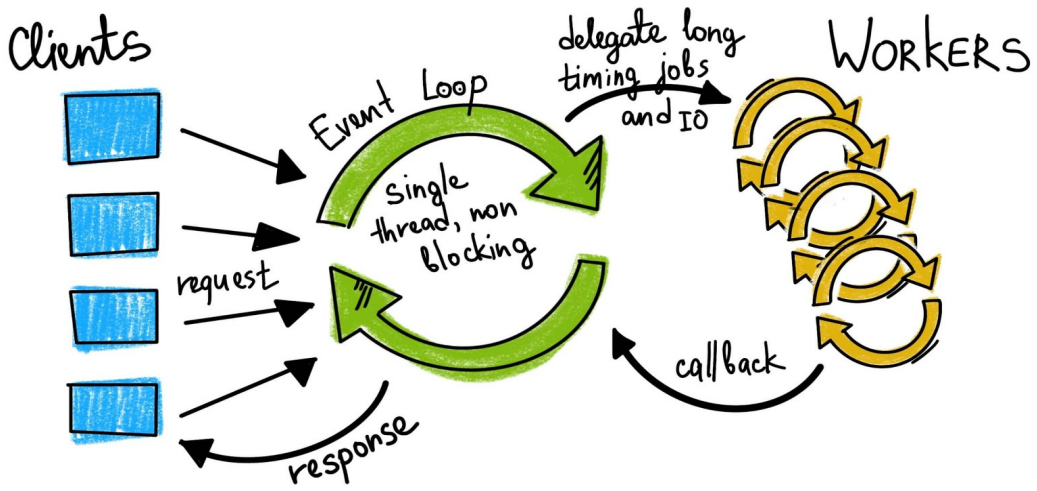
```
expression: 2+3
result=5
expression: 4+4
result=8
expression: 2*6+1
result=13
expression: fin
end
```

NB : l'instruction **eval**() du langage javascript permet de déclencher l'interprétation (et exécution) immédiate d'expression javascript récupérée sous forme de simple chaîne de caractères .
(exemple : eval("2*Math.PI*r"))

NodeJs s'est en partie inspiré de REPL mais a dû mettre en place une adaptation sophistiquée tenant compte des éléments suivants :

- pas de simple interactions "utilisateur" en mode texte
  mais interactions "web" (html/js/http/...) vues comme des événements utilisateurs à gérer.
- certaines tâches à déclencher son potentiellement longues et doivent être effectuées en mode asynchrone non bloquant

Single Threaded Event Loop Model

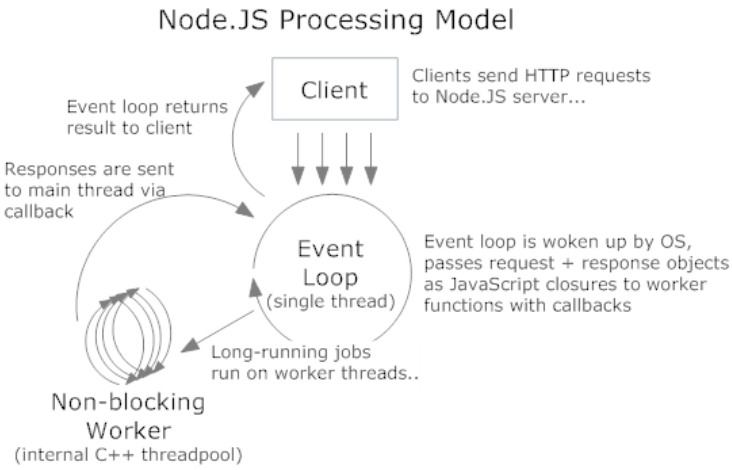


Les interactions "utilisateurs/clients" sont concrètement des appels http/ajax allant d'un navigateur web vers le serveur nodeJs.

NodeJs traite toutes les requêtes entrantes avec un seul grand thread principal qui :

- boucle sans arrêt sur :
  - réceptionner requêtes (vues comme des événements)
  - évaluer des réponses/résultats
  - renvoyer les réponses via http (vers navigateur ou application cliente)
- n'effectue que des actions non bloquantes (pour ne pas bloquer les autres utilisateurs/clients)

NB :
Certaines opérations longues sont déléguées à des "workers / thread secondaires" en tâche de fond qui vont à leur tour essayer d'optimiser les I/O en mode asynchrone non bloquant lorsque c'est possible (ex : accès non bloquant au système de fichiers , communications réseaux via des mécanismes "non blocking sockets" , ...)



Speudo Code of Main EventLoop of nodeJs :

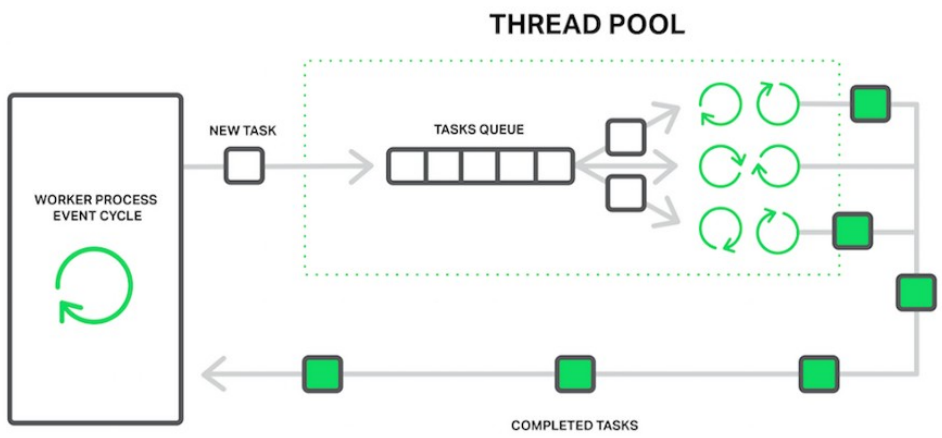
```
while(true){
    if(Event Queue receives a JavaScript Function Call){
        request = EventQueue.getClientRequest();
        if(request requires BlokingIO or
           takes more computation time)
            Assign request to main back Thread "T1"
        else
            Process and Prepare response immediatly
        }
    }
```

NB :

- Lorsque les traitements long effectués par les "workers / threads secondaires" en arrière plan sont terminés , les résultats sont récupérés du coté "single thread main javascript loop" via des fonctions "callbacks" dont les arguments sont les résultats (positifs ou négatifs)
- la majorité des résultats/réponses sont positifs (exemple : données recherchées , ...) et certaines réponses sont négatives (ex : ressources (fichiers/urls) inaccessibles , timeout , ...)

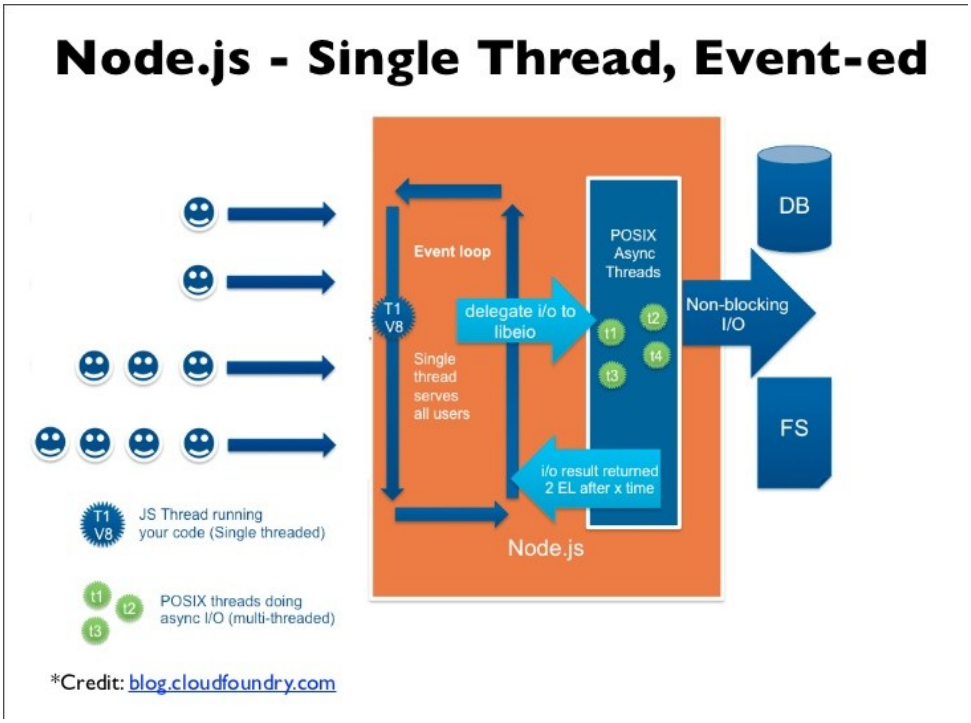
==> autrement dit , beaucoup de code est souvent nécessaire pour tenir compte de l'issue d'un traitement long déclenché en différé . Heureusement qu'il existe try/catch !

Exemple d'exécution asynchrone en c/c++ : nginx

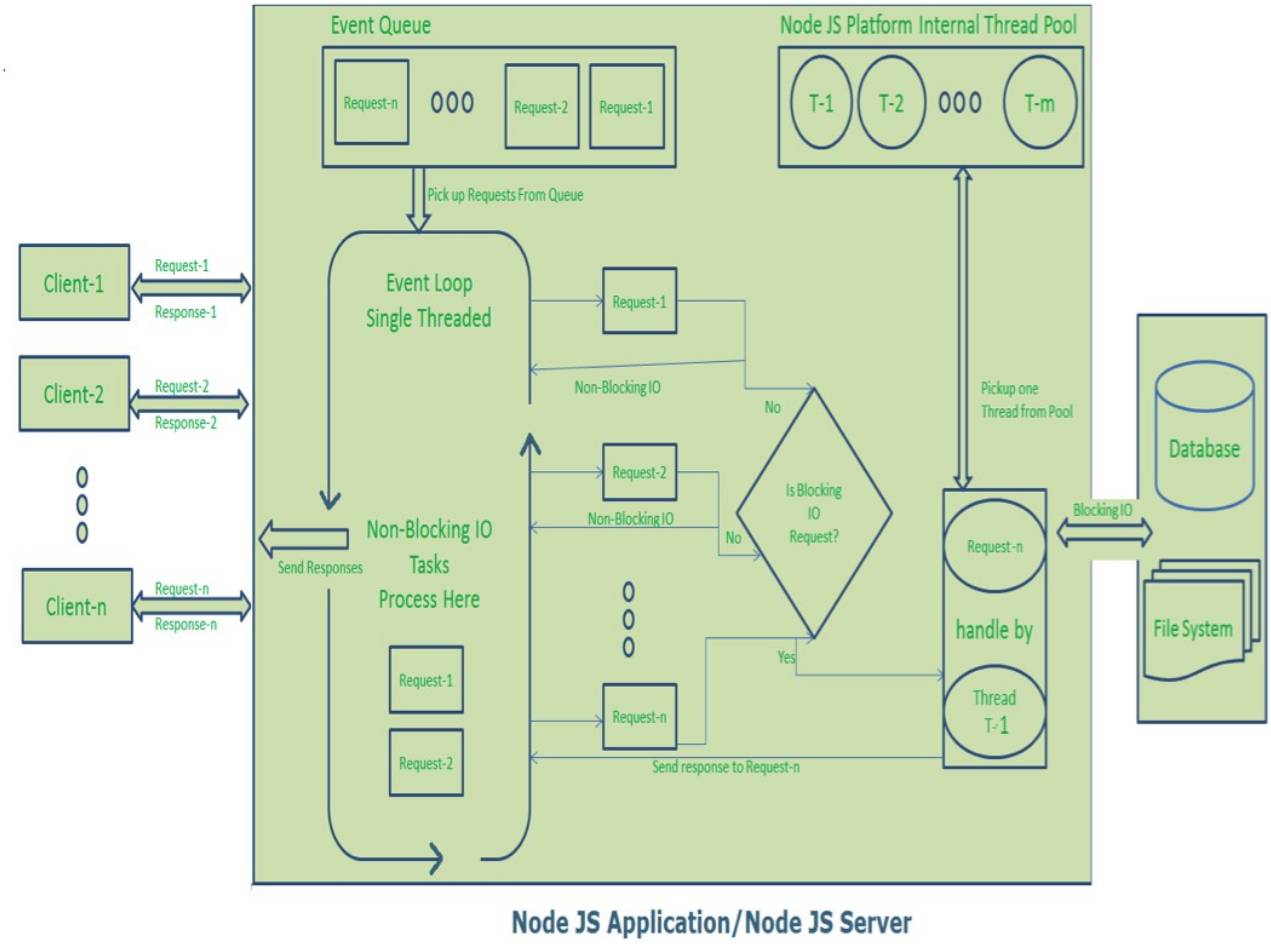


Dans nodeJs (en arrière plan) :
Threads asynchrones "POSIX" (avec un maximum de "non blocking I/O").

Architecture interne de nodeJs



Vue d'ensemble sur les mécanismes internes de nodeJs :



3. programmation asynchrone (nodeJs)

3.1. pas séquentiel , pas appels bloquants

Une application "nodeJs" se programme d'une manière très différente d'une application PHP ou Java/JEE :

- Pas d'appel bloquant mais toutes les opérations potentiellement longues sont déclenchées de manières asynchrones avec des fonctions "callbacks" à coder pour récupérer les résultats en différé.
  Principaux appels non bloquant et asynchrones : "requêtes SQL".

- Code très rarement séquentiel/linéaire mais plutôt événementiel (avec plein de points d'entrées) : dès qu'un événement survient , on déclenche un bloc de code pour le gérer.
  Les actions asynchrones alors déclenchées sont alors à l'origine d'autres événements et ainsi de suite ...

3.2. callbacks (avec ou sans Promise , async-await)

Les traitements liés au fonctions "callbacks" doivent quelquefois être synchronisés/ordonnés pour gérer une certaine logique métier (règle de gestion, algorithme, ...).
Sans astuce , ceci peut devenir une prise de tête souvent appelé "l'enfer des callbacks" .

Heureusement, les "Promise" (es2015) et les "async-await" permettent de :
- rendre le code plus fiable et plus lisibles
- simplifier la syntaxe
- retarder l'entrée à l'asile des développeurs

NB : la compréhension de "async/await" passe par une étude préalable des "Promise" .

II - Vue d'ensemble (node , npm , express, ...)

1. Ecosystème node+npm

node (nodeJs) est un **environnement d'exécution javascript** permettant essentiellement de :
- compartimenter le code à exécuter en **modules** (import/export)
- exécuter du code en mode "**appels asynchrones non bloquants** + callback" (sans avoir recours à une multitudes de threads)
- exécuter directement du code javascript sans avoir à utiliser un navigateur web

npm ***(node package manager)*** est une sous partie fondamentale de node qui permet de :
- **télécharger et gérer des packages utiles à une application** (librairies réutilisables)
- télécharger et utiliser des utilitaires pour la phase de développement (ex : grunt , jasmine , gulp , …)
- **prendre en compte les dépendances entre packages** (téléchargements indirects)
- générer éventuellement de nouveaux packages réutilisables (à déployer)
- ….

node est à peu près l'équivalent "javascript" d'une machine virtuelle java.
npm ressemble un peu à maven de java : téléchargement des librairies , construction d'applications.

Un **projet basé sur npm** se configure avec le fichier ***package.json*** et les packages téléchargés sont placés dans le sous répertoire **node_modules** .

Principales utilisations/applications de node :
- application "serveur" en javascript (répondant à des requêtes HTTP)
- application autonome (ex : StarUML2+ = éditeur de diagrammes UML , …)
- ….

2. Express

Express correspond à un des packages téléchargeables via npm et exécutables via node.
L**a technologie "express" permet de répondre à des requêtes HTTP** et ressemble un peu à un Servlet java ou à un script CGI .

A fond **basé sur des mécanismes souples et asynchrones (avec "routes" et "callbacks")** , **"express" permet de coder assez facilement/efficacement des applications capables de** :
- **générer dynamiquement des pages HTML** (ou autres)
- mettre en œuvre des **web services "REST"** (souvent au format "JSON") .
- prendre en charge les détails du protocoles **HTTP** (authentification "basic" et/ou "bearer" , autorisations "CORS" , ….)
- ….

"express" est souvent considéré comme une technologie de bas niveau lorsque l'on la compare à d'autres technologies "web / coté serveur" telles que ASP , JSP , PHP , …

"express" permet de construire et retourner très rapidement une réponse HTTP (avec tout un tas de paramétrages fin si nécessaire) . Pour tout ce qui touche au format de la réponse à générer , il faut

utiliser des technologies complémentaires (ex : templates de pages HTML avec remplacements de valeurs) .

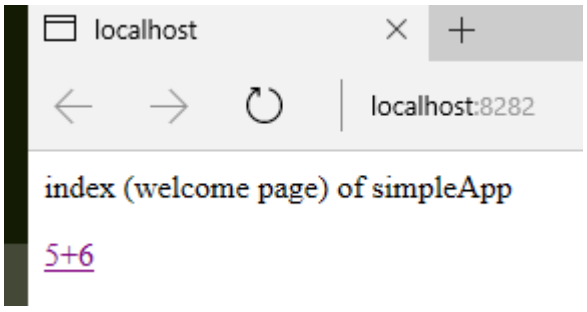
3. Exemple élémentaire "node+express"

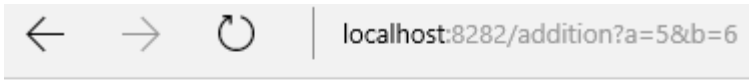
first_express_server.js

```
//modules to load:
var express = require('express');

var app = express();

app.get('/', function(req, res , next) {
     res.setHeader('Content-Type', 'text/html');
     res.write("<html> <body>");
     res.write('<p>index (welcome page) of simpleApp</p>');
     res.write('<a href="addition?a=5&b=6">5+6</a>');
     res.write("</body></html>");
     res.end();
});

//GET addition?a=5&b=6
app.get('/addition', function(req, res , next) {
     a = Number(req.query.a);   b = Number(req.query.b);
     resAdd = a+b;
     res.setHeader('Content-Type', 'text/html');
     res.write("<html> <body>");
     res.write('a=' + a + '<br/>');  res.write('b=' + b + '<br/>');
     res.write('a+b=' + resAdd + '<br/>');
     res.write("</body></html>");
     res.end();
});

app.listen(8282 , function () {
  console.log("simple express node server listening at 8282");
});
```

lancement : **node first_express_server.js**

via http://localhost:8282 au sein d'un navigateur web , on obtient le résultat suivant :





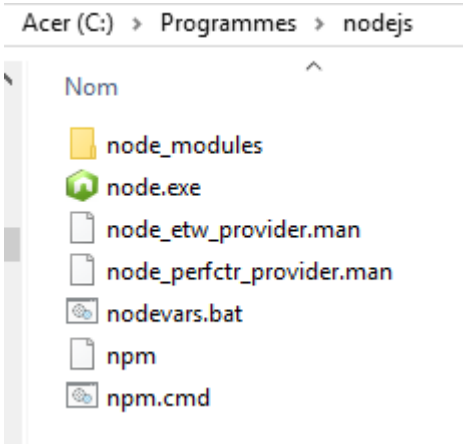
a=5
b=6
a+b=11

III - Node et npm : installation et utilisation

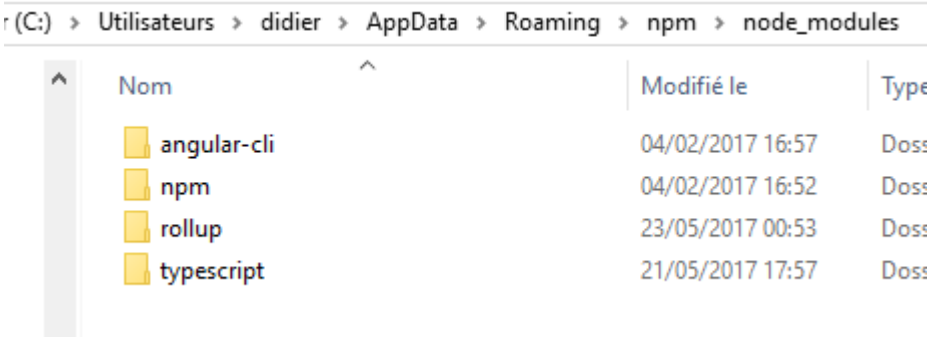
1. Installation de node et npm

Téléchargement de l'installeur ***node-v24.11.1-x64.msi*** (ou autre) depuis le site officiel de nodeJs
(https://nodejs.org)
Lancer l'installation et se laisser guider par les menus.
Cette opération permet sous windows d'installer node et npm en même temps .

Sur une machine windows 64bits , nodejs s'installe par défaut dans ***C:\Program Files\nodejs***



Et le répertoire pour les installations de packages en mode "global" (-g) est par défaut
C:\Users\username\AppData\Roaming\npm\node_modules



Vérification de l'installation (dans un shell "CMD") :

node --version
v24.11.1 (ou autre)

npm --version
11.6.2 (ou autre)

2. Configuration et utilisation de npm

2.1. Initialisation d'un nouveau projet

Un développeur utilise généralement npm dans le cadre d'un projet spécifique (ex : xyz).
Après avoir créer un répertoire pour ce projet (ex : C:\tmp\temp_nodejs\xyz) et s'être placé dessus, on peut lancer la c*ommande interactive* **npm init** de façon à **générer un début de fichier "*package.json*"**

Exemple de fichier ***package.json*** généré :

```
{
 "name": "xyz",
 "version": "1.0.0",
 "description": "projet xyz",
 "main": "index.js",
 "scripts": {
   "test": "echo \"Error: no test specified\" && exit 1"
 },
 "author": "didier",
 "license": "ISC"
}
```

2.2. installation de nouveau package en ligne de commande :

npm install --save express
npm install -s mongoose

permet de télécharger les packages "express" et "mongoose" (ainsi que tous les packages indirectement nécessaires par analyse de dépendances) dans le sous répertoire **nodes_modules** et de mettre à jour la liste des dépendances dans le fichier **package.json** :

```
…. ,
"dependencies": {
   "express": "5.1.0",
   "mongoose": "^8.17.1"
 },
….
```

Sans l'option **--save** (ou son alias **-s**) , les packages sont téléchargés mais le fichier package.json n'est pas modifié.

Par défaut , c'est la dernière version du package qui est téléchargé et utilisé.
Il est possible de choisir une **version spécifique** en la précisant après le caractère @ :

npm install -s mongoose@4.10
ou bien (autre exemple) :

npm install -s mongodb@2.0.55

Autre procédure possible :

1) **éditer** le fichier ***package.json*** en y ajoutant des dépendances (au sein de la partie "dependencies") :
   exemple :
   "dependencies": {
   "express": "^5.1.0",
   "markdown": "^0.5.0",
   "mongoose": "^8.17.1"
   }
2) lancer **npm install** (ou *npm update* ultérieurement) sans argument

Ceci permet de ***lancer le téléchargement et l'installation de tous les packages nécessaires listés dans le fichier package.json*** (ici "*mardown*" en plus) dans le sous répertoire **node_modules** .

Installation de packages utilitaires (pour le développement) :

Si l'on souhaite ensuite expliciter une dépendance de "développement" au sein d'un projet , on peut utiliser l'option **--save-dev** de **npm install** de façon ajouter celle ci dans la partie "devDependencies" de package.json :

npm install **--save-dev grunt**

```
…. ,
"devDependencies": {
   "grunt": "^1.6.1"
 }
```

2.3. Installation en mode global (-g)

L'option **-g** de **npm install** permet une installation en mode global : le package téléchargé sera installé dans ***C:\Users\username\AppData\Roaming\npm\node_modules*** sous windows 64bits (ou ailleurs sur d'autres systèmes) **et sera ainsi disponible (en mode partagé) par tous les projets** .

Le mode global est souvent utilisé pour installer des packages correspondant à des "utilitaires de développement" (ex : ***grunt*** ou ***typescript*** ou ***lite-server*** ou ***nodemon***) .

Exemple :
npm install -g nodemon
npm install -g lite-server

3. Utilisation basique de node

hello_world.js

```
console.log("hello world");
```

node ***hello_world.js***

4. quelques précisions sur les versions

~1.0.1 installe la version patch la plus récente pour la version mineur 1.0 (ça peut inclure 1.0.1 , 1.0.2 , 1.0.3 , … ~~mais pas 1.1.0~~)

^1.0.1 installe la version patch la plus récente pour la version majeure 1 (ça peut inclure 1.0.1 , 1.0.2 , 1.1.0 , 1.2.0 , 1.3.0 … ~~mais pas 2.0.0~~)

Et donc si deux développeur d'une même équipe télécharge un même fichier package.json depuis un référentiel de code source git mais qu'ils lancent la commande *npm install* à des moments différents il y a des chances pour que certains modules téléchargés le soit dans des versions mineures un peu différentes.

Pour éviter cet écueil , le fichier package-lock.json est généré (ou re-généré) par npm install en complément du fichier package.json .
Le fichier **package-lock.json** comporte une image précise des versions exactes de toutes les dépendances (directes ou indirectes) téléchargées à un moment précis .

Cas d'usage concret :

- Un développeur 1 , développe sur son ordinateur une application angular (avec un fichier package.json et package-lock.json précis) .
- Ce développeur met au point une première version de son application et publie cela au sein d'un référentiel git partagé/distant .
- Un développeur 2 créer un clone de ce projet sur son poste, puis au sein du répertoire du projet cloné, il lance la commande ***npm ci*** ou ***npm install .***

En lançant **npm install** , le fichier package-lock.json n'est pas pris en compte et les versions mineures peuvent être différentes .

En lançant **npm ci** , le fichier package-lock.json est pris en compte et les versions mineures seront les mêmes .

NB: **npm ci** signifie "**c**lean **i**nstall" avec npm ≥ 5.7.1 .

5. Scripts dans package.json

Exemple :

npm install -g tsc
npm install -g webpack
npm install -g mocha

package.json

```
{
  "name": "modules-webpack-ts",
  "version": "1.0.0",
  "description": "blabla",
  "main": "index.js",
  "scripts": {
    "build": "tsc && webpack",
    "test": "mocha",
    "webpack-watch": "webpack --watch"
  },
  "author": "didier",
  "license": "ISC",
  "devDependencies": {
    "webpack": "^4.44.2",
    "webpack-cli": "^3.3.12"
  }
}
```

npm run *nom_script_a_lancer*

---

npm run build

npm run webpack-watch

npm run test

---

npm test *est un raccourci pour* **npm run test**

…

6. Publication de modules via npm

6.1. Publication npm vers le référentiel public par défaut

NB : npm link (permettant de créer un lien symbolique local entre différents projets) comporte beaucoup de limitations/bugs .

NB :

- **Une réelle publication d'un module (ex: librairie réutilisable) vers le référentiel npm par défaut (https://www.npmjs.com/) nécessite de changer la version du package publié (ex 0.0.1 --> 0.0.2) .**

- **Tout nom de package publié doit être unique** (vaut donc mieux un nom de package avec un préfixe personnel ou d'entreprise).

1) se créer un compte sur le site officiel de npm
c'est gratuit et il faut respecter des règles de bonne conduite .

2)

```
cd myprefix-libxx
npm login
 ...saisir username et password ...
npm publish
```

3) visualiser les packages publiés
via une URL du genre **https://www.npmjs.com/package/myprefix-libxx**

4) **utiliser le package publié en tant que dépendance d'un autre projet**

```
cd pyy
npm install -s myprefix-libxx
...
```

6.2. Publication npm vers un référentiel privé

En phase de développement (surtout au début) , on a souvent besoin d'un gestionnaire de référentiel npm simple et léger de manière à tester des packages localement publiés sans devoir changer la version .

Gestionnaires de référentiel npm :

- **verdaccio** : simple/léger et agissant à la fois comme un référentiel privé et un comme un proxy vers le référentiel npm public .
- Partie npm de **nexus >=3** .
- ….

Utilisation de Verdaccio :

---
npm i -g verdaccio

C:\Users\usernameXy\AppData\Roaming**verdaccio\config.yaml** : *config file after first launch*
http://localhost:4873/ : *default verdaccio url*
---
verdaccio
Ctrl-C
---
npm set registry http://localhost:4873/
(replace default "npm set registry https://registry.npmjs.org")
---
npm adduser --registry http://localhost:4873
npm login *(ex: user1 or ...)*
password (ex: pwd1 or ...)
email (ex: user1@worldcompany.com or ....)
---
npm whoami
npm publish

NB: si derniere version publiée sur https://registry.npmjs.org
est 0.0.n alors en phase de dev (sur local-verdaccio) : 0.0.N+1

Pour si besoin rétablir l'utilisation du référentiel par défaut :
npm set registry https://registry.npmjs.org

//npm version patch //for incrementing version 0.0.n 0.0.n+1

IV - Modules - nodeJs

1. Modules dans environnement "nodeJs"

Un des grands atouts de nodejs est un programmation à base de **modules bien compartimentés**.

Grands principes :
- chaque module correspond à un fichier séparé
- **seuls les éléments exportés par un module pourront être vus par les autres**
- un module doit commencer par importer certains éléments d'un autre module avant de pouvoir les utiliser

Intérêts de la programmation à base de modules :
- moins d'effets de bords (moins de conflits de noms de variables , types , ...)
- écosystème à la fois simple/efficace et très extensible (sans forcer une complexité inutile : on ne charge que les modules nécessaires)
- chargement dynamique (souple et performant)
- écosystème ouvert (à des évolutions , de nouveaux modules concurrents, ...)
- ...

Il existe cependant plusieurs technologies de modules dans le monde javascript :
- L'écosystème nodeJs utilise depuis longtemps en interne des modules au format [**CJS**] "**cjs/commonjs**" avec une syntaxe "***module.exports... = ...***" et "***var mxy = require('xy')***"

- Depuis la version normalisée "es6/es2015" de javascript/ecmaScript , certains modules peuvent être codés en s'appuyant sur la syntaxe des **modules es2015** [**ESM**] (***export*** ... , ***import … from '..'***)

Epoques	***Possibilités pour les modules***	***Tendances***
Avant fin **2015** , 2016 et node Js 4,5,**6**,…,8	**cjs uniquement**	Habitudes prises avec premières versions *(module.exports… , require)*
Aux alentours de **2018** et nodeJs 9,**10**,…,12	cjs (fiable / mature) et es2015 (.msj ou bien .ts ⇒ .js)	Souvent cjs avec .js et es2015 avec typescript (.ts)
A partir de **2021** environ et nodeJs **14,..,16,..18,**...	**cjs (*legacy* , *par habitude*)** **et** **ESM (.ts ou directement .js via option *"type" : "module"* de *package.json*) ou via ".mjs"**	**cjs et .js dans anciens projets en maintenance évolutive.** **ESM dans projets modernes (js ou bien ts → js)**

Aujourd'hui et à partir de 2022 :
- on peut considérer l'ancienne technologie de module historique "**cjs**" comme devenant petit à petit "has been" (legacy) mais cependant encore très utilisée car mature/fiable et massivement utilisée par beaucoup d'anciens projets.
- Les modules **ESM** sont maintenant bien supportés et on l'immense avantage d'être standardisé (syntaxe normalisé utilisable partout : coté navigateur et coté nodeJs , en javascript et en typescript) . Ça peut être un choix à faire dès à présent sur de nouveaux

projets (si cette liberté est autorisée) pour se projeter dans l'avenir.

NB: selon le type exact de projet (angular , node/express/mongoose , nestjs , …) il y a souvent certains éléments imposés (ex : javascript ou typescript , "CJS" ou "ESM") et il faut évidemment en ternir compte .

2. Anciens modules "cjs/commonjs" (require)

Attention: petit à petit **obsolète/has-been/legacy** .

2.1. Module avec élément(s) exporté(s)

mycomputer_module.js

```
var myAddStringFct = function(a,b) {
  result=a+b;
  resultString = "" + a + " + " + b +" = " + result;
  return resultString;
};

module.exports.myAddStringFct = myAddStringFct;
```

NB: ***Seuls les éléments exportés seront vus par les autres modules !!!***

2.2. Importation de module(s) via require()

basic_exemple_with_modules.js

```
//chargement / importation des modules :
var mycomputer_module = require('./mycomputer_module'); // ./ for searching in local relative
var markdown = require('markdown').markdown; // without "./" in node_modules sub directory

//utilisation des modules importés :
var x=5;
var y=6;
var resString = mycomputer_module.myAddStringFct(x,y);

console.log(resString);
var resHtmlString = markdown.toHTML("**"+resString+"**");
//NB: "markdown" est un mini langage de balisage
// où un encadrement par ** génère un équivalent de
// <strong> HTML (proche de <bold>)
console.log(resHtmlString);
```

node basic_exemple_with_modules

résultats:

```
5 + 6 = 11
<p><strong>5 + 6 = 11</strong></p>
```

3. Modules "es2015" / "typescript" / env. "nodeJs"

Normalisés en 2015 (es2015/es6) , utilisable coté nodeJs dès 2017/2018 en typescript à partir de fin 2020 en javascript , les ***modules ES2015+*** correspondent à la "**technologie de modules" officielle et standard du langage javascript** .

Durant une période de transition (de 2017 à 2020 environ), les modules "es2015" étaient principalement utilisés dans des projets codés en langage typescript :

code_source.ts ----> transformation/transpilation **tsc** -------> code javascript à exécuter via node
import {...} from '...' **var ... = require('...')**

tsconfig.json

```
{
 "compilerOptions": {
   "target": "es2017",          /* si nodeJs récent , sinon target "es2015" ou "es5" */
   "module": "commonjs",
   "lib": ["es2015"],   /* ou autre  */
...
}
```

A partir de la version 14 de nodeJs (soit fin 2020 , 2021 environ) , en ajoutant "type": "module" dans package.json, il est enfin possible d'utiliser directement la syntaxe des modules "ESM" dans des fichiers ".js" et l'utilisation de typescript coté nodeJs devient donc facultative pour utiliser la syntaxe "import … from …." .

Si l'on ajoute pas "type": "module" dans package.json, il faut alors renommer le fichier "xy.js" en "xy.mjs" pour que la syntaxe *import … from ….* soit bien interprétée/comprise.

3.1. Exportation multiple en javascript (ou typescript)

math-util.js

```
export function additionner(x :number  , y :number) :number{
  return x + y;
}
export function multiplier(x :number  , y :number) :number{
  return x * y;
}
```

ou bien

```
function additionner(x :number  , y :number) :number{
    return x + y;
  }
function mult(x :number  , y :number) :number{
    return x * y;
  }
export { additionner, mult as multiplier };
```

On peut également exporter de la même façon des classes , des variables , ...

3.2. Importation multiple en javascript (ou typescript)

main.js

```
import { additionner as add, multiplier } from "./math-util.js"

function carre(x){
  return multiplier(x,x) ;
}
var msg1 = "Le carre de 5 est " + carre(5);   console.log(msg1);
var msg2 = "4 * 3 vaut " + multiplier(4, 3);  console.log(msg2);
var msg3 = "5 + 6 vaut " + add(5, 6); console.log(msg3);
```

Variantes d'importation avec éventuels préfixes :

main.js

```
import { f1 , f2 , f3 , f4  } from "./modxy.js";
let f_msg=f1('abc')+'-'+f2('abc')+'-'+f3('abc')+'-'+f4('abc');
```

ou bien (quand c'est possible) :

```
import * as f from "./modxy.js";
let f_msg=f.f1('abc')+'-'+f.f2('abc')+'-'+f.f3('abc')+'-'+f.f4('abc');
```

3.3. default export (one per module)

xy.js

```
export function mult(x, y) {
    return x * y;
  }

  //export default function_or_object_or_class (ONE PER MODULE)
  export default {
          name : "xy",
          features : { x : 1 , y: 3 }
  }
```

main.js

```
import xy , { mult }  from "./xy.js";
...
let msg = xy.name + "--" + JSON.stringify(xy.features) + "--" + mult(3,4);
```

3.4. exportation et importation globale (style nodeJs)

xy.js

```
function mult(x, y) {
        return x * y;
      }

const config {
          name : "xy",
          features : { x : 1 , y: 3 }
  }

export default { mult , config } ;
//équivalent à
//export default { mult: mult , config: config } ;
```

main.js

```
import xy  from "./xy.js";
...
let msg = xy.config.name + "--" + JSON.stringify(xy.config.features) + "--" + xy.mult(3,4);
```

Ce style d'exportation et d'importation "ESM" est le plus proche par rapport aux anciens modules common-js et est par conséquent le style conseillé pour une transition douce.

3.5. importation de modules chargés dans node_modules via npm

Exemples à adapter au contexte (javascript ou typescript):

```
import * as http from 'http';
//import express , { Request, Response } from 'express';
import { dirname } from 'path';
import { fileURLToPath } from 'url';
import express from 'express';
...
```

NB :

- **from 'http'** (et pas ~~from './http'~~)
- **from 'express'** (sans "./") si e**xpress téléchargé via npm** et *pas codé par nous même*
- **from './xyz-api-routes.js'** si './session-api-routes.js' est un **chemin relatif sur un fichier de notre projet**

3.6. package.json avec "type" : "module"

```
{
  ...
  "type" : "module",
  ...
  "dependencies": {
    "express": "^4.17.1",
    "mongoose": "^6.0.12"
  }
}
```

NB : **"type" : "module"** est indispensable pour que **node** prenne en charge les fichiers **".js"** (et non seulement les fichiers ".mjs") comme des **modules "es2015" .**

4. Modules prédéfinis/intégrés/noyaux de node

NodeJs comporte quelques **modules fondamentaux déjà intégrés dans le coeur de nodeJs**.
On peut directement utiliser ces modules (via require() ou import … from ou quelquefois directement avec une importation explicite) sans avoir à les récupérer préalablement par npm .

Modules prédéfinis/intégrés	Fonctionnalités
console	Affichage console : console.log("message qui va bien")
process , child_process	Gestion du processus en cours (accès aux variables d'environnement , arrêt du process, ...)
os	Informations sur système d'exploitation hôte
fs	Gestion du "File system"
path	Gérer les chemins vers des fichiers
util	Quelques fonctions utilitaires
http , https	Pour que l'appli nodeJs se comporte en serveur http ou https
events	Gestion des événements (base de l'asynchronisme)
buffer, stream	Gestion des buffers et des flux/streams
url , querystring	Analyse d'url , ...
net, dns	Gestion du réseau , du dns, ...
zlib	compression/décompression "zip" , ...
crypto	OpenSSL
...	
...	

4.1. Mini module intégré "console" :

Importation implicite / utilisation directe .

```
console.log("hello world") ;
console.warn("warning qui va bien") ;
console.error(new Error('error message'));  → avec new Error("...") pour "stack trace" .
```

```
hello world
warning qui va bien
Error: error message
    at Object.<anonymous> (C:\tp\local-git-mycontrib-repositories\tp_node_js\essais\builtin.js:4:15)
    at Module._compile (node:internal/modules/cjs/loader:1101:14)
    at Object.Module._extensions..js (node:internal/modules/cjs/loader:1153:10)
    at Module.load (node:internal/modules/cjs/loader:981:32)
    at Function.Module._load (node:internal/modules/cjs/loader:822:12)
    at Function.executeUserEntryPoint [as runMain] (node:internal/modules/run_main:81:12)
    at node:internal/main/run_main_module:17:47
```

Pleins d'autres possibilités avancées ; créer un objet console à partir de "streams" , …
https://nodejs.org/api/console.html pour approfondir le sujet si besoin .

4.2. module intégré "process" :

Importation implicite / utilisation directe .

```
console.log("PATH=" + process.env.PATH) ; //AFFICHE variable d'environnement PATH
process.exit(0);
console.log("pas de suite et message pas affiché si process arreté") ;

console.log("process.title=" + process.title); //ex: "... node  builtin.js"
console.log("id of this process=" + process.pid); //ex: 19544
console.log("id of parent process=" + process.ppid); //ex: 320
console.log(`Current working directory: ${process.cwd()}`);
console.log("Memory usage:" + JSON.stringify(process.memoryUsage()));

let xyz = process.env.XYZ  || "default_value_for_XYZ" ;
```

https://nodejs.org/api/process.html pour approfondir le sujet si besoin .

4.3. Mini module intégré "os" :

```
const os = require('node:os');
console.log(`os name is ${os.type()}`); //ex: Windows_NT or Linux or ...
console.log(`os platform is ${os.platform()}`); //ex: Win32 or ...
console.log(`user home directory is ${os.homedir()}`); //C:\Users\username or /home/username
console.log(`hostname is ${os.hostname()}`);
console.log(`map of ip address= ${JSON.stringify(os.networkInterfaces())}`);
console.log(`nb cpus = ${os.cpus().length}`);//ex: 8
console.log(`free memory=${os.freemem()} , total=${os.totalmem()}`);//ex: 8491798528, total=16497295360
```

https://nodejs.org/api/os.html pour approfondir le sujet si besoin .

V - Express (essentiel)

1. Essentiel de Express

1.1. Http sans express

Une application web basique basée que sur le coeur de nodeJs peut s'appuyer sur le module fondamental "**http**" *(toujours disponible , sans besoin de npm install).*

basic_http_server.js

```
//var http = require('http'); //ancienne syntaxe cjs
import http from 'http'; //module es2015 , typescript ou javascript

var myHttpFunction = function(req , res ) {
  res.writeHead(200 , {"Content-Type": "text/html"}); //OK=200
  res.write("<html> <body> <b> hello world </b> </body></html>")
  res.end();
};

var server = http.createServer(myHttpFunction);
server.listen(8282);  console.log("http://localhost:8282");
```

1.2. sur la route vers express

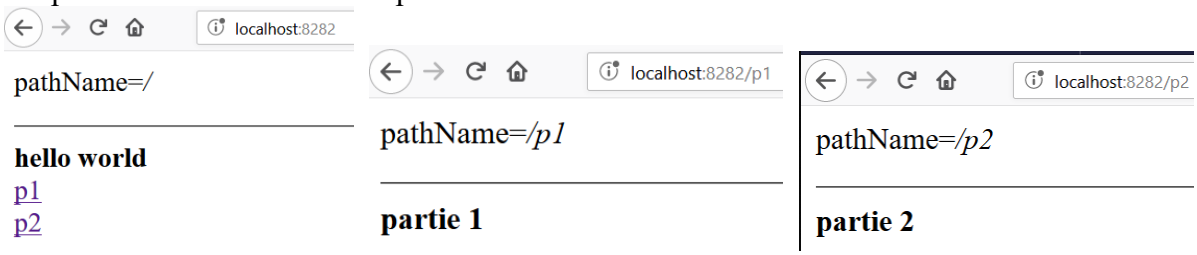
```
//var http = require('http'); //ancienne syntaxe cjs
//var url = require('url');
import http from 'http'; //module es2015 , typescript ou javascript
import url from 'url';

var myHttpFunction = function(req , res ) {
  res.writeHead(200 , {"Content-Type": "text/html"}); //OK=200
  res.write("<html> <body>");
  var pathName= url.parse(req.url).pathname; // "/" ou "/p1" ou "/p2"
  res.write("<p>pathName=<i>"+pathName+"</i></p><hr/>");
  switch(pathName){
    case '/p1' :
       res.write("<b> partie 1 </b>");    break;
    case '/p2':
       res.write("<b> partie 2 </b>");    break;
    case "/":
    default:
      res.write("<b> hello world </b><br/>");
      res.write("<a href='p1'>p1</a><br/>");
      res.write("<a href='p2'>p2</a><br/>");
  }
  res.write("</body></html>");
  res.end();
};
```

```
var server = http.createServer(myHttpFunction);
server.listen(8282);  console.log("http://localhost:8282");
```

Dans l'exemple de code précédent , l'instruction

```
var pathName= url.parse(req.url).pathname;
```

permet de récupérer dans la variable "pathName" la fin de l'url de la requête.
On peut ainsi différencier la réponse associée via un switch/case :



Pour avoir la même fonctionnalité (en beaucoup plus sophistiqué) sur un vrai projet , on utilise systématiquement le module/framework "**express**" permettant de différencier les réponses en fonctions de la fin de l'url de la requête entrante ("route vers un morceaux de code ou un autre").

1.3. Utilisation élémentaire de express (avec routes simples)

npm install -s express
→ ***"express": "^4.17.1"*** ou autre dans ***package.json***

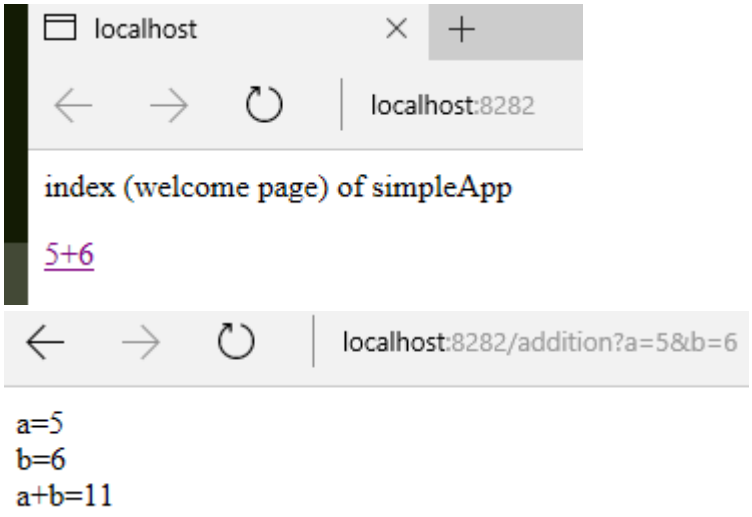
first_express_server.js

```
//modules to load:
//var express = require('express');
import express from 'express' ;

var app = express();

app.get('/', function(req, res , next) {
     res.setHeader('Content-Type', 'text/html');
     res.write("<html> <body>");
     res.write('<p>index (welcome page) of simpleApp</p>');
     res.write('<a href="addition?a=5&b=6">5+6</a>');
     res.write("</body></html>");
     res.end();
});

//GET addition?a=5&b=6
app.get('/addition', function(req, res , next) {
     let a = Number(req.query.a);     let b = Number(req.query.b);
     let resAdd = a+b;
     res.setHeader('Content-Type', 'text/html');
     res.write("<html> <body>");
     res.write('a=' + a + '<br/>');  res.write('b=' + b + '<br/>');
     res.write('a+b=' + resAdd + '<br/>');
     res.write("</body></html>");
     res.end();
});

app.listen(8282 , function () {
```

```
  console.log("simple express node server http://localhost:8282");
});
```

lancement : **node first_express_server.js**

via http://localhost:8282 au sein d'un navigateur web , on obtient le résultat suivant :



Adaptations pour le langage **typescript** :

first_express_server**.ts**

```
import express , { Request, Response } from 'express';

var app :express.Application = express();

app.get('/', function(req :Request, res :Response ) {
...
}
```

et **npm install --save-dev @types/express** en complément de *npm install -s express*

1.4. Gestion par express d'une partie "statique"

Si besoin (dans le haut de server.js):

```
import { dirname } from 'path';
import { fileURLToPath } from 'url';
const __dirname = dirname(fileURLToPath(import.meta.url));
```

server.js/ts

```
...
//les routes en /html/... seront gérées par express
//par de simples renvois des fichiers statiques du répertoire "./html"
app.use('/html', express.static(__dirname+"/html"));
...
```

Ceci permet à express de retourner directement fichiers nécessitants aucune interprétation coté serveur (ex: index.html , images , css , ...)

éventuelle route de redirection vers une page statique :

```
app.get('/', function(req , res  ) {
  res.redirect('/html/index.html');
});
```

Via cette route , en se connectant à *http://localhost:8282/* on se retrouve automatiquement redirigé vers la page statique *http://localhost:8282/html/index.html* (ex: SPA angular ou ...) .

NB : une page "statique coté serveur" pourra (lorsqu'elle fonctionnera (une fois téléchargée) au niveau du navigateur) appeler des web services REST de l'application courante "Node+express".

1.5. Expression des routes "express" :

app.**get**() , app.**post**() , app.**put**() , app.**delete**() , ... selon la méthode HTTP (GET , POST , PUT , DELETE) de la requête entrante .

1.6. Récupération des paramètres HTTP :

Pour récupérer les valeurs des paramètres véhiculés en fin d'URL en mode GET (ex : .../*addition*?***a=5&b=6***) , la syntaxe à employer est

```
var a = req.query.a ;
var b = req.query.b ;
```

La récupération des valeurs des paramètres HTTP véhiculés en mode **POST** dans le corps (***body***) de la requête entrante s'effectue avec la syntaxe **req.body.*paramName*** et nécessite la préparation et l'enregistrement d'un "***bodyParser***" :

```
...
//support parsing of application/x-www-form-urlencoded post data
app.use(express.urlencoded({ extended: true }))

...
//POST addition with body containing a=5&b=6 (application/x-www-form-urlencoded)
app.post('/addition', function(req : Request, res : Response) {
  let va = Number(req.body.a);
  let vb = Number(req.body.b);
  ...
});
```

2. Middleware to plugin (connect / express)

Le terme "**middleware**" signifie à peu près "**code au milieu**" et correspond à un **traitement intermédiaire** entre la **prise en charge de la requête entrante** et la **fabrication de la réponse**.

Le **concept** de "**middleware** *nodeJs/connect/express*" correspond un peu au **filtre** de l'api "servlet/jsp" du monde Java/JEE et peut aussi s'apparenter à un **intercepteur** .

Un "***middleware*** node/connect/express" peut également être vu comme un ***maillon/élément*** d'une implémentation du *design pattern* "***chain of responsability***" .

2.1. Historique (http , connect et express)

- à la base , la couche basse **http.Server** du module http
- ensuite le **framework "Connect"** (*à considérer comme l'ancètre de express*)
  qui a **popularisé le concept de middleware** (to plugin) dans le cycle de vie d'un traitement de (requête/réponse) au niveau d'un serveur nodeJs .
- **L'ancienne version 3 du framework express utilisait connect en interne .**
- **A partir de la v4 , express n'utilise plus connect en interne** mais en reprenant les **mêmes principes de fonctionnement et mêmes syntaxes que connect**, express 4 est en général compatible avec les middlewares écrits pour connect.

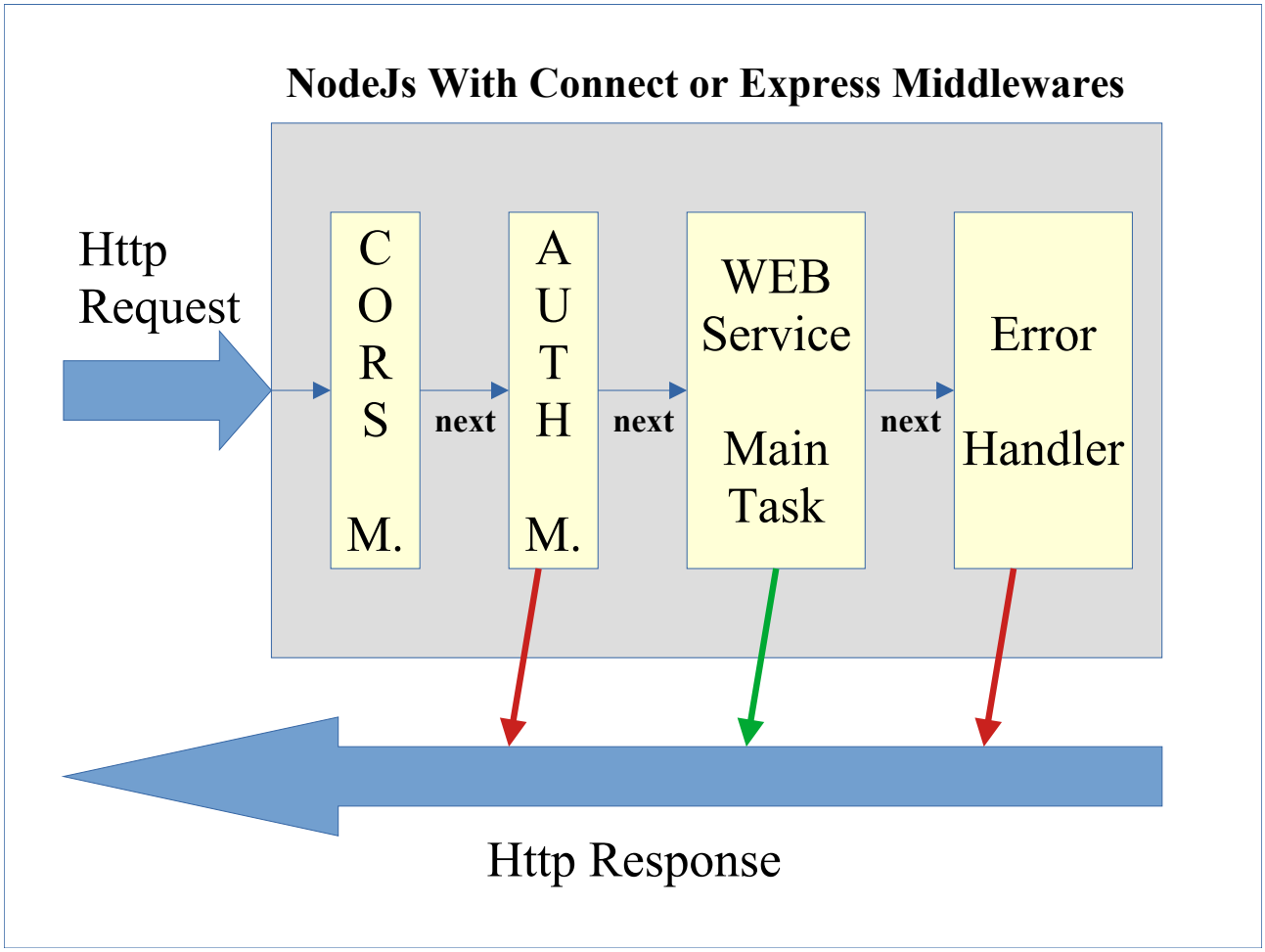
2.2. Plugin de middleware via app.use()

server.js

```
import express from 'express';
var app = express();
...
app.use(middleware_1);
app.use(middleware_2);
app.use(middleware_3);
app.use(middleware_4);

app.use(apiRouter_A);
app.use(apiRouter_B);
app.use(route_xyz);

app.use(error_handler);

…
app.listen(backendPort , function () {
  console.log("http://localhost:"+backendPort);
});
```

NB : l'ordre est souvent important !

2.3. Enchaînement classique de middleware



Au sein du schéma ci dessus, la requête entrante est gérée par l'application successive de plusieurs middlewares , d'un traitement principal et d'un éventuel traitement d'erreur automatisé :

1. le middleware "cors" ajoute si besoin des "autorisations CORS" dans l'entête de la réponse http.
2. le middleware "auth" gère l'authentification (par exemple en vérifiant la validité d'un jeton JWT) En cas d'authentification non vérifiée , ce middleware créer et retourne retourne directement une réponse négative (401=Not_Authorized ou 403=Forbidden) . Si l'authentification est réussie , un appel à next() permet d'enchaîner le traitement suivant.
3. le bloc traitement principal correspond par exemple au code d'un WebService REST (créer une réponse à une requête qui arrive en mode GET , POST , PUT ou DELETE).
4. un éventuel gestionnaire d'erreur (error handler) permet de construire automatiquement une réponse http négative (avec statut et message) en fonction du type d'exception/erreur remonté par le bloc de traitement précédent.

2.4. Détails sur les signatures et les rôles

type	*signature*	*rôle(s)*
Middleware *(pré-traitement* *ou* *traitement particulier)*	**func(req, res, next)**	- contrôles (auth, …) - analyse de la requête et ajout d'éléments dans celle-ci (ex : jsonParser) - tout autre pré-traitement utile (ex : gestion de l'upload-file, autorisations cors, …) - traitement particulier (ex: returning static files).
Route *(traitement principal unitaire)*	**func(req, res, next)**	Traitement principal unitaire du bloc (requête, réponse) selon route. Un potentiel appel à next(error) permet si besoin de déclencher l'enchaînement d'un traitement d'erreur.
Router *(sous-groupe de traitement)*	**app.use(routerXx)** *ou bien* **app.use("/pathYy",routerYy)**	Un router est un sous groupe de traitements (paquet de routes complémentaires), une sorte de mini application. Un router peut à son tour comporter des "sous-middlewares" spécifiques et peut être associé à un chemin (path) spécifique (ex : "/admin") .
Error Handler *(post traitement)*	**func(error,req, res, next)**	*En fin de chaîne* : post traitement consistant à frabriquer une réponse HTTP négative avec statut HTTP (500, 404, …) et message d'erreur en fonction du type d'erreur ou exception préalablement remontée.

2.5. Exemple de middleware

```
//basic middleware for checking apiKey:
app.use(function(req, res, next) {
  let  apiKey = req.query.apiKey;
  if(apiKey == null){
    res.status(400).send({ err : "apiKey missing in request" , example :  "?apiKey=abc123"})
    //...
    //plus other code for checkin apiKey in database ...
  }else  next();
});
```

```
app.get("/api-xy/xx" , function(req,res,next){
  res.send([ { id: 1 , message : "xx1" } , { id: 2 , message : "xx2" } ]);
});
```

Comportement:

http://localhost:8282/api-xy/xx

400 et {"err":"apiKey missing in request","example":"?apiKey=abc123"}

http://localhost:8282/api-xy/xx?apiKey=azerty2

200/OK et [{"id":1,"message":"xx1"},{"id":2,"message":"xx2"}]

2.6. Exemple de "error handler"

```
app.get("/api-xy/xx/:id" , function(req,res,next){
  let id = req.params.id;
  if(id==null || id <=0 ) next({errorType : "BAD_REQUEST" ,
                                message: "id should be a valid positive integer and not 0"})
  if( id > 999) next({errorType : "NOT_FOUND" , message: "not xx found with id="+id})
  res.send( { id: id , message : "xx"+id } );
});
```

```
//basic error handler:
app.use(function(error, req, res, next) {
  let status = 500;
  let errorType = error?error.errorType:null ;
  switch(errorType){
    case "BAD_REQUEST" : status = 400; break;
    case "NOT_FOUND" : status = 404; break;
    //...
    case "CONFLICT" : status = 409; break;
    default: status = 500;
  }
  res.status(status).send(error);
});
```

Comportement:

http://localhost:8282/api-xy/xx/0?apiKey=azerty2
400 et `{"errorType":"BAD_REQUEST","message":"id should be a valid positive integer and not 0"}`

http://localhost:8282/api-xy/xx/1200?apiKey=azerty2
404 et {"errorType":"NOT_FOUND","message":"not xx found with id=1200"}

http://localhost:8282/api-xy/xx/12?apiKey=azerty2

200/OK et `{"id":"12","message":"xx12"}`

3. Présentation de "express-generator"

express-generator (à intaller via npm en mode **-g**) , sert à **générer un squelette d'application "express"** (avec des options de type "*avec pug*" , ….) .

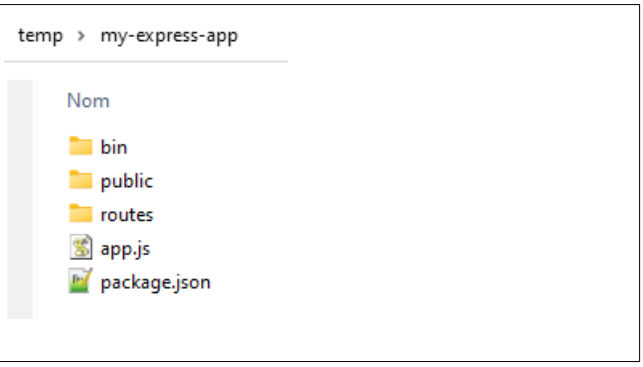
npm install -g express-generator

express -h

```
Usage: express [options] [dir]
Options:
      --version        output the version number
  -e, --ejs            add ejs engine support
      --pug            add pug engine support
      --hbs            add handlebars engine support
  -H, --hogan          add hogan.js engine support
  -v, --view <engine>  add view <engine> support (dust|ejs|hbs|hjs|jade|pug|twig|vash) (defaults to jade)
      --no-view        use static html instead of view engine
  -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
  -h, --help           output usage information
```

Mode opératoire :

1) créer un répertoire **my-express-app**
2) cd **my-express-app**
3) **express --view=hbs**
   ou bien
   express --view=pug
   ou bien
   express --view=ejs
   ou bien
   express --no-view
4) **npm install**
5) **npm start**



NB : ça génére du code en ancien style (CJS : require , …)
et basé sur des technologies "WEB coté serveur"
(ancienne architecture sans frontEnd SPA) !!!!!!!

4. Frameworks alternatifs à express (Hapi , ...)

Hapi.js est un framework concurrent de express (avec des choses en plus mais aussi des choses en moins , pas plus performant , ...)

Pour l'instant la plupart des projets/développeurs utilisent express (et sont "happy" avec express) .

5. Express avec template-engine (ex: handlebars)

NB:

- Ce paragraphe montre comment utiliser efficacement le framework express de manière à générer des pages HTML (un peu comme les vielles technologies concurrentes ASP , JPS et PHP du début des années 2000) .
- Bien que techniquement opérationnel , cela correspond à une architecture logicielle de plus en plus désuette / obsolète .
  Au sein des applications récentes (aux alentours de 2020) , le serveur "backend" a plutôt besoin de générer des réponses "JSON" à des appels de Web Services "REST" .

5.1. Utilisation d'un moteur de template (ex : EJS, handlebars)

Générer directement coté serveur des pages html à coup de ***res.write(" ... ")*** c'est fastidieux , peu lisible, peu maintenable et pas productif sur un projet sérieux.

Pour générer plus simplement des pages HTML coté serveur on peut :

- utiliser le design pattern "**MVC** (Model Vue Controller)"

- utiliser un "***moteur de templates***" (ex : ~~**jade**~~ **/ pug** , **EJS** , **Handlebars**, ...)

Principes de fonctionnement :

1. la requête HTTP est d'abord gérée par une route express ("contrôleur") permettant d'analyser les paramètres et d'effectuer (par délégation) des traitements appropriés
2. Le code final de la route express prépare un objet javascript ("modèle") permettant de passer des valeurs de la réponse à retourner vers le "template" de page HTML.
3. Durant la phase de rendu, le moteur de template effectue un remplacement des zones {{a}} ... {{b}} ... par les valeurs correspondantes de l'objet "modèle/js" passé en argument de la fonction **res.render**('view_name' , { a : ... , b : ... }) .

Moteurs de rendu	***Caractéristiques***
Handlebars (hbs)	Simple , peut également fonctionner du coté navigateur .
EJS	Très semblable au vrai standard HTML, syntaxe claire .
jade/pug	Avantage de jade/pug : **syntaxe très compacte**. Inconvients: de moins en moins utilisé car copier/coller moins facile depuis exemples html standard Remarque importante: le projet open-source **jade**(npm) à été **renommé "pug"** car jade est une marque déposée (avec nom réservé)

5.2. Exemple "Express + handlebars"

package.json

```
...
"dependencies": {
   "express": "^4.17.1",
   "express-handlebars": "^6.0.6"
 }
...
```

(à installer via "***npm install -s ...***") .

Structure appropriée des répertoires :

```
├── server.js
    assets
       css
          styles.css
└── views
    ├── server-home.handlebars
        calcul.handlebars
        addition.handlebars
    └── layouts
        └── main.handlebars
```

server.js/ts

```
//var express = require('express');
import express from 'express';

//var exphbs  = require('express-handlebars')

import exphbs from 'express-handlebars';

var app = express();

app.engine('handlebars', exphbs.engine());

app.set('view engine', 'handlebars');


import { dirname } from 'path';

import { fileURLToPath } from 'url';

const __dirname = dirname(fileURLToPath(import.meta.url));

app.use(express.static(__dirname+'/assets'))


app.get('/', function(req , res  ) {
  res.redirect('/server-home');
```

```
});

app.get('/server-home', function(req , res  ) {
    res.render('server-home');//rendering views/server-home.handlebars in the context of
                              //views/layouts/main.handlebars
});

app.get('/calcul', function(req , res  ) {
   res.render('calcul'); //views/calcul.handlebars
});

//GET addition?a=5&b=6
app.get('/addition', function(req , res ) {
  let va = Number(req.query.a);
  let vb = Number(req.query.b);
  let vaPlusVb = va+vb;
  res.render('addResult', { a : va ,
                            b: vb ,
                            resAdd : vaPlusVb });
  //rendering views/addResult.handlebars with js values for  {{a}} , {{b}} , {{resAdd}}
});

app.listen(8282 , function () {
  console.log("http://localhost:8282");
});
```

main template "views/layouts/***main.handlebars***"

```
<html>
<head>
   <meta charset="utf-8">
   <title>{{title}}</title>
   <link rel="stylesheet" href="css/styles.css">
</head>
<body>
   <div class="entete"><i>common header layout from views/layouts/main.handlebars</i></div>
   {{{body}}}
   <hr/>
   <div class="piedPage">
      <a href="/server-home" >server-home page</a>
        <a href="calcul">nouveau calcul</a> <br/>
      <!-- <a href="/html/index.html" >(main) index.html</a> -->
      <p><i>common footer layout from views/layouts/main.handlebars</i></p>
   </div>
</body>
```

```
</html>
```

assets/css/styles.css

```
.entete { background-color: azure;}
.piedPage { background-color: bisque;}
```

template "views/***server-home.handlebars***"

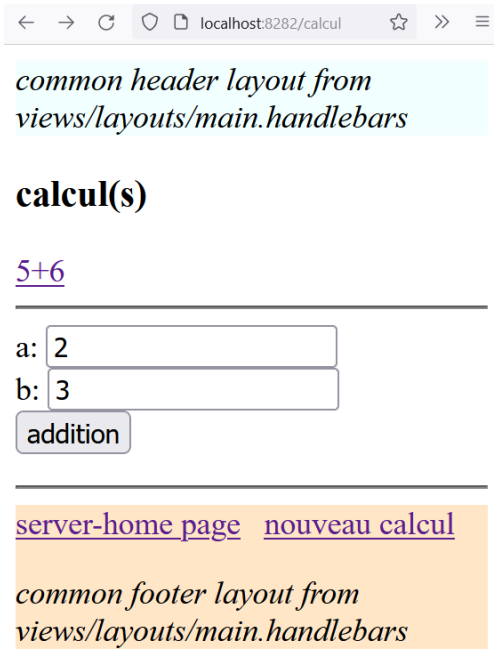
```
<p>...</p>  <a href="calcul"> nouveau calcul</a><br/>
```

template "views/***calcul.handlebars***"

```
<h3>calcul(s)</h3>
<a href="addition?a=5&b=6">5+6</a><br/>
...
<form method="GET" action="addition">
   a: <input name="a" value="2" /> <br/>
   b: <input name="b" value="3" /> <br/>
   <input type='submit' value="addition">
</form>
```

template "views/***addResult.handlebars***"

```
a={{a}} <br/>
b={{b}} <br/>
a+b=<b>{{resAdd}}</b><br/>
```



5.3. Aperçu "express + pug"

```
...
"dependencies": {
   "pug": "^3.0.2" ,
   "express": "^4.17.1"
 }
```

```
...
app.set('view engine', 'pug');//nécessite npm install -s pug
```

```
...
res.render('addResult', {a: va, b: vb, resAdd: vaPlusVb , couleurs : listeCouleurs });
   //rendering views/addResult.pug with js values
   //for #{a} , #{b} , #{resAdd} in .pug
```

views/addResult.pug

```
doctype html
html
 head
   title addResult
   link(rel='stylesheet' , href='css/styles.css')
 body
   h3 resultat du calcul
   p
     label a=
```

```
  span #{a}
 p
  label b =
  span #{b}
 p
  label a+b=
  span #{resAdd}
 hr
 if resAdd>=0
  span le resultat est positif
 else
  span le resultat est negatif
 hr
 ul
  each c in couleurs
   li #{c}
```

5.4. Aperçu "express + EJS"

```
...
"dependencies": {
   "ejs": "^3.1.6",
   "express": "^4.17.1"
 }
```

```
...
app.set('view engine', 'ejs');//nécessite npm install -s ejs
```

```
...
res.render('addResult', {a: va, b: vb, resAdd: vaPlusVb });
   //rendering views/addResult.ejs with js values
   //for <%=a%> , <%=b%> , <%=resAdd%>
```

views/addResult.ejs

```
<html>
<head>
   <meta charset="utf-8">    <title>addResult</title>
   <link rel="stylesheet" href="css/styles.css" />
</head>
```

```
<body>
   <h3>Resultats du calcul</h3>
   a=<%= a %> <br />
   b=<%= b %> <br />
   <p>a+b=<i> <%= resAdd %> </i></p>
    <%- include('./partials/footer'); %> <!-- ../partials/footer.ejs -->
</body>
</html>
```

VI - Web services REST avec express

1. WS REST élémentaire avec node+express

1.1. Récupérer des données entrantes au format JSON

La récupération des valeurs JSON véhiculées en mode **POST** ou **PUT** dans le corps (***body***) de la requête entrante s'effectue avec la syntaxe **req.body** et nécessite la préparation et l'enregistrement d'un "***bodyParser***" :

```
//var express = require('express');
import express from 'express' ;
var app = express();

//support parsing of JSON post data
var jsonParser = express.json({  extended: true});
app.use(jsonParser);
...
//POST ... with body { "firstname" : "Jean"  , "lastname" : "Bon" }
app.post('/xyz', function(req , res ) {
  let user = req.body ; //as javascript object via automatic use of jsonParser
//let user = JSON.parse(req.body); //si sans jsonParser
// ...
});
```

NB : il existe une variante de la méthode app.post() où l'on peut passer un "bodyParser" spécifique en second paramètre (pour certains cas pointus/spécifiques):

```
// POST /login gets urlencoded bodies :
app.post('/login', urlencodedParser, function (req, res) {
  res.send('welcome, ' + req.body.username)
})

// POST /api/users gets JSON bodies :
app.post('/api/users', jsonParser, function (req, res) {
  // use user in req.body
})
```

Dans le cas d'une api homogène (quasiment tout en JSON) , il est tout de même plus simple de paramétrer l'utilisation par défaut d'un bodyParser JSON via app.use() :

```
var jsonParser = express.json({  extended: true});
app.use(jsonParser)
```

1.2. Renvoyer des données/réponses au format JSON

La fonction prédéfinie ***res.send(jsObject)*** effectue en interne a peu près les opérations suivantes :

```
res.setHeader('Content-Type', 'application/json');
res.write(JSON.stringify(jsObject));
res.end();
```

Cette méthode ".send()" est donc tout à fait appropriée pour retourner la réponse "JSON" à un appel de Web Service REST .

1.3. Renvoyer si besoin des statuts d'erreur (http)

Via retour direct :

```
res.status(404).json({ error : "no product to update with code=" +code });
```

ou bien via "errorHandler" ...

1.4. récupération de paramètres

```
…

// GET (array) /minibank/operations?numCpt=1
app.get('/minibank/operations', function(req, res,next) {
  let numCpt =  Number(req.query.numCpt) ;
  // ...
});

// GET /minibank/compte/1
app.get('/minibank/compte/:numero', function(req, res,next) {
   let numCpt =  Number(req.params.numero) ;
     // ...
});

...
```

NB :

- **req.query.**pxy récupère la valeur d'un **paramètre http** ayant un **nom explicite** en fin d'URL (**?pxy=valXy**&pzz=valZz)
- **req.params**.idXy récupère la valeur d'un paramètre logique (avec:) et sans nom explicite en fin d'URL (ex : app.get("xyz/*:idXy*") et xyz/**2**)

2. Exemple simple (CRUD) sans base de données

server.js

```
//var express = require('express');
import express from 'express' ;

//var produitApiRoutes = require('./produit-api-routes');
import produitApiRoutes from './produit-api-routes-memory.js';

import { dirname } from 'path';
import { fileURLToPath } from 'url';
const __dirname = dirname(fileURLToPath(import.meta.url));

var app = express();

//support parsing of JSON post data
var jsonParser = express.json({  extended: true});
app.use(jsonParser);

//les routes en /html/... seront gérées par express
//par de simples renvois des fichiers statiques du répertoire "./html"
app.use('/html', express.static(__dirname+"/html"));

app.get('/', function(req , res ) {
  res.redirect('/html/index.html');
});

//delegate REST API routes to apiRouter(s) :
app.use(produitApiRoutes.apiRouter);
//app.use(otherApiRoutes.apiRouter);

app.listen(8282 , function () {
  console.log("http://localhost:8282");
});
```

produit-api-routes-memory.js

```
//var express = require('express');
import express from 'express' ;
const apiRouter = express.Router();

var allProduits = [];

allProduits.push({ code : 1 , nom : 'classeur' , prix : 4.0 });
allProduits.push({ code : 2 , nom : 'cahier' , prix : 2.1 });
allProduits.push({ code : 3 , nom : 'colle' , prix : 2.4 });
allProduits.push({ code : 4 , nom : 'stylo' , prix : 1.9 });

var codeMax=4; //pour simulation auto_incr

function findProduitInArrayByCode(produits,code){
        var produit=null;
        for(let i in produits){
```

```
                if(produits[i].code == code){
                        produit=produits[i]; break;
                }
        }
        return produit;
}



function removeProduitInArrayByCode(produits,code){
        var delIndex;
        for(let i in produits){
                if(produits[i].code == code){
                        delIndex=i; break;
                }
        }
        if(delIndex){
                produits.splice(i,1);
        }
}

function findProduitsWithPrixMini(produits,prixMini){
        var selProduits=[];
        for(let i in produits){
                if(produits[i].prix >= prixMini){
                        selProduits.push(produits[i]);
                }
        }
        return selProduits;
}

//exemple URL: http://localhost:8282/produit-api/public/produit/1
apiRouter.route('/produit-api/public/produit/:code')
.get( function(req , res  , next ) {
        var codeProduit = req.params.code;
        var produit = findProduitInArrayByCode(allProduits,codeProduit);
        res.send(produit);
});

//exemple URL: http://localhost:8282/produit-api/public/produit (returning all produits)
//          http://localhost:8282/produit-api/public/produit?prixMini=1.05
apiRouter.route('/produit-api/public/produit')
.get( function(req , res  , next ) {
        var prixMini = req.query.prixMini;
        if(prixMini){
                res.send(findProduitsWithPrixMini(allProduits,prixMini));
        }else{
          res.send(allProduits);
        }
});

// http://localhost:8282/produit-api/private/role-admin/produit en mode post
// avec { "code" : null , "nom" : "produitXy" , "prix" : 12.3 }
```

```
//ou bien {  "nom" : "produitXy" , "prix" : 12.3 }dans req.body
apiRouter.route('/produit-api/private/role-admin/produit')
.post( function(req , res  , next ) {
        var nouveauProduit = req.body;
        //simulation auto_incr :
        if(nouveauProduit.code == null){
                codeMax++; nouveauProduit.code = codeMax;
        }
        console.log("POST,nouveauProduit="+JSON.stringify(nouveauProduit));
        allProduits.push(nouveauProduit);
        //res.send(nouveauProduit); //with default status OK/200
        res.location(`/prod…./${nouveauProduit.code}`).status(201).send(nouveauProduit);
                        //201: successfully created
});

// http://localhost:8282/produit-api/private/role-admin/produit/1 en mode PUT
// avec { "code" : 1 , "nom" : "produit_xy" , "prix" : 16.3 } dans req.body
apiRouter.route('/produit-api/private/role-admin/produit/:id')
.put( function(req , res  , next ) {
        let idRes= req.params.id ; //code of produit/entity to update
        let newValueOfProduitToUpdate = req.body;
        console.log("PUT,newValueOfProduitToUpdate="
                        +JSON.stringify(newValueOfProduitToUpdate));
        var produitToUpdate =
                        findProduitInArrayByCode(allProduits,
                                                idRes /* newValueOfProduitToUpdate.code */);
        if(produitToUpdate!=null){
                produitToUpdate.nom = newValueOfProduitToUpdate.nom;
                produitToUpdate.prix = newValueOfProduitToUpdate.prix;
                res.send(produitToUpdate); //200:OK with updated entity as Json response body
                //res.status(204).send();//NO_CONTENT
        }else{
                res.status(404).json({ error : "no produit to update with code="
                                        + idRes /* newValueOfProduitToUpdate.code */ });
        }

});

// http://localhost:8282/produit-api/private/role-admin/produit/1 en mode DELETE
apiRouter.route('/produit-api/private/role-admin/produit/:code')
.delete( function(req , res  , next ) {
        var codeProduit = req.params.code;
        console.log("DELETE,codeProduit="+codeProduit);
        removeProduitInArrayByCode(allProduits,codeProduit);
        //res.send({ deletedProduitCode : codeProduit } );
        res.status(204).send(); // NO_CONTENT
});

// exports.apiRouter = apiRouter; // ancienne syntaxe common-js
export default { apiRouter }; // nouvelle syntaxe es2015
```

3. Eventuelles autorisations "CORS"

```
//var express = require('express');
import express from 'express' ;
var app = express();
…

// CORS enabled with express/node-js :
app.use(function(req, res, next) {
  res.header("Access-Control-Allow-Origin", "*");
                          //ou avec "www.xyz.com" à la place de "*" en production
  res.header("Access-Control-Allow-Methods",
                  "POST, GET, PUT, DELETE, PATCH"); //default: GET, ...
  res.header("Access-Control-Allow-Headers",
                 "Origin, X-Requested-With, Content-Type, Accept, Authorization");
  if(req.method === 'OPTIONS'){
     res.header('Access-Control-Allow-Methods', 'POST, GET, PUT, PATCH, DELETE');
                  //to give access to all the methods provided
     return res.status(200).json({});
  }
  next();
});

…
…
app.get(...) , app.post(...) , …

app.listen(8282 , function () {
  console.log("rest server with CORS enabled listening at 8282");
});
```

Cela peut quelquefois être pratique/utile durant la phase de développement (tant que pas d'intermédiaire "reverse-proxy" ni "api-gateway").

Attention :

A partir de cette page , tous les exemples de code sont en ancienne syntaxe (common-js ou bien typescript) .

4. Avec mode post et authentification minimaliste

```
var express = require('express');
var bodyParser = require('body-parser');   //dépendance indirecte de express via npm
var app = express();

var myGenericMongoClient = require('./my_generic_mongo_client');

var uuid = require('uuid'); //to generate a simple token
```

```
app.use(bodyParser.json()); // to parse JSON input data and generate js object : (req.body)
app.use(bodyParser.urlencoded({ extended: true}));

...

// POST /minibank/verifyAuth  {  "numClient" : 1 , "password" : "pwd1" }
app.post('/minibank/verifyAuth', function(req, res,next) {
        var verifAuth = req.body; // JSON input data as jsObject with ok = null
        console.log("verifAuth :" +JSON.stringify(verifAuth));
        if(verifAuth.password == ("pwd" + verifAuth.numClient) ){
                verifAuth.ok= true;
                verifAuth.token=uuid.v4(); //OU BIEN token en version JWT
                //éventuelle transmission parallèle via champ "x-auth-token" :
                res.header("x-auth-token", verifAuth.token);
                //+éventuel stockage dans une map pour vérif ultérieure : ....
                }
        else {
                verifAuth.ok= false;
                verifAuth.token = null;
        }
   res.send(verifAuth);    // send  back with ok = true or false and token
});

// GET /minibank/comptes/1
//app.get( route , pre_middleware1 , pre_middleware2 , main_action_function) ;
app.get('/minibank/comptes/:numero',
   displayHeaders,  verifTokenInHeaders  /*un peu securisé*/ ,
   function(req, res,next) {
        myGenericMongoClient.genericFindOne('comptes', { '_id' : Number(req.params.numero) },
                function(err,compte){
                        sendDataOrError(err,compte,res);
                });
});

function sendDataOrError(err,data,res){
        if(err==null) {
                        if(data!=null)
                                res.send(data);
                        else res.status(404).send(null);//not found
                }
                else res.status(500).send({error: err});//internal error (ex: mongo access)
}


//var secureMode = false;
var secureMode = true;

function extractAndVerifToken(authorizationHeader){
        if(secureMode==false) return true;
        /*else*/
        if(authorizationHeader!=null ){
                if(authorizationHeader.startsWith("Bearer")){
```

```
                        var token = authorizationHeader.substring(7);
                        console.log("extracted token:" + token);
                        //code extrememement simplifié ici:
                        //idealement à comparer avec token stocké en cache (si uuid token)
                        //ou bien tester validité avec token "jwt"
                        if(token != null && token.length>0)
                                return true ;
                         else
                                return false;
                }
                else
                        return false;
        }
        else
                return false;
}

// verif bearer token in Authorization headers of request :
function verifTokenInHeaders(req, res, next) {
  if( extractAndVerifToken(req.headers.authorization))
     next();
  else
          res.status(401).send(null);//401=Unautorized or 403=Forbidden
}

// display Authorization in request (with bearer token):
function displayHeaders(req, res, next) {
  //console.log(JSON.stringify(req.headers));
  var authorization = req.headers.authorization;
  console.log("Authorization: " + authorization);
  next();
}
...
app.listen(8282 , function () {
  console.log("minibank rest server listening at 8282");
});
```

5. Divers éléments structurants

5.1. ORDRE IMPORTANT et Routers en tant que modules annexes

server.ts

```
import express  from 'express';
import * as bodyParser from 'body-parser';
export const  app :express.Application = express();
import { apiErrorHandler} from './api/apiErrorHandler'
import { apiRouter } from './api/apiRoutes';
import { initSequelize } from './model/global-db-model'


//PRE TRAITEMENTS (à placer en haut de server.ts) !!!!

//support parsing of JSON post data
var jsonParser = express.json({  extended: true});
app.use(jsonParser);


//ROUTES ORDINAIRES (après PRE traitements , avant POST traitements)

app.use(apiRouter); //delegate REST API routes to apiRouter
//app.use(apiRouter2); //delegate REST API routes to apiRouter2


//POST TRAITEMENTS (à placer en bas de server.ts) !!!

app.use(apiErrorHandler); //pour gérer les erreurs/exceptions
                          //pas rattrapées par try/catch et qui se propagent
                          //alors automatiquement au niveau "express appelant mon code"
                          //ou bien pour gérer les erreurs explicitement déléguées ici via next(err) ;

export const server = app.listen(8282 , function () {
   console.log("http://localhost:8282");
   initSequelize(); // ou autre initialisation
});
```

api/apiRoutes.ts

```
import { Request, Response ,NextFunction, Router} from 'express';
import { Devise } from '../model/devise';
//import { ErrorWithStatus , NotFoundError, ConflictError} from '../error/errorWithStatus'
import { MemoryMapDeviseService } from '../dao/memoryMapDeviseService';
import { SqDeviseService } from '../dao/sqDeviseService';
import { DeviseDataService } from '../dao/deviseDataService';

export const apiRouter = Router();

var  deviseService : DeviseDataService = new SqDeviseService();
```

```
apiRouter.route('/devise/:code')
.get( function(req :Request, res :Response , next: NextFunction ) {
    let codeDevise = req.params.numero;
    deviseService.findById(codeDevise)
     .then((devise)=> { res.send(devise) })
    .catch((err)=>{next(err);} );
});

//POST ... with body { "code": "M1" , "nom" : "monnaie1" , "change" : 1.123 }
apiRouter.route('/devise').post( function(req :Request, res :Response , next: NextFunction ) {
   let  devise :Devise =  req.body ; //as javascript object
   //deviseService.insert(devise)
   deviseService.saveOrUpdate(devise)
   .then((savedDevise)=> { res.send(savedDevise)})
   .catch((err)=>next(err));
});


// DELETE http://localhost:8282/devise/EUR
apiRouter.route('/devise/:code')
.delete( function(req :Request, res :Response , next: NextFunction ) {
   let codeDevise = req.params.code;
   deviseService.deleteById(codeDevise)
   .then(()=> { res.status(200).send({ "action" : "devise with code="+codeDevise + " was
deleted"});})
   .catch((err)=>next(err));
});


// http://localhost:8282/devise renvoyant tout [ {} , {}]
// http://localhost:8282/devise?changeMini=1.1 renvoyant [{}] selon critere
apiRouter.route('/devise').get(function(req :Request, res :Response , next: NextFunction ) {
   let  changeMini = req.query.changeMini;
   deviseService.findAll()
   .then((deviseArray)=> {
      if(changeMini){
         //filtrage selon critère changeMini:
         deviseArray = deviseArray.filter((dev)=>dev.change >= changeMini);
      }
      res.send(deviseArray)
   })
   .catch((err)=>next(err));
});
```

5.2. gestionnaire d'erreurs simple

```
export function statusCodeFromEx(ex){
        let status = 500;
        let error = ex?ex.error:null ;
        switch(error){
                case "BAD_REQUEST" : status = 400; break;
                case "NOT_FOUND" : status = 404; break;
                //...
                case "CONFLICT" : status = 409; break;
                default: status = 500;
        }
        return status;
}
```

```
…
async function xxxxx(yyy,zzz) {
   ….
   if(...){
    ….
   }else
     throw { error : "NOT_FOUND" ,
             reason : "no entity to delete with id=" + idOfEntityToDelete }
 }
...
….
```

```
…

apiRouter.route("….")
        .get( async function(req , res  , next ) {
                let idRes = req.params.id;
                try {
                        … await ...
                         ...
                } catch(ex){
                        res.status(statusCodeFromEx(ex)).send(ex);
                }
        });
```

5.3. Documentation/définition de l'api

NB: l'annexe "**Api doc (swagger-ui-express et swagger-jsdoc)**" montre comment produire une documentation/description normalisée de l'api rest au format swagger/openApi .

6. Token JWT avec NodeJs

L'équivalent de "spring-security" pour NodeJs s'appelle "**passportjs**"

Le site officiel est "**http://www.passportjs.org/**"

passportjs s'intègre très bien dans une application "node + express" .
passportjs peut être considéré comme un mini framework très paramétrable à base plugins.

Il existe toutes sortes de **plugins** pour passportjs :

- pour jeton JWT
- pour OAuth2
....

VII - Accès aux bases de données (node)

1. Accès à MySQL via mysqljs/mysql (node)

1.1. Installation de mysqljs/mysql ou mysql2

Installation d'une version stable via

```
npm install --save mysql
```

ou bien

```
npm install --save mysql2
```

Exemple de version dans package.son :

```
...
"dependencies": {
  // "mysql": "^2.17.1",
    "mysql2": "^1.6.5",
   "sequelize": "^5.8.6"
 }
```

sites de référence : **https://github.com/mysqljs/mysql**
https://www.npmjs.com/package/mysql2

NB:

- mysql2 est une version améliorée de mysql d'un point de vue "performance" .
- l'api "mysql2" est en très grande partie compatible avec l'api "mysql".
- l'ORM "sequelize" nécessite (depuis la v4) l'utilisation de "mysql2"

1.2. Etablissement d'une connexion :

```
var mysql = require('mysql2');

var cnx = mysql.createConnection({
  host: "localhost",
  port: "3306",
  database : "minibank_db_node",
  user: "root",
  password: "root"
});

cnx.connect(function(err) {
  if (err) throw err;
  console.log("Connected!");
  onConnectedToMysqlDB(cnx);
});
```

D'autres possibilités sont envisageables : déclencher une query() sur une connexion non établie active automatiquement un .connect() interne .

1.3. Déconnexion

Déconnexion douce (le temps de traiter les requêtes en cours):

```
cnx.end( function(err) {
  // The connection is terminated now
});
```

ou bien plus brutalement : ***cnx.destroy() ;*** sans callback pour une déconnexion immédiate

1.4. Déclenchement de requetes

Exemple simple de déclenchement de requête :

```
function onConnectedToMysqlDB(cn){
        var sql="select * from Client";
        cn.query(sql, function (err, results ) {
                if (err) throw err;
                console.log("Results: " + JSON.stringify(results));
        });
}
```

==>

Results: [{"nom":"Therieur","prenom":"Alex","numClient":2,...}]

Plus précisément **cn.query(sqlString, callback)** permet de déclencher une requête SQL simple (sans remplacement de valeur de paramètres variables) tandis que

.query(sqlStringWithParam, valuesOfParams, callback)

permet de passer une requête comportant des ? (liés à des paramètres variables) et un tableau de valeurs pour ces paramètres .

Exemples :

```
connection.query('SELECT * FROM `books` WHERE `author` = "David"',
                 function (error, results, fields) {
...
}) ;
```

```
connection.query('SELECT * FROM `books` WHERE `author` = ?', ['David'],
                 function (error, results, fields) {
}) ;
```

NB :

- La requête sql peut soit être passée à .query() sous forme de "string" , soit passée comme un objet littéral javascript avec des options { sql : "..." , timeout : 40000 }
- Si une requête SQL simple ne possède qu'un seul paramètre (?) , on peut éventuellement

passer directement cette valeur sans les [ ] d'un tableau englobant .

Exemple :

```
connection.query({
   sql: 'SELECT * FROM `books` WHERE `author` = ?' ,
   timeout: 40000, // 40s
  },
  ['David'] /* ou 'David' */,
  function (error, results, fields) {
    ...
  }
);
```

Finalement la fonction callback comporte jusqu'à 3 paramètres

```
function (error, results, fields) {
  // error will be an Error if one occurred during the query
  // results will contain the results of the query
  // fields will contain information about the returned results fields (if any)
}
```

NB : `connection.escape()` est appelée en interne pour remplacer de façons sophistiquées les valeurs des ? variables de la requête sql (prise en compte des formats , de la sécurisation, ...)

Exemples :

```
connection.query('UPDATE users SET foo = ?, bar = ?, baz = ? WHERE id = ?',
                        ['a', 'b', 'c', userId], function (error, results, fields) {
                                if (error) throw error;
                                // ...
});
```

```
var post  = {id: 1, title: 'Hello MySQL'};
var query = connection.query('INSERT INTO posts SET ?', post, function (error, results, fields) {
        if (error) throw error;
        // ....
});
console.log(query.sql); // INSERT INTO posts SET `id` = 1, `title` = 'Hello MySQL'
```

Récupération de la valeur d'une clef primaire auto-incrémentée par le serveur MySQL:

```
cn.query('INSERT INTO Client SET ?', {prenom: 'Jean' , nom: 'Bon'}, function (error, results) {
            if (error) throw error;
            var autoIncrId = results.insertId;
            console.log("autoIncrId="+autoIncrId + " " + typeof autoIncrId); //autoIncrId=6 number
   });
```

NB : au sein de la callback, **results.affectedRows** permet de récupérer le nombre de lignes affectées par un ordre SQL de type insert, update ou delete .

Et dans le cas particulier d'un ordre UPDATE , **results.changedRows** permet de connaître le nombre de lignes dont les valeurs ont changé .

1.5. Quelques exemples "CRUD" :

```
const nouvelleAdresse = { idAdr : null , codePostal: '76000' , ville: 'Rouen' , rue : '123 rue xyz' };
cn.query('INSERT INTO Adresse SET ?', nouvelleAdresse, function (error, results, fields) {
      if (error) throw error;
            console.log('In table Adresse, last insert ID:', results.insertId);
   });
```

```
cn.query('SELECT * FROM Adresse', function (error, results, fields) {
      if (error) throw error;
            console.log('list of Adresse:', JSON.stringify(results));
   });
```

```
cn.query('UPDATE Adresse SET rue = ? Where idAdr = ?',
                              ['rue qui va bien', 3], (err, result) => {
                                    if (err) throw err;
                                    console.log(`Changed ${result.changedRows} row(s)`);
});
```

```
cn.query('DELETE FROM Adresse WHERE idAdr = ?', [4], (err, result) => {
   if (err) throw err;
    console.log(`Deleted ${result.affectedRows} row(s)`);
});
```

Attention : ne pas enchaîner ceci en mode synchrone mais en mode asynchrone (avec éventuelles "Promise" ou async/await) !!!

1.6. Transactions simples :

```
connection.beginTransaction(function(err) {
 if (err) { throw err; }
 connection.query('INSERT INTO posts SET title=?', title, function (error, results, fields) {
   if (error) {
     return connection.rollback(function() {
       throw error;
     });
   }

   var log = 'Post ' + results.insertId + ' added';

   connection.query('INSERT INTO log SET data=?', log, function (error, results, fields) {
```

```
    if (error) {
      return connection.rollback(function() {
        throw error;
      });
    }
    connection.commit(function(err) {
      if (err) {
        return connection.rollback(function() {
          throw err;
        });
      }
      console.log('success!');
    });
  });
});
```

.beginTransaction(), .commit() et .rollback() sont de simples fonctions qui déclenchent en interne les ordres sql "START TRANSACTION", "COMMIT" et "ROLLBACK" .

2. Accès à une base locale sqlite via nodeJs

NB : **sqlite** est une base relationnelle locale ne nécessitant pas de serveur (un peu comme H2 en java) et est très pratique en phase de tests / développement .

```
npm install -s sqlite3
```

(v5 en 2021)

produit-dao-sqlite.sql

```
var sqlite3 = require('sqlite3').verbose();

function withDbConnection(callbackWithDb){
  var db = new sqlite3.Database('mydb.db');
  callbackWithDb(db);
  db.close();
}

function init_produit_db(){
withDbConnection(function(db) {

   db.serialize(function() {
    // Produit with pk = code (integer)
    //nb , the column with INTEGER PRIMARY KEY is a alias
    //    for rowid implicit autoincr colomn of any slqlite table
    // do not use AUTOINCREMENT keyword with sqlite (not useful)

    db.run("CREATE TABLE if not exists produit (code INTEGER PRIMARY KEY , nom VARCHAR(64) NOT NULL , prix DOUBLE)");

    db.run("DELETE FROM produit");

    var pst = db.prepare("INSERT INTO produit(code,nom,prix) VALUES (?,?,?)");
    pst.run([1 , "Classeur" , 4.0]); //en [] , la liste des valeurs qui remplacent les ?
    pst.run([2 , "Cahier" , 2.1]);
    pst.run([3 , "Colle" , 2.4]);
    pst.run([4 , "Stylo" , 1.9]);
```

```
    pst.finalize();

    db.each("SELECT code,nom,prix FROM produit", function(err, row) {
       console.log(JSON.stringify(row));
    });
  }); //end de db.serialize() : déclenchements en mode séquentiel
 }); //end of withDbConnection()
}

function insert_new_produit(produit , cb_with_err_and_lastId){
 withDbConnection(function(db) {
  var pst = db.prepare("INSERT INTO produit (code, nom, prix) VALUES(?,?,?)");
  pst.run( [ null, produit.nom, produit.prix ], function(err) {
        cb_with_err_and_lastId(err,this.lastID)
      });
  pst.finalize();
 }); //end of withDbConnection()
}

function update_produit(produit , cb_with_err_and_nbChanges){
 withDbConnection(function(db) {
  var pst = db.prepare("UPDATE produit SET nom=? , prix=? WHERE code=?");
  pst.run( [ produit.nom, produit.prix ,produit.code ], function(err) {
         cb_with_err_and_nbChanges(err,this.changes)
      });
  pst.finalize();
 }); //end of withDbConnection()
}

function get_produits_by_WhereClause(whereClause , cb_with_err_or_res){
 withDbConnection(function(db) {
  let sql = "SELECT code,nom,prix FROM produit " + whereClause;
  db.all(sql, [], function(err, rows) {
        //console.log(JSON.stringify(rows));
        cb_with_err_or_res(err,rows)
      });
 }); //end of withDbConnection()
}

function get_produit_by_code(code , cb_with_err_or_res){
 withDbConnection(function(db) {
  let sql = "SELECT code,nom,prix FROM produit WHERE code=?";
  db.get(sql, code, function(err, row) {
        //console.log(JSON.stringify(row));
        cb_with_err_or_res(err,row)
      });
 }); //end of withDbConnection()
}

function delete_produit_by_code(code , cb_with_err_and_nbChanges){
 withDbConnection(function(db) {
  let sql = "DELETE FROM produit WHERE code=?";
  db.run(sql, code, function(err) {
        cb_with_err_and_nbChanges(err,this.changes)
      });
 }); //end of withDbConnection()
}
...
```

NB : **this.changes** correspond au nombres de lignes affectées (modifiées ou supprimées ou ...)

this.lastID correspond à la valeur de la dernière clef primaire auto-incrémentée (rowid ou alias)

La syntaxe **this.** est un utiliser ici dans des callbacks en **function(...){...}** mais ~~pas (...) => {...}~~

Utilisation depuis *produit-api-routes-sqlite.js* (code partiel) :

```
var express = require('express');
const apiRouter = express.Router();

var produit_dao_sqlite = require('./produit-dao-sqlite');

// exemple URL: http://localhost:8282/produit-api/public/produit
// returning all produits if no ?prixMini
// http://localhost:8282/produit-api/public/produit?prixMini=1.05
apiRouter.route('/produit-api/public/produit')
.get( function(req , res , next ) {
   var prixMini = Number(req.query.prixMini);
   var whereClause=prixMini?"WHERE prix >= "+prixMini : "";
         //console.log("whereClause="+whereClause);
         produit_dao_sqlite.get_produits_by_WhereClause(whereClause,function(err,produits){
                  if(err) {
                           console.log("err="+err);
               }
               res.send(produits);
         });//end of get_produits_by_WhereClause()
});

// http://localhost:8282/produit-api/private/role-admin/produit en mode post
// avec { "code" : null , "nom" : "produitXy" , "prix" : 12.3 }
//ou bien { "nom" : "produitXy" , "prix" : 12.3 }dans req.body
apiRouter.route('/produit-api/private/role-admin/produit')
.post( function(req , res , next ) {
   var nouveauProduit = req.body;
   console.log("POST,nouveauProduit="+JSON.stringify(nouveauProduit));
   produit_dao_sqlite.insert_new_produit (nouveauProduit,
      function(err,lastID){
         if(err==null){
           nouveauProduit.code = lastID;
           res.send(nouveauProduit);
         }
         else
           res.status(500).send({err : "cannot insert in database" ,  cause : err});
      });
});


// http://localhost:8282/produit-api/private/role-admin/produit/1
// en mode DELETE
apiRouter.route('/produit-api/private/role-admin/produit/:code')
.delete( function(req , res , next ) {
   var codeProduit = req.params.code;
   console.log("DELETE,codeProduit="+codeProduit);
   produit_dao_sqlite.delete_produit_by_code( codeProduit ,
      function(err,nbChanges){
         if(err || nbChanges ==0)
           res.status(404).send({ err : "not found , no delete" } );
         else
           res.send({ deletedProduitCode : codeProduit } );
      });
});
....
```

Accès à MongoDB (No-SQL , JSON) via node

2.1. Vue d'ensemble sur l'accès à mongoDB depuis nodeJs

Avec ou sans mongoose

Avec l'api de bas niveau "***mongodb/MongoClient***" seulement	**Code souple** (d'assez bas niveau) sans structuration rigoureuse des éléments stockés ou récupérés dans la base de données
Avec l'api "**mongoose**"	**Code plus strict (et d'un peu plus haut niveau) avec structuration rigoureuse des éléments situés dans la base de données**

Avec ou sans Promise & async/await

Avec des **callbacks** seulement	**C'est l'enfer des callbacks (surtout lorsque les algorithmes sont complexes)**
Avec **Promise & async/await**	**Code plus lisible/maintenable** ***(style de code devenu possible un peu après 2017 et conseillé aujourd'hui)***

2.2. Via mongodb/MongoClient (sans mongoose ni Promise)

Ce type de code (assez ancien) , uniquement basé sur des callbacks et sur l'api de bas niveau " mongodb/MongoClient" devient petit à petit de plus en plus déconseillé aujourd'hui .

my_generic_mongo_client.js

```
//myGenericMongoclient module (with MongoDB/MongoClient)
var MongoClient = require('mongodb').MongoClient;
var ObjectId = require('mongodb').ObjectID;

var mongoDbUrl = 'mongodb://127.0.0.1:27017/test'; //by default
var dbName = "test" //by default
var currentDb=null; //current MongoDB connection

var setMongoDbUrl = function(dbUrl){
        mongoDbUrl = dbUrl;
}

var setMongoDbName = function(mongoDbName){
        dbName = mongoDbName;
}

var closeCurrentMongoDBConnection = function(){
        currentDb.close();
        currentDb=null;
}

var executeInMongoDbConnection = function(callback_with_db) {
  if(currentDb==null){
```

```
  MongoClient.connect(mongoDbUrl, function(err, db) {
        if(err!=null) {
                console.log("mongoDb connection error = " + err + " for dbUrl=" + mongoDbUrl );
        }
        else{
          console.log("Connected correctly to mongodb database" );
          //currentDb = db; //with mongodb client v2.x
          currentDb = db.db(dbName);//with mongodb client >= v3.x
          callback_with_db(currentDb);
        }
        });
  }else{
        callback_with_db(currentDb);  //réutilisation de la connexion.
  }
}


var genericUpdateOne = function(collectionName,id,changes,callback_with_err_and_results) {
   executeInMongoDbConnection( function(db) {
          db.collection(collectionName).updateOne( { '_id' : id }, { $set :  changes } ,
                        function(err, results) {
                                if(err!=null) {
                                 console.log("genericUpdateOne error = " + err);
                                } else{
                                        if(results.matchedCount == 0)
                                          err = "no existing object with this id was found , no update"
                                }
                                callback_with_err_and_results(err,results);
                        });
         });
};

var genericInsertOne = function(collectionName,newOne,callback_with_err_and_newId) {
  executeInMongoDbConnection( function(db) {
          db.collection(collectionName).insertOne( newOne , function(err, result) {
                  if(err!=null) {
                    console.log("genericInsertOne error = " + err);
                    newId=null;
                  }
                  else {newId=newOne._id;
                  }
                  callback_with_err_and_newId(err,newId);
                });
  });
};

var genericFindList = function(collectionName,query,callback_with_err_and_array) {
    executeInMongoDbConnection( function(db) {
          var cursor = db.collection(collectionName).find(query);
          cursor.toArray(function(err, arr) {
                  callback_with_err_and_array(err,arr);
                });
  });
```

```
};


var genericFindOne = function(collectionName,query, callback_with_err_and_item) {
 executeInMongoDbConnection( function(db) {
          db.collection(collectionName).findOne(query , function(err, item) {
                if(err!=null) {
                console.log("genericFindById error = " + err);
            }
             callback_with_err_and_item(err,item);
             });
  });
};

var genericRemove = function(collectionName,query,callback_with_err_and_result) {
        executeInMongoDbConnection( function(db) {
                db.collection(collectionName).remove(query ,function(err, obj) {
                if(err!=null) {
                        console.log("genericRemove error = " + err);
                                }
                //if (err) throw err;
                console.log(obj.result.n + " document(s) deleted");
                callback_with_err_and_result(err,obj.result);
                });
  });
};

var genericDeleteOneById =
function(collectionName,mongoIdAsString,callback_with_err_and_booleanResult) {
        executeInMongoDbConnection( function(db) {
                db.collection(collectionName).deleteOne(
                     { '_id' : /*new ObjectID(*/mongoIdAsString } ,
                        function(err,deleteWriteOpResultObject) {
                            if(deleteWriteOpResultObject.deletedCount!=1) {
                              console.log("genericDeleteOneById --> no delete");
                                callback_with_err_and_booleanResult("no delete",false);
                            }
                        else {
                            console.log(" 1 document deleted");
                            callback_with_err_and_booleanResult(null,true);
                        }
                });
  });
};

exports.genericUpdateOne = genericUpdateOne;
exports.genericInsertOne = genericInsertOne;
exports.genericFindList = genericFindList;
exports.genericFindOne = genericFindOne;
exports.genericRemove = genericRemove;
exports.genericDeleteOneById = genericDeleteOneById;
exports.setMongoDbUrl= setMongoDbUrl;
exports.closeCurrentMongoDBConnection=closeCurrentMongoDBConnection;
```

NB : ce ***code*** *(ici en version assez basique)* est ***améliorable*** et doit être adapté en fonction du contexte .

Exemple d'utilisation (ici avec _id fixé selon code de Devise , pas généré automatiquement):

```
var express = require('express');
const apiRouter = express.Router();
var myGenericMongoClient = require('./my_generic_mongo_client');

function replace_mongoId_byCode(devise){
	devise.code = devise._id;	delete devise._id;	return devise;
}

function replace_code_byMongoId(devise){
	devise._id = devise.code;	delete devise.code;	return devise;
}

function replace_mongoId_byCode_inArray(deviseArray){
	for(i in deviseArray){
		replace_mongoId_byCode(deviseArray[i]);
	}
	return deviseArray;
}



//exemple URL: http://localhost:8282/devise-api/public/devise/EUR
apiRouter.route('/devise-api/public/devise/:code')
.get( function(req , res  , next ) {
	var codeDevise = req.params.code;
	myGenericMongoClient.genericFindOne('devises',
		{ '_id' : codeDevise },
		 function(err,devise){
			if(devise==null)
				res.status(404).send({ err : 'not found'});
			else
				res.send(replace_mongoId_byCode(devise));
		});

});

//exemple URL: http://localhost:8282/devise-api/public/devise (returning all devises)
//		http://localhost:8282/devise-api/public/devise?changeMini=1.05
apiRouter.route('/devise-api/public/devise')
.get( function(req , res  , next ) {
	var changeMini = Number(req.query.changeMini);
	var mongoQuery = changeMini ? { change: { $gte: changeMini }  } : { } ;
	//console.log("mongoQuery=" + JSON.stringify(mongoQuery));
	myGenericMongoClient.genericFindList('devises',mongoQuery,function(err,devises){
		res.send(replace_mongoId_byCode_inArray(devises));
	});//end of genericFindList()
});
```

```
// http://localhost:8282/devise-api/private/role-admin/devise en mode post
// avec { "code" : "mxy" , "nom" : "monnaieXy" , "change" : 123 } dans req.body
apiRouter.route('/devise-api/private/role-admin/devise')
.post( function(req , res  , next ) {
        var nouvelleDevise = req.body;
        console.log("POST,nouvelleDevise="+JSON.stringify(nouvelleDevise));
        //nouvelleDevise._id=nouvelleDevise.code;
        var nouvelleDevisePourMongoAvecId = replace_code_byMongoId(nouvelleDevise);
        myGenericMongoClient.genericInsertOne('devises',
           nouvelleDevisePourMongoAvecId,
                function(err,eId){
                        if(err==null && eId !=null)
                                res.send(replace_mongoId_byCode(nouvelleDevise));
                        else
                        res.status(500)
                            .send({err : "cannot insert in database" ,  cause : err});
                });
});

// http://localhost:8282/devise-api/private/role-admin/devise en mode PUT
// avec { "code" : "USD" , "nom" : "Dollar" , "change" : 1.123 } dans req.body
apiRouter.route('/devise-api/private/role-admin/devise')
.put( function(req , res  , next ) {
        var newValueOfDeviseToUpdate = req.body;
        console.log("PUT,newValueOfDeviseToUpdate="
                +JSON.stringify(newValueOfDeviseToUpdate));
        myGenericMongoClient.genericUpdateOne('devises',
        newValueOfDeviseToUpdate.code ,
        { nom : newValueOfDeviseToUpdate.nom ,
          change : newValueOfDeviseToUpdate.change} ,
        function(err,devise){
                if(err){
                        res.status(404).json({ err : "no devise to update with code="
                                        + newValueOfDeviseToUpdate.code });
                        }else{
                                res.send(newValueOfDeviseToUpdate);
                }
        });     //end of genericUpdateOne()
});

// http://localhost:8282/devise-api/private/role-admin/devise/EUR en mode DELETE
apiRouter.route('/devise-api/private/role-admin/devise/:code')
.delete( function(req , res  , next ) {
        var codeDevise = req.params.code;
        console.log("DELETE,codeDevise="+codeDevise);
        myGenericMongoClient.genericDeleteOneById('devises', codeDevise ,
        function(err,isDeleted){
                if(!isDeleted)
                  res.status(404).send({ err : "not found , no delete" } );
                else
           res.send({ deletedDeviseCode : codeDevise } );
   });
```

```
});

exports.apiRouter = apiRouter;
```

2.3. Via mongoose

Mongoose permet de mettre en œuvre une sorte de correspondance automatique entre un type d'objet javascript et un type de document "mongoDB" .

ODM : Object Document Mapping *(proche de la notion d' ORM pour les bases relationnelles)*

Mode opératoire :

- **on définit une structure de données** (mongoose.**Schema**(....)) **avec noms et types précis de propriétés (: String , : Number , ...)** et d'éventuels alias (pour _id ou autres).

- **on génère ensuite un modèle de classe javascript** à partir de ce schéma et de certains paramétrages (méthodes additionnelles , ...)

- **on peut ensuite utiliser le modèle en tant que "DAO" pour déclencher des opérations "CRUD" dans une base de données "mongoDB"**. ***On peut également appeler directement certaines méthodes telles que ".save() , .remove() , …" sur des instances du modèle considérées comme des objets potentiellement persistants.***

Avantages et inconvénients vis à vis de la solution précédente (via mongoClient) :

- **mongoose gère automatiquement l'ouverture des connexions à partir d'un appel CRUD effectué sur le modèle** (pas besoin d'enchaîner connection + ordre via des fonctions utilitaires)

- **mongoose apporte de la rigueur structurelle** via l'application d'un modèle précis

- les **alias** (pour _id ou autres) et les **plugins** (auto-incrémentaion de _id) de mongoose **apportent de la souplesse**.

- seul petit inconvénient : *mongoose* (en tant que sur-couche de mongoClient basée sur la définition de modèles précis) *se prête un peu moins bien à l'écriture d'un code générique (utilisable sur n'importe quelle collection)*

Installation de mongoose via npm :

npm install -s mongoose
(v6 en 2022 , v7 en 2023)

Précautions :

- A partir de la version 7 (2023) , mongose ne supporte que le nouveau style d'appel (avec Promise , sans callback de bas niveau).

- Attention à ce comportement de mongoose :

By default, Mongoose adds an _id property to your schemas.

You can also overwrite Mongoose's default _id with your own _id.
Just be careful: Mongoose will refuse to save a document that doesn't have an _id,
so you're responsible for setting _id if you define your own _id
If specific _id in schema ==> id value must be set by code before add/insert

2.4. Paramétrage de l'accès à la base de données (mongoose)

lancer-node.bat

```
REM set MONGO_DB_URL=mongodb://127.0.0.1:27017/test
set MONGO_DB_URL=mongodb+srv://dbAdmin:dba007!@cluster0.a570g.mongodb.net/test
REM nodemon server.js
node server.js
pause
```

db-mongoose.js

```
import mongoose from 'mongoose';

var mongoDbUrl = process.env.MONGODB_URL || "mongodb://127.0.0.1:27017"; //default
//ou bien dangeureusement "mongodb://superuser:motdepasse@127.0.0.1:27017"


console.log("mongoDbUrl="+mongoDbUrl);
mongoose.connect(mongoDbUrl, {useNewUrlParser: true,
                              useUnifiedTopology: true ,
                              user : "" , pass : "" ,
                              /*user: "username_telque_superuser" , pass : "motdepasse",*/
                              dbName : 'mydb'});
var thisDb  = mongoose.connection;

thisDb.on('error' , function() {
    console.log("mongoDb connection error = " + " for dbUrl=" + mongoDbUrl )
  });

thisDb.once('open', function() {
    // we're connected!
    console.log("Connected correctly to mongodb database" );
  });

export default { thisDb } ;
```

2.5. Paramétrage d'un schéma mongoose avec _id par défaut

produit-dao-mongoose.js

```
//var mongoose = require('mongoose');
import mongoose from 'mongoose'; // npm install -s mongoose

import dbMongoose from './db-mongoose.js' ;
var thisDb = dbMongoose.thisDb;

//NB: This is for current entity type ("Devise" or "Customer" or "Product" or ...)
//NB: thisSchema end ThisPersistentModel should not be exported (private only in this current module)
var thisSchema;//mongoose Shcema (structure of mongo document)
var ThisPersistentModel; //mongoose Model (constructor of persistent ThisPersistentModel)

function initMongooseWithSchemaAndModel () {
    mongoose.Connection = thisDb;
```

```
    thisSchema = new mongoose.Schema({
      /* default mongo _id: { type : ObjectId , alias : "id" } ,*/
      nom: String,
      prix : Number
    });
    thisSchema.set('id',true); //virtual id alias as string for _id
    thisSchema.set('toJSON', { virtuals: true ,
                               versionKey:false,
                               transform: function (doc, ret) {   delete ret._id  }
                             });

    //"Produit" model name is "produits" collection name in mongoDB database
    ThisPersistentModel = mongoose.model('Produit', thisSchema);
}

initMongooseWithSchemaAndModel();
//...
export default { ThisPersistentModel  }
```

```
 //defaut _id in schema : _id: { type: mongoose.ObjectId, alias: "id" },
// automatic mongoose default _id (of type mongoose.Types.ObjectId) will be automatically
// generated during add/insert
```

Ceci permet d'avoir en base :

```
{ _id: ObjectId("618d53514e0720e69e2e54c8"), nom: 'classeurB', prix: 4.4, __v: 0 }
```

et d'avoir dans les objets javascript/JSON :

```
{"nom":"classeurB","prix":4.4,"id":"618d53514e0720e69e2e54c8"}
```

2.6. Paramétrage d'un schéma mongoose avec _id spécifique

devise-dao-mongoose.js

```
//var mongoose = require('mongoose');
import mongoose from 'mongoose'; // npm install -s mongoose
import dbMongoose from './db-mongoose.js' ;
var thisDb = dbMongoose.thisDb;

var thisSchema;//mongoose Shcema (structure of mongo document)
var ThisPersistentModel; //mongoose Model (constructor of persistent ThisPersistentModel)

function initMongooseWithSchemaAndModel () {
    mongoose.Connection = thisDb;
    thisSchema = new mongoose.Schema({
      _id: { type : String , alias : "code" } ,
      name: String,
      change : Number
    });
    thisSchema.set('id',false); //no default virtual id alias for _id
    thisSchema.set('toJSON', { virtuals: true ,
                               versionKey:false,
```

```
                        transform: function (doc, ret) {   delete ret._id  }
                    });

    //"Devise" model name is "devises" collection name in mongoDB database
    ThisPersistentModel = mongoose.model('Devise', thisSchema);
}

initMongooseWithSchemaAndModel();
//...
export default { ThisPersistentModel  }
```

db.devises.find(); //depuis un shell de mongodb (pour visualiser contenu en base)

```
{ _id: 'USD', name: 'Dollar', change: 1.1 }
{ _id: 'EUR', name: 'Euro', change: 1.0 }
{ _id: 'GBP', name: 'Livre', change: 0.9}
{ _id: 'JPY', name: 'Yen', change: 123.7 }
```

NB : le __v quelquefois généré par mongoose correspond à un champ technique "versionKey"

Vision coté navigateur client de l'api-rest :

```
[{"name":"Dollar","change":1.1,"code":"USD"},
{"name":"Euro","change":1,"code":"EUR"},
{"name":"Livre","change":0.9,"code":"GBP"},
{"name":"Yen","change":123.7,"code":"JPY"}, ...
]
```

NB: au sein de cet exemple, dans la base mongoDB la collection devises comporte une clef primaire **_id** correspondant au code de la monnaie (pas d'auto génération prévue).
La vision externe (coté api REST) correspond à l'**alias "code"** considéré par mongoose comme un champ virtuel (d'où le paramétrage ***virtuals : true*** dans ***deviseSchema***) .
.toJSON() paramétré au niveau de *deviseSchema* est en interne utilisé par JSON.stringify() .

2.7. Eventuelle réinitialisation d'un jeu de données

xyz-dao-mongoose.js

```
…
function reinit_db(){
  return new Promise( (resolve,reject)=>{
    const deleteAllFilter = { }
    ThisPersistentModel.deleteMany( deleteAllFilter)
            .then(()=>{ //insert elements after deleting olds
      (new ThisPersistentModel({ _id : '618d53514e0720e69e2e54c8' ,
                                 nom : "classeur" , prix : 4.0 })).save();
      (new ThisPersistentModel({ _id : '618d53514e0720e69e2e54c9' ,
                                 nom : "cahier" , prix : 2.1 })).save();
      //(new ThisPersistentModel({ code : "EUR" , name : "Euro" , change : 1.0})).save();
      //(new ThisPersistentModel({ code : "USD" , name : "Dollar" , change : 1.1})).save();
      resolve({action:"xyz collection re-initialized in mongoDB database"})
    })
    .catch((err)=>{ console.log(JSON.stringify(err)) ;
      reject({error : "cannot delete in database" , cause : err}); }  );
  });
}
```

```
export default { reinit_db , ...};
```

2.8. Utilisation directe/classique du modèle de persistance:

xyz-api-routes.js

```
//var express = require('express');
import express from 'express';
const apiRouter = express.Router();

import xyzDao from './xyz-dao-mongoose.js';

var PersistentXyzModel =  xyzDao.ThisPersistentModel; ;
....
//recherche par id/pk :
 let xyz = await PersistentXyzModel.findById(idXyz );


//recherche multiple via critères:
 let criteria=prixMini?{ prix: { $gte: prixMini } }:{};  //$gte signifie greater or equal
 let xyzArray = await PersistentXyzModel.find(criteria );

//ajout d'une nouvelle entité en base :
//si custom _id and code alias in schema :
//  nouveauXyz.code = new mongoose.Types.ObjectId();
//sinon _id généré automatiquement
let persistentXyz = new PersistentXyzModel(nouveauXyz);
let savedXyz = await persistentXyz.save();


//mise à jour d'une entité :
let filter = { _id : newValueOfXyzToUpdate.id };
await PersistentXyzModel.updateOne(filter ,newValueOfXyzToUpdate );

//suppression d'une entité :
const filter = { _id : idXyz };
await PersistentXyzModel.deleteOne(filter);
```

2.9. mongoose encapsulé dans DAO :

Au lieu d'utiliser mongoose en direct , on peut éventuellement encapsuler le modèle de persistance mongoose dans un **DAO (Data Access Object) personnalisé** pour une ou plusieurs des raisons suivantes :

- ***ne pas être directement dépendant des méthodes imposées par une version précise de mongoose (v5 ou v6 ou v7)*** et pouvoir éventuellement basculer ultérieurement sur autre technologie.
- ***automatiser la gestion des cas d'erreurs en remontant des exceptions dans un format bien uniforme , bien contrôlé*** de manière à pouvoir éventuellement mettre en place un

middleware express de type "gestionnaire automatique d'erreur" .

generic-promise-mongoose.js

```
//V2 (essentiel ok , compatible mongoose 7 sans callback) - peaufinable

function findByIdWithModel(id,PersistentModel) {
   return new Promise( (resolve,reject)=>{
     PersistentModel.findById( id )
        .then((entity)=>{ if(entity == null)
                          reject({error:'NOT_FOUND' , reason : "no entity found with id="+id});
                        else resolve(entity);
                       })
        .catch((err)=>{ reject({error:'can not find by id' , cause : err}); }  );

   });
 }



 //exemple of criteria : {} or { unitPrice: { $gte: 25 } } or ...
 function findByCriteriaWithModel(criteria,PersistentModel) {
  return new Promise( (resolve,reject)=>{
    PersistentModel.find( criteria)
               .then((entities)=>{ resolve(entities);  })
               .catch((err)=>{ reject({error:'can not find' , cause : err}); }  );
    });
 }

 function saveWithModel(entity,PersistentModel) {
  return new Promise( (resolve,reject)=>{
    let  persistentEntity = new PersistentModel(entity);
    persistentEntity.save()
              .then((savedEntity)=>{ entity.id = savedEntity.id;  resolve(entity);  })
              .catch((err)=>{
                if(err && err.code == 11000)
                  reject({ error : "CONFLICT" , reason : "existing entity with same id"  });
                else
                  reject({error : "cannot insert in database" , cause : err});
               }  );
  });
 }

 function updateOneWithModel(newValueOfEntityToUpdate,idOfEntityToUpdate,PersistentModel) {
  return new Promise( (resolve,reject)=>{
    const filter = { _id :  idOfEntityToUpdate };
    //console.log("filter of updateOne=" +JSON.stringify(filter));
    PersistentModel.updateOne(filter , newValueOfEntityToUpdate)
              .then((opResultObject)=>{
                console.log("opResultObject of updateOne=" +JSON.stringify(opResultObject))
                if(opResultObject.matchedCount == 1)
                   resolve(newValueOfEntityToUpdate);
                else reject({ error : "NOT_FOUND" ,
                              reason : "no entity to update with id=" + idOfEntityToUpdate });
                })
              .catch((err)=>{ reject({ error : "cannot updateOne " , cause :  err}); }  );
  });
```

```
 }

 function deleteOneWithModel(idOfEntityToDelete,PersistentModel) {
  return new Promise( (resolve,reject)=>{
    const filter = { _id : idOfEntityToDelete };
    console.log("filter of deleteOne=" +JSON.stringify(filter));
    PersistentModel.deleteOne(filter )
            .then((opResultObject)=>{
              console.log("opResultObject of deleteOne=" +JSON.stringify(opResultObject))
              if(opResultObject.deletedCount == 1) resolve({ deletedId : idOfEntityToDelete });
                else reject({ error : "NOT_FOUND" ,
                              reason : "no entity to delete with id=" + idOfEntityToDelete });
              })
            .catch((err)=>{ reject({ error : "cannot delete " , cause :  err}); }  );
  });
 }


export default { findByIdWithModel , findByCriteriaWithModel , saveWithModel , updateOneWithModel
, deleteOneWithModel }
```

NB : Au sein des callbacks de mongoose **opResultObject.matchedCount** ou **opResultObject.deletedCount** correspond au nombre d'éléments affectés (modifiés ou supprimés ou ...) par la requête qui vient d'être déclenchée.

URL de la documentation officielle de mongoose (pour approfondir) :
https://mongoosejs.com/docs/guide.html

Dao personnalisé (xyz = devise ou product ou customer ou …)

xyz-dao-mongoose.js

```
import mongoose from 'mongoose';
import dbMongoose from './db-mongoose.js';
import genericPromiseMongoose from './generic-promise-mongoose.js';

var thisDb = dbMongoose.thisDb;

//NB: This is for current entity type ("Devise" or "Customer" or "Product" or ...)
//NB: thisSchema end ThisPersistentModel should not be exported (private only in this current module)
var thisSchema;//mongoose Shcema (structure of mongo document)
var ThisPersistentModel; //mongoose Model (constructor of persistent ThisPersistentModel)

function initMongooseWithSchemaAndModel () {
    //...
    ThisPersistentModel = mongoose.model('Xyz', thisSchema);
}

initMongooseWithSchemaAndModel();


function findById(id) {
```

```
  return genericPromiseMongoose.findByIdWithModel(id,ThisPersistentModel);
}


function findByCriteria(criteria) {
  //exemple of criteria : {} or { unitPrice: { $gte: 25 } } or ...
  return genericPromiseMongoose.findByCriteriaWithModel(criteria,ThisPersistentModel);
}

function save(entity) {
  return genericPromiseMongoose.saveWithModel(entity,ThisPersistentModel);
}

function updateOne(newValueOfEntityToUpdate) {
  return genericPromiseMongoose.updateOneWithModel(newValueOfEntityToUpdate,
                                                   newValueOfEntityToUpdate.id,
                                                   ThisPersistentModel);
}

function deleteOne(idOfEntityToDelete) {
  return genericPromiseMongoose.deleteOneWithModel(idOfEntityToDelete,ThisPersistentModel);
}


export default { ThisPersistentModel ,  findById , findByCriteria , save , updateOne ,  deleteOne};
```

Gestion uniforme des cas d'erreurs :

generic-express-util.js

```
export function statusCodeFromEx(ex){
        let status = 500;
        let error = ex?ex.error:null ;
        switch(error){
                case "BAD_REQUEST" : status = 400; break;
                case "NOT_FOUND" : status = 404; break;
                //...
                case "CONFLICT" : status = 409; break;
                default: status = 500;
        }
        return status;
}


export function nullOrEmptyObject(obj){
        return obj==null || ( Object.keys(obj).length === 0 && obj.constructor === Object );
}
```

2.10. Utilisation du dao mongoose dans routes "express"

produit-api-routes.js (ou xyz-api-routes.js)

```
import express from 'express';
const apiRouter = express.Router();
import { statusCodeFromEx , nullOrEmptyObject } from "./generic-express-util.js";
import produitDao from './produit-dao-mongoose.js';

//PersistentProduitModel  to use only for specific extra request (not in dao)
var PersistentProduitModel = produitDao.ThisPersistentModel;

/*
conventions d'URL :
http://localhost:8282/produit-api/private/xyz en accès private (avec auth nécessaire)
http://localhost:8282/produit-api/public/xyz en accès public (sans auth nécessaire)

NB: dans vrai projet d'entreprise , public pour get pas confidentiel et private pour tout le reste
   ICI Exceptionnellement EN TP , presques toutes les URLs sont doublées : appelables en public et private

NB2: par défaut les requetes en mode DELETE ou PUT retourneront "204/NoContent" quand tout se passe bien
    via l'option facultative ?v=true (au sens verbose=true) la réponse sera 200/OK accompagné d'un message json
*/

//exemple URL: http://localhost:8282/produit-api/public/reinit
apiRouter.route(['/produit-api/public/reinit','/produit-api/private/reinit'])
.get( async function(req , res  , next ) {
    try{
        let doneActionMessage = await produitDao.reinit_db();
        res.send(doneActionMessage);
   } catch(ex){
            res.status(statusCodeFromEx(ex)).send(ex);
   }
});


//exemple URL: http://localhost:8282/produit-api/public/produit/618d53514e0720e69e2e54c8
apiRouter.route('/produit-api/public/produit/:id')
.get( async function(req , res  , next ) {
        var idProduit = req.params.id;
        try{
                let Produit = await produitDao.findById( idProduit);
                res.send(Produit);
   } catch(ex){
            res.status(statusCodeFromEx(ex)).send(ex);
   }
});

// exemple URL: http://localhost:8282/produit-api/public/produit
// returning all produits if no ?prixMini
// http://localhost:8282/produit-api/public/produit?prixMini=1.05
apiRouter.route('/produit-api/public/produit')
.get( async function(req , res  , next ) {
        var prixMini = req.query.prixMini;
        var criteria=prixMini?{ prix: { $gte : prixMini } }:{};
        try{
                let produits = await produitDao.findByCriteria(criteria);
```

```
                res.send(produits);
    } catch(ex){
            res.status(statusCodeFromEx(ex)).send(ex);
    }
});


// http://localhost:8282/produit-api/private/produit en mode post
// avec { "id" : null , "nom" : "produitXy" , "prix" : 12.3 }
//ou bien { "nom" : "produitXy" , "prix" : 12.3 } dans req.body
apiRouter.route([ '/produit-api/public/produit',  '/produit-api/private/produit'])
.post(async function(req , res  , next ) {
        let newEntity = req.body;
        //console.log("POST,newEntity="+JSON.stringify(newEntity));
        if(nullOrEmptyObject(newEntity)) { res.status(400).send(); return; } //BAD REQUEST
        try{
                let savedEntity = await produitDao.save(newEntity);
                let id = newEntity.id ; //saved id (sometimes auto_incr id)
                //NB: res.location('/produit/' + id) because some clients may send two calls:
                //1. a post call to create new resource on server
                //   the server respond 201 with Location: /produit/pxy in http response header
                //2. the client may send a get request with /produit/pxy at url end to retreive full entity value
                res.location('/produit/' + id).status(201).send(savedEntity);//201: successfully created
    } catch(ex){
            res.status(statusCodeFromEx(ex)).send(ex);
    }
});



// http://localhost:8282/produit-api/private/produit en mode PUT
// ou bien http://localhost:8282/produit-api/private/produit/618d53514e0720e69e2e54c8 en mode PUT
// avec { "code" : "618d53514e0720e69e2e54c8" , "nom" : "produit_xy" , "prix" : 16.3 } dans req.body
apiRouter.route([ '/produit-api/public/produit','/produit-api/public/produit/:id',
                  '/produit-api/private/produit','/produit-api/private/produit/:id'])
.put( async function(req , res  , next ) {
    let newValueOfEntityToUpdate = req.body;
    //console.log("PUT,newValueOfEntityToUpdate="+JSON.stringify(newValueOfEntityToUpdate));
    if(nullOrEmptyObject(newValueOfEntityToUpdate))
                { res.status(400).send(); return; //BAD REQUEST }
    //l'id de l'entity à mettre à jour en mode put peut soit être précisée en fin d'URL
    //soit être précisée dans les données json de la partie body
    //et si l'information est renseignée des 2 façons elle ne doit pas être incohérente:
   let entityId = req.params.id; //may be found (as string) at end of URL
   if(newValueOfEntityToUpdate.id != null && entityId != null
    && newValueOfEntityToUpdate.id != entityId ) { res.status(400).send(); return; } //BAD REQUEST (incoherent id)
   if(newValueOfEntityToUpdate.id == null && entityId != null) newValueOfEntityToUpdate.id = entityId;
   if(newValueOfEntityToUpdate.id != null && entityId == null ) entityId = newValueOfEntityToUpdate.id;

   let verbose = req.query.v=="true"; //verbose mode (default as false)
        try{
                let updatedEntity = await produitDao.updateOne(newValueOfEntityToUpdate);
                if(verbose)
                  res.send(updatedEntity); //200:OK with updated entity as Json response body
                else
                  res.status(204).send();//NO_CONTENT
    } catch(ex){
```

```
            res.status(statusCodeFromEx(ex)).send(ex);
    }
});


// http://localhost:8282/produit-api/private/produit/618d53514e0720e69e2e54c8
//avec ou sans    ....?v=true en mode DELETE
apiRouter.route([ '/produit-api/private/produit/:id', '/produit-api/public/produit/:id'])
.delete( async function(req , res  , next ) {
        let entityId = req.params.id;
        //console.log("DELETE,entityId="+entityId);
        let verbose = req.query.v=="true"; //verbose mode (default as false)
        try{
                let deleteActionMessage = await produitDao.deleteOne(entityId);
                if(verbose)
                  res.send(deleteActionMessage);
                else
                  res.status(204).send();//NO_CONTENT
    } catch(ex){
            res.status(statusCodeFromEx(ex)).send(ex);
    }
});

export  default { apiRouter };
```

2.11. Paramétrages avancés d'un schéma mongoose

Schéma mongoose avec sous-schéma :

```
let addressSchema = new  mongoose.Schema({
      num: String,
      street : String,
      zipCode : String,
      town : String,
      country : String
    });
    addressSchema.set('id',false); //no default virtual id alias as string for _id
    addressSchema.set('toJSON', { virtuals: true ,
                          versionKey:false,
                          transform: function (doc, ret) {   delete ret._id;   }
                         });

 let   thisSchema = new mongoose.Schema({
      /* default mongo _id: { type : ObjectId , alias : "id" } ,*/
      firstName: String,       lastName : String,
      username : { type: String , unique : true },
      birthDay : String,       email : String,       mobilePhoneNumber : String ,
      address : addressSchema
     /* address : [ addressSchema ] // if array of address */
    });
    thisSchema.set('id',true); //default virtual id alias as string for _id
    thisSchema.set('toJSON', { virtuals: true ,
                          versionKey:false,
                          transform: function (doc, ret) {   delete ret._id; delete ret._v;  }
                         });

  //"Customer" model name is "customers" collection name in mongoDB customer_db database
 let   ThisPersistentModel = mongoose.model('Customer', thisSchema);
```

Ceci permet d'écrire cela en javascript :

```
(new ThisPersistentModel({ _id: "618d54d5386fcff631470c78" , firstName : "jean" ,
lastName : "Bon" ,  username : "jeanBon" , birthDay : "1977/02/11" ,  email :
"jean.bon@labas.fr" , mobilePhoneNumber : "0601020304", address : { num: "23" , street :
"rue abc", zipCode : "75000" , town : "Paris" , country : "France"}})).save();
```

et d'obtenir cela dans la base de données :

```
{ _id: ObjectId("618d54d5386fcff631470c78"),
  firstName: 'jean',  lastName: 'Bon',  username: 'jeanBon',  birthDay: '1977/02/11',
  email: 'jean.bon@labas.fr',  mobilePhoneNumber: '0601020304',
  address:
   { num: '23',     street: 'rue abc',     zipCode: '75000',     town: 'Paris',     country: 'France',
     _id: ObjectId("61969c7e0d5feb3ec8b7f14e") },
  __v: 0 }
```

VIII - Événements , File system , Streams

1. Essentiel sur "Events nodeJs"

Le module intégré "**events**" de nodeJs permet d'**établir des communications asynchrones** entre différentes parties d'une application (potentiellement réparties dans plusieurs modules).

```
import events from'events';
var myEventEmitter = new events.EventEmitter();
```

Le déclenchement d'un événement personnalisé s'effectue via la méthode .**emit**() :

```
myEventEmitter.emit('custom-eventName', [arg1] , ... , [argN] );
```

où les factultatifs arguments peuvent éventuellement être des objets .

La réception/écoute de chaque nouvelle occurrence de l'événement s'effectue via la méthode .**on**()

```
myEventEmitter.on('custom-eventName', function([evt_arg1] , ... , [evt_argN]){ ....}) ;
```

NB :

- **.once()** est une variante de .on() qui n'est **déclenchée qu'une fois lors de la première occurrence de l'événement.**
- Attention: events.EventEmitter ne gère pas de singleton en interne.
  Pour que plusieurs modules puissent se synchroniser sur une même instance , on pourra par exemple faire en sorte qu'un module exporte une instance de EventEmitter et que les autres modules intéressés importe cette unique instance préalablement exportée .

1.1. Exemple :

with-fire-event.mjs

```
import events from'events';
import express from 'express';

const apiRouter = express.Router();
var counterEventEmitter = new events.EventEmitter();

var counter = 0;

apiRouter.route('/api-xy/xx')
.get( function(req , res  , next ){
   counter++;
   counterEventEmitter.emit('new-counter',{counter:counter});
   console.log("new-counter event was fire with counter="+counter)
   res.send([ { id: 1 , message : "xx1" } , { id: 2 , message : "xx2" } ]);
});

export  default { apiRouter , counterEventEmitter};
```

with-event-handler.mjs

```
import express from 'express';
import with_fire_event from './with-fire-event.mjs';
const apiRouter = express.Router();

var comment = "no comment";
// .on() ré-appelé à chaque nouvelle occurrence de l'événement
with_fire_event.counterEventEmitter.on('new-counter', function(evt){
   console.log("received event: " + JSON.stringify(evt));
   comment = "new-counter="+evt.counter;
});

// .once() appelé qu'au moment de la première occurrence de l'événement
with_fire_event.counterEventEmitter.once('new-counter', function(evt){
   console.log(">>> new-counter now not 0: " + JSON.stringify(evt));
});

apiRouter.route('/api-cc/comment')
.get( function(req , res  , next ){
   res.send([ { timestamp: (new Date()).toISOString() , comment : comment}  ]);
});

export  default { apiRouter };
```

server_event.mjs

```
import express from 'express';
var app = express();

import xyApiRoutes from './with-fire-event.mjs';
import ccApiRoutes from './with-event-handler.mjs';

app.use(xyApiRoutes.apiRouter);
app.use(ccApiRoutes.apiRouter);

let backendPort = process.env.PORT || 8282;
app.listen(backendPort , function () {
  console.log("http://localhost:"+backendPort);
});

/*   node server_events.mjs
http://localhost:8282/api-cc/comment → […  ,"comment":"no comment"}]
http://localhost:8282/api-xy/xx (à appeler 2 ou 3 fois dans autre onglet)
http://localhost:8282/api-cc/comment → […  ,"comment":"new-counter=3"}]
*/
```

Affichages sur console :

received event: {"counter":1}
>>> new-counter now not 0: {"counter":1}
new-counter event was fire with counter=1

received event: {"counter":2}
new-counter event was fire with counter=2

2. File system (node Js)

Le module "file system (fs)" de nodeJs peut être utilisé en version CJS/require ou bien ESM/import .
Au sein des versions récentes de nodeJs (>=10) , le module "file system" peut être utilisé en versions "promises" avec async/await par exemple .
Déjà intégré dans le coeur de nodeJs, le module "fs" ne nécessite pas de téléchargement via npm.

2.1. Utilisation en version traditionnelle (avec callback) :

```
const fs = require('fs');
//import fs from 'fs';


//ajout à la fin (sans saut de ligne par défaut) , si fichier inexistant , un nouveau fichier est créé
fs.appendFile('fichierA.txt', '\ncontenuA',
    function (err) {
      if (err) throw err;
      console.log('fichierA créé ou agrandi !');
   });
```

2.2. Utilisation en version récente avec promises

```
import fs from 'fs/promises';  //dans .mjs ou bien avec "type" : "module" dans package.json

async function ajoutFinFichier(){
  try{
     //ajout à la fin (sans saut de ligne par défaut) , si fichier inexistant , un nouveau fichier est créé
     await fs.appendFile('fichierABis.txt', '\ncontenuA');
     console.log("fichierABis créé ou agrandi")
  }catch(err){
    console.error("err="+err);
  }
}
ajoutFinFichier();
```

2.3. Lecture/écriture de petits fichiers

fs.**appendFile**(pathName,contenuAjoute,[cb_err])	Ajout à la fin ou création
fs.**writeFile**(pathName,nouveauContenu,[cb_err])	Création ou remplacement de tout l'ancien contenu
fs.**readFile**(pathName, 'utf8',[cb_err_data])	Lecture en un seul bloc d'un petit fichier texte
...	

Exemples :

```
await fs.writeFile('fichierB.txt', 'ligne1\nligne2');
```

```
let data = await fs.readFile('fichierB.txt');
console.log("contenu du fichierB = " + data);
```

2.4. ouverture et lecture/écriture de fichiers

```
import fs from 'fs/promises';

async function ouvertureEcritureRelectureFichier(){
  try{
     let file = await fs.open('fichierC.txt', 'w+'); //"w+" : writing and reading
     let buf = Buffer.from('ligne1\nligne2\nligne3');
     let pos = 0,offset = 0, len = buf.length;
     let resWrite = await file.write( buf, offset, len, pos);
     console.log("nb bytes written=" + resWrite.bytesWritten);
     let buf2 = Buffer.alloc(len);
     await file.read( buf2,offset, len, pos);
     console.log("buf2="+buf2.toString());
     await file.close();
  }catch(err){
    console.error("err="+err);
  }
}
ouvertureEcritureRelectureFichier();
```

2.5. Manipulations de fichiers (copy , rename, ...)

fs.**copyFile**(scrPathName,destPathName,[cb_err])	Copie/duplication de fichier
fs.**rename**(oldPathName,newPathName,[cb_err])	Renommer/déplacer un fichier
fs.**unlink**(pathName,[cb_err])	Suppression de fichier
...	

Exemple :

```
import fs from 'fs/promises';

async function manipulationFichier(){
  try{
     await fs.writeFile('fichierD1.txt', 'ligne1\nligne2'); //création de D1
     await fs.copyFile('fichierD1.txt', 'fichierD2.txt'); //copie de D1 vers D2
     await fs.rename('fichierD1.txt', 'fichierD1.old.txt'); //renommer D1 en D1.old
     await fs.mkdir("temp"); //création nouveau répertoire "temp"
     await fs.rename('fichierD2.txt', 'temp/fichierD2.txt'); //deplacer D2 vers temp/D2
     //await fs.unlink("fichierB.txt"); //supression du fichierB.txt
  }catch(err){
    console.error("err="+err);
  }
}
manipulationFichier();
```

3. Streams (node Js)

3.1. Stream asynchrone vs Buffer synchrone

Dans un contexte nodeJs :

- un **Buffer** est un **bloc de donnée** de **taille connue** à l'avance et dont le contenu est accessible de manière **synchrone**.
- un **Stream** est un **flow/flux de donnée** de **taille inconnue** et dont le **contenu** est **accessible par paquets (chunks) au fil du temps** de manière **asynchrone**.

Exemple de stream dans la vie courante : *streaming vidéo* : on peut commencer à visualiser le début d'un film dès que les premiers morceaux ont été téléchargés (pas besoin d'attendre le téléchargement du film entier).

Un Stream comporte généralement un buffer interne pour mémoriser les paquets/chuncks/morceaux qu'il contient dans une zone "mémoire tampon".

Un Stream est dit "**readable**" lorsqu'il produit en interne des chunks que l'on peut lire à l'extérieur.
Un Stream est dit "**writable**" lorsqu'on peut lui envoyer des chunks à traiter.

Un Stream c'est également un émétteur d'événement (EventEmitter) . Par exemple, un Stream «Readable» va émettre un événement "*readable*", pour indiquer à celui qui le consomme, que de la donnée est prête à être consommée.

3.2. Buffer

Un **Buffer**, c'est un **conteneur** dans lequel est stockée de la **donnée au format binaire**. On peut le voir comme un ***tableau de bytes***, c'est-à-dire un tableau de nombres compris **entre 0 et 255**.

```
function bufferExample(){
   const s1 = "vidéo";
   console.log("s1.length="+s1.length); //5

   const buffer = Buffer.from(s1, "utf8");
   console.log("buffer.length="+buffer.length); //6 car é codé en utf8 sur 2 octets

   const bytes = Array.from(buffer.values());
   console.log("bytes=" + bytes); // 118,105,100,195,169,111

   const s2 = Buffer.from(bytes).toString();
   console.log("s2="+s2); // "vidéo"
}
```

3.3. Vue d'ensemble sur les streams de NodeJs

Readable	Source de données à lire (ex : fs.createReadStream)
Writable	Destination à alimenter (ex : fs.createWriteStream)
Duplex	À la fois "Readable" et "Writable" (ex : net.Socket)
Transform	Transformation intermédiaire entre "reader" et "writer" (ex : file-compression , toUppercase() , …)

```
import { Readable , Writable ,Transform , pipeline} from 'stream';
```

3.4. Exemples simples (avec file system)

```
import fs from 'fs';

import fsp from 'fs/promises';


function simpleStream(){
   let writeStream = fs.createWriteStream('fichierF.txt');
   writeStream.write('Mon contenu de flux', 'utf-8');
   writeStream.write('\nMon contenu2 de flux', 'utf-8');
   //NB: finish event will be fired when calling .end()
   writeStream.on('finish', () => {   console.log('fichierF mis à jour !');});
   writeStream.end();
}
simpleStream();

async function copierFichierViaStreams(){
   try{
      await fsp.writeFile('fichierE1.txt', 'ligne1\nligne2'); //création de E1

      //copie de E1 en E2 :
      //fs.createReadStream('fichierE1.txt').pipe(fs.createWriteStream('fichierE2.txt'));
      let fileE1 = await fsp.open('fichierE1.txt', 'r');
      let fileE2 = await fsp.open('fichierE2.txt', 'w');
      fileE1.createReadStream().pipe(fileE2.createWriteStream());
   }catch(err){
     console.error("err="+err);
   }
 }
 copierFichierViaStreams();
```

3.5. Readable Streams de nodeJs

Exemple de stream simple générant des micros chuncks de type compteur 1,2,3,4,5 :

```
class ReadableCounterStream extends Readable {
   constructor(){
      super({ encoding: 'utf8' }); //default highWaterMark = 16kb (a little like back pressure)
      this.data = 0; //counter
   }

   //nb: _read() will not be called directly , but via node.
   _read() {
      this.data += 1;
      if (this.data <= 5) {
         const chunk = this.data.toString();
         //console.log("_read() was called with chunk="+chunk)
         this.push(chunk);
      } else {
         this.push(null); //push null to say "end of stream"
      }
   }
}
```

Type d'évènements sur Readable Streams	Significations
"data" en mode push/flowing	*partie/chunk/paquet accessible (déjà poussé) à traiter/consommer :* readerStream.on('data', function(chunk) { …}) ;
"readable" en mode paused/polling	*partie/chunk/paquet disponible à aller chercher via un appel à .read()*
"end" *et variante .close()*	*Fin du flux*
"error"	*Erreur interne au stream :* readerStream.on('error', function(err) { console.log(err.stack);});

Un "readable stream" peut être utilisé de 2 manières :
- en mode "**paused**" (pull)
- en mode "**flowing**" (push)

Exemple d'utilisation d'un stream en mode "push / flowing mode" :

```
function simpleUse_flowingMode(){
  const myReadableCounterStream = new ReadableCounterStream();

  //NB: l'état initial d'un readable stream est sensé être en mode paused (or null / IDLE)  ???
  myReadableCounterStream.pause(); //demande explicite du mode "paused"
  console.log("after .pause() , isPaused():" + myReadableCounterStream.isPaused());

  //flowing mode is like a websocket push
  myReadableCounterStream.on('data', chunk => {
   console.log("data chunk=" + chunk);
  });
  myReadableCounterStream.resume(); //inverse of pause (flowing mode /push mode)
  console.log("after .resume() , isPaused():" + myReadableCounterStream.isPaused());
}
```

NB : en mode push/flowing , il faut absolument s'être préalablement abonné à l'événement 'data' avant d'appeler .resume() sinon certaines données risques d'être perdues (jamais traitées) .

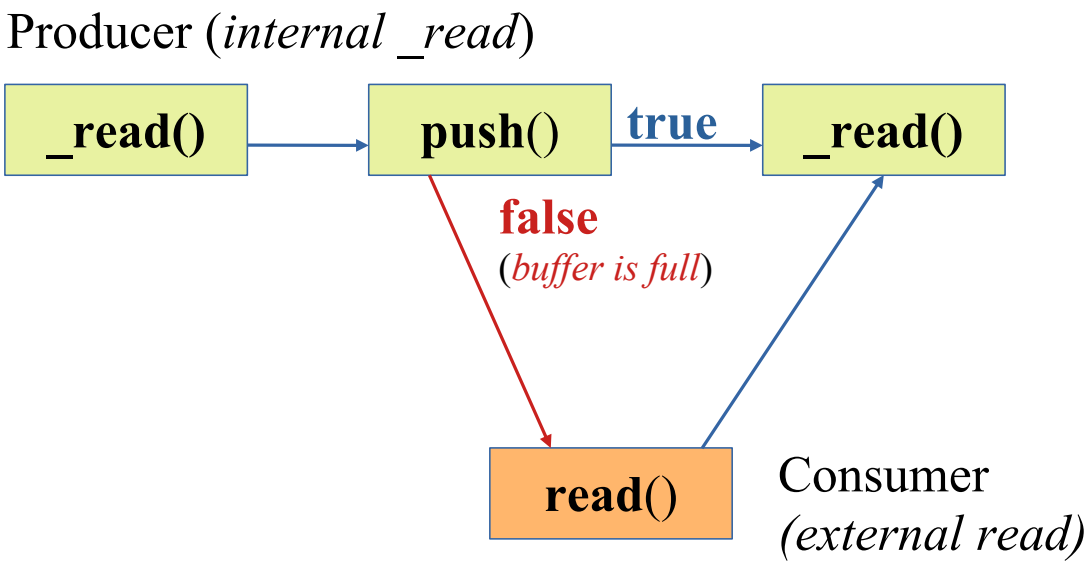
after .pause() , isPaused():true
after .resume() , isPaused():false
data chunk=1
data chunk=2
data chunk=3
data chunk=4
data chunk=5

Exemple d'utilisation d'un stream en mode "pull / paused mode" :

```
function simpleUse_pauseMode(){
   const myReadableCounterStream = new ReadableCounterStream();

   myReadableCounterStream.on('readable', () => {
      let chunk;
      //calling .read() (without _) in paused mode is like a polling rest call
      while ((chunk = myReadableCounterStream.read()) !== null) {
         console.log("readable data chunk=" + chunk);
      }
      });
 }
```

readable data chunk=12
readable data chunk=3
readable data chunk=4
readable data chunk=5

Cycle de production/consommation des "readable" Streams



push() may return ***true*** or ***false***

3.6. Writable Streams de nodeJs

Exemple simple de "Writeable Stream" ecrivant/affichant les chunks à la console :

```
class WritableLoggerStream extends Writable {
   _write(chunk, encoding, next) {
   console.log(`internal buffer length=${this.writableLength} chunk=${chunk}`);
   next(); //indiquer "près pour récupérer le prochain chunk"
   }
}
```

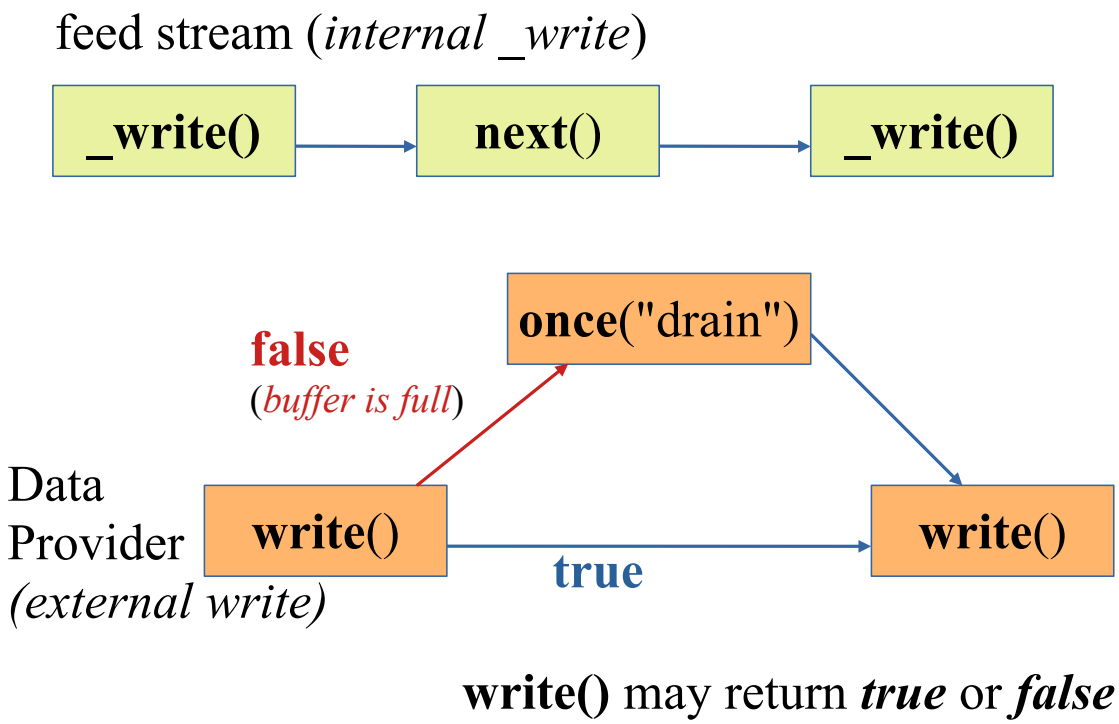
Type d'évènements sur Writable Streams	Significations
"finish"	*Ecritures terminées*
"error"	*Erreur interne au stream :* readerStream.on('error', function(err) { console.log(err.stack);});
"pipe" et "unpipe"	*Relié ou déconnecté avec un "readable stream"*

Exemple simple d'utilisation d'un writable stream :

```
function simpleWritableUse(){
   const myWritableLoggerStream = new WritableLoggerStream();
   let data = 0; //compteur (de 0 à 5)
   let feedStream = () => {
      data += 1;

      if (data < 5) {
         const isNowStillWritable = myWritableLoggerStream.write(data.toString());
                                    //write() without _
         if (isNowStillWritable) {
           setTimeout(feedStream, 50);//next asynchronous sent after 50ms
         }else{
           //'drain' event will be fired by stream when 'writable' in the future
           myWritableLoggerStream.once('drain', feedStream);
         }
      } else {
         myWritableLoggerStream.end(data.toString());//last sent with data or empty load
      }
   }

   feedStream();
}
```

```
internal buffer length=1 chunk=1
internal buffer length=1 chunk=2
internal buffer length=1 chunk=3
internal buffer length=1 chunk=4
internal buffer length=1 chunk=5
```

Cycle d'alimentation des "writable" Streams



3.7. Liaisons de streams via pipe ou pipeline

Avec .pipe() et gestion séparée des erreurs sur readable et writable stream :

```
function simpleReadablePipeWritableUse(){
   const myReadableCounterStream = new ReadableCounterStream();
   myReadableCounterStream.on('error',(err) => {console.error(err.message);});

   const myWritableLoggerStream = new WritableLoggerStream();
   myWritableLoggerStream.on('error',(err) => {console.error(err.message);});

   myReadableCounterStream.pipe(myWritableLoggerStream);
}
```

Avec pipeline() et gestion mutualisée des erreurs :

```
function simpleReadablePipelineWritableUse(){
   const myReadableCounterStream = new ReadableCounterStream();
   const myWritableLoggerStream = new WritableLoggerStream();
   //pipeline() of 'streams' module pour mutualiser le traitement des erreurs
   pipeline(myReadableCounterStream ,
            myWritableLoggerStream ,
         (err) => { if(err) console.error(err.message);}
         ) ;
}
```

3.8. Liaisons avec traitement (Transform)

```
function simplePipeWithTransformUse(){
   const myReadableCounterStream = new ReadableCounterStream();
   const myWritableLoggerStream = new WritableLoggerStream();

   const multByTwoTransform = new Transform({
      transform(chunk, encoding, callback) {
         callback(null, (chunk *  2).toString());
      },
    });

   myReadableCounterStream.pipe(multByTwoTransform).pipe(myWritableLoggerStream);
}
```

internal buffer length=1 chunk=2
internal buffer length=1 chunk=4
internal buffer length=1 chunk=6
internal buffer length=1 chunk=8
internal buffer length=2 chunk=10

//Exemple avec transformation prédéfinie :

```
import fs = from 'fs';
import zlib from 'zlib';
const file = "f1";

fs.createReadStream(file)
  .pipe(zlib.createGzip())
  .pipe(fs.createWriteStream(file + '.gz'));
```

Petite variante :

```
…
import { Transform } from 'stream';


const reportProgress = new Transform({
  transform(chunk, encoding, callback) {
    process.stdout.write('.');
    callback(null, chunk);
  }
});

fs.createReadStream(file)
  .pipe(zlib.createGzip())
  .pipe(reportProgress)
  .pipe(fs.createWriteStream(file + '.zz'))
  .on('finish', () => console.log('Done'));
```

IX - NodeJs: aspects divers et avancés

1. Suppression/remplacement d'attribut javascript

```
var obj = { prenom:"jean", nom:"Bon" , age:25 };
console.log(JSON.stringify(obj));
obj.name=obj.nom;  //ajout de la propriété .name (par copie de la valeur de .nom)
console.log(JSON.stringify(obj));
delete obj.nom; //supression de la propriété .nom
console.log(JSON.stringify(obj));
```

2. Saisie asynchrone en mode texte (stdin)

2.1. Saisies implicitement en boucle

```
var stdin = process.openStdin();  //ou process.stdin

stdin.addListener("data", function(d) {
    // note:  d is an object, and when converted to a string it will
    // end with a linefeed.  so we (rather crudely) account for that
    // with toString() and then trim()
    console.log("you entered: [" +   d.toString().trim() + "]");
  });
```

2.2. Saisie unitaire avec question préalable et callback

```
var stdin = process.stdin;
var stdout = process.stdout;

function ask(question, callback) {
 stdin.resume();
 stdout.write(question + ": ");
 stdin.once('data', function(data) {
  data = data.toString().trim();
    callback(data);
  });
}

//utilisation chaînée avec callbacks imbriquées:

ask("x", function(valX){
                var x=Number(valX);
                        ask("y", function(valY){
                                        var y=Number(valY);
                                        var res=x+y ;
                                        console.log("res = (x+y)=" +res);
```

```
                process.exit();
            });
});
```

3. Fonctionnalités des versions récentes de node

3.1. Prise en charge directe de typescript avec node ≥ 23

s1.ts

```
let x=5;
let y : number;
y=7;
let z=x+y;
console.log(z);
```

Depuis la version 23, node peut maintenant directement lancer des scripts codés en typescript (.ts) sans transpilation préalable (plus absolument besoin de *tsc s1.ts* préalable pour transformer .ts en .js) .

```
node s1.ts
```

Ceci ne fonctionne bien que dans des cas simples.
Si besoin l'option ***--experimental-transform-types*** est là pour certains cas complexes (avec enums ou autres)

3.2. require(esm_module.mjs)

Depuis de nombreuses années, on peut importer un module cjs au sein d'un module ESM via une syntaxe de ce genre :

```
import calcul , { add } from './calcul.mjs'
```

Mais l'inverse était encore récemment impossible.

Depuis les versions 23,24 de nodeJs, il est maintenant possible d'importer via require(…) un module ESM :

calcul1.mjs

```
function mult(a,b){
        return a*b;
}

//ESM exports :
export default {
        name : "calcul1" ,
        mult : mult
}
```

calcul2.mjs

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

main.js (au sein d'un projet avec *"type" : "commonjs"*)

```
//esm import via require():
const { add } = require('./calcul2.mjs');
let x=add(6,8);
console.log("x=6+8="+x); //x=14

//esm import (with .default) via require():
const  calcul1  = require('./calcul1.mjs');
const  calcul1Default= calcul1.default;

console.log("calcul1.default.name="+calcul1Default.name);//calcul1.default.name=calcul1
let y=calcul1Default.mult(6,8);
console.log("y=6*8="+y); //y=48
```

node main.js

3.3. localStorage et sessionStorage du coté node/serveur

L'api localStorage/sessionStorage est disponible dans tous les navigateurs modernes depuis de nombreuses années.

Certaines technologies "frontEnd" telles que Angular (ou React ou …) peuvent éventuellement fonctionner de manière isomorphe (pré-rendu effectué coté serveur puis prise de relai du coté navigateur) .
Etant donné que localStorage et sessionStorage n'était jusqu'à maintenant disponible que du coté navigateur, il fallait jusqu'ici parfois tester si un code javascript est exécuté du coté navigateur/client ou bien node/serveur pour l'utiliser sans risquer un nullPointerException/undefined .

A partir de node 24, les api localStorage et sessionStorage sont directement intégrées à node (de manière expérimentale pour l'instant). On peut donc à long terme espérer une simplification de certains codes de "serveur side rendering" ou autres.

withStorages.js

```
localStorage.setItem("couleur", "rouge");
let c = localStorage.getItem("couleur");
console.log("c="+c); //affiche c=rouge

localStorage.setItem("x", 5);
let x = localStorage.getItem("x");
console.log("x="+x); //affiche x=5
```

```
node --experimental-webstorage --localstorage-file myLocalStorageFile.txt withStorages.js
```

Avec node 24, plus absolument besoin de l'api node-localstorage
car c'est maintenant directement intégré à node
via l'option --experimental-webstorage et --localstorage-file myLocalStorageFile.txt

3.4. Autres nouveautés des versions récentes et expérimentales :

- amélioration des performances
- ...

X - WebSockets , Socket.io

1. WebSockets avec node js

1.1. WebSockets HTML5 (standard)

Les "WebSockets" constituent une adaptation "HTTP" des classiques sockets "tcp/ip" .

C'est une sorte d'annexe/extension vis à vis du protocole HTTP .

Ayant leurs propres préfixes (schemes : ws : , wss :) , les "WebSockets" peuvent également être vues comme un nouveau protocole (avec même la notion de sous protocole possible tel que STOMP)

En tant que "sockets" , une "**WebSockets**" est un **canal de communication bi-directionnel établi durablement (tant que pas fermé)** entre un client et un serveur par exemple .

Ce canal bi-directionnel peut servir à spontanément envoyer (dans les 2 sens) des messages dans format quelconque (texte , json, ... ou binaire).

1.1.a. Fonctionnalités des WebSockets

Les "webSockets" sont surtout utilisés dans une logique de "**push**" ou "**subscribe/publish**" .

Une fois une connexion établie , le serveur peut spontanément envoyer de nouvelles valeurs (qui viennent de changer) vers le coté client/navigateur sans que celui-ci soit obligé d'effectuer une requête préalable d'actualisation .

En règle générale, de nombreux clients sont simultanément connectés à un même serveur .

Le serveur peut alors via une simple boucle diffuser une information vers tous les clients connectés via une websocket active .

Application classique :

- discussion en tant réel : "chat" , messagerie instantanée
- actualisation automatique de graphique (ex : SVG, canvas, ...) dès qu'une valeur change
- tableau (board) partagé en équipe
- toute autre communication en mode push (avec ou sans RxJs / mode réactif) .

1.1.b. Principe de fonctionnement

Une connexion "webSocket" s'effectue en partant d'une connexion "http" existante puis en "upgradant" celle-ci durant une phase d'échange d'informations appelée "**handshake** " .

Le client envoi une requête HTTP de ce type

```
GET /chat HTTP/1.1
Host: server.example.com
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Key: dGhlIHNhbXBsZSBub25jZQ==
Sec-WebSocket-Origin: http://example.com
```

```
Sec-WebSocket-Protocol: chat, superchat
Sec-WebSocket-Version: 6
```

Le serveur doit s'il accepte l'upgrade , renvoyer une réponse de ce type :

```
HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: s3pPLMBiTxaQ9kYGzzhZRbK+xOo=
Sec-WebSocket-Protocol: chat
```

Au niveau du paramètre du constructeur WebSocket() de l'api html5/javascript, l'url d'une websocket à préciser pour établir une connexion ne commence par par http:// ni par https:// mais par **ws://** ou **wss://** . Cependant , comme la connexion ws:// est un upgrade d'une connexion HTTP , le numéro de port utilisé par les "websockets" reste le standard **80** et donc pas de soucis en général pour que les requêtes/réponses puis passer à travers proxy ou firewall .

Une fois la connexion établie , Chaque protagoniste (client et serveur) voit l'autre coté du canal de communication comme un "**endpoint**" vers lequel on peut spontanément envoyer des messages via une méthode .**send**() .

Du coté réception , les méthodes "callback" suivantes seront automatiquement appelées :

- **`onopen`** : ouverture d'une WebSocket
- **`onmessage`** : réception d'un message
- **`onerror`** : erreur(s) survenue(s)
- **`onclose`** : fermeture de WebSocket (de l'autre coté)

1.1.c. Exemple de "chat"(webSocket) : code client "html5+javascript"

```
<?xml version="1.0" encoding="UTF-8"?>
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">
<head>    <title>Apache Tomcat WebSocket Examples: Chat</title>
   <style type="text/css">...</style>
   <script type="application/javascript"><![CDATA[
      "use strict";
      var Chat = {};  //custom literal js object

      Chat.socket = null;
      Chat.connect = (function(host) {
         if ('WebSocket' in window) {
            Chat.socket = new WebSocket(host);
         } else if ('MozWebSocket' in window) {
            Chat.socket = new MozWebSocket(host);
         } else {
            Console.log('Error: WebSocket is not supported by this browser.');
            return;
         }

         Chat.socket.onopen = function () {
            Console.log('Info: WebSocket connection opened.');
            document.getElementById('chat').onkeydown = function(event) {
               if (event.keyCode == 13) {
```

```
                Chat.sendMessage();
            }
        };
    };

    Chat.socket.onclose = function () {
        document.getElementById('chat').onkeydown = null;
        Console.log('Info: WebSocket closed.');
    };

    Chat.socket.onmessage = function (message) {
        Console.log(message.data);
    };
});

Chat.initialize = function() {
    if (window.location.protocol == 'http:') {
        Chat.connect('ws://' + window.location.host + '/examples/websocket/chat');
    } else {
        Chat.connect('wss://' + window.location.host + '/examples/websocket/chat');
    }
};

Chat.sendMessage = (function() {
    var message = document.getElementById('chat').value;
    if (message != '') {
        Chat.socket.send(message);
        document.getElementById('chat').value = '';
    }
});

var Console = {};  //custom literal js object

Console.log = (function(message) {
    var console = document.getElementById('console');
    var p = document.createElement('p');
    p.style.wordWrap = 'break-word';
    p.innerHTML = message;
    console.appendChild(p);
    while (console.childNodes.length > 25) {
        console.removeChild(console.firstChild);
    }
    console.scrollTop = console.scrollHeight;
});

Chat.initialize();
]]></script>
</head>
<body>
  <p>
    <input type="text" placeholder="type and press enter to chat" id="chat" />
  </p>
  <div id="console-container">
    <div id="console"/>
  </div>
</div> </body> </html>
```

1.2. WebSockets nodeJs via ws

npm install --save express

npm install --save ws

```
"dependencies": {
   "express": "^4.18.2",
   "ws": "^8.9.0"
  }
```

my_websockets.mjs

```
import { WebSocketServer }  from 'ws';
//détails sur https://www.npmjs.com/package/ws

//if { port : 6000 , path : "/ws" } , two differents ports for express and websocket : it's work
//if { noServer: true , path : "/ws" } need to call expressServer.on('upgrade', (request, socket, head)
// => { ..} on express server AND just one port
//creating ws_server that will be attached to express server --> ws://localhost:5000/ws
const ws_server = new WebSocketServer({ noServer : true ,  path : "/ws" });

ws_server.on('connection', (ws) => {
        console.log('New client connected!');

        ws.on('message', (data) => {
                console.log("received message=" + data);
                //..
        });

         ws.on('close', () => console.log('Client has disconnected!'));
});

export default { ws_server };
```

app.mjs

```
import express from 'express';
import my_websockets from './my_websockets.mjs';

import { dirname } from 'path';
import { fileURLToPath } from 'url';
const __dirname = dirname(fileURLToPath(import.meta.url));

const app = express(); //express app

//les routes en /html/... seront gérées par express par
//de simples renvois des fichiers statiques du répertoire "./html"
app.use('/html', express.static(__dirname+"/html"));
app.get('/', function(req , res ) {
  res.redirect('/html/index.html');
});
```

```
//start express server:
const expressServer = app.listen(5000,
  () => console.log('http://localhost:5000')
);

const ws_server = my_websockets.ws_server;

//upgrade express server to "http + ws" server:
//détails sur https://www.npmjs.com/package/ws
expressServer.on('upgrade', (request, socket, head) => {
  ws_server.handleUpgrade(request, socket, head, socket => {
    ws_server.emit('connection', socket, request);
  });
});
```

Exemple1 (affichage timestamp serveur toutes les 5s)

dans my_websockets.mjs

```
…
function broadcast_json_message_to_all(objData){
  broadcast_message_to_all(JSON.stringify(objData));
}

function broadcast_message_to_all(message){
  ws_server.clients.forEach((client) => {
    client.send(message);
  });
}

setInterval(() => {
  broadcast_json_message_to_all({ type: "timestamp" , info : new Date().toTimeString() });
}, 5000);

...
```

html/test-ws.html

```
<html>
<head>
  <script>
    let webSocket = new WebSocket('ws://localhost:5000/ws');
    let el;
    webSocket.onopen = () => { console.log('Info: WebSocket connection opened.'); };

    webSocket.onmessage = (event) => {
      el = document.getElementById('time');
    let objData = JSON.parse(event.data);
    if(objData.type == "timestamp" )
        el.innerHTML = '(info): Current time on server is: ' + objData.info;
```

```
  };
 </script>
</head>

<body>
 <h1>test-ws.html</h1>
 <p id="time"></p>
</body>

</html>
```

test-ws.html

(info): Current time on server is: 16:47:02 GMT+0200 (heure d'été d'Europe centrale)

avec l'heure qui évolue automatiquement toutes les 5s .

Exemple2 (chat)

dans my_websockets.mjs

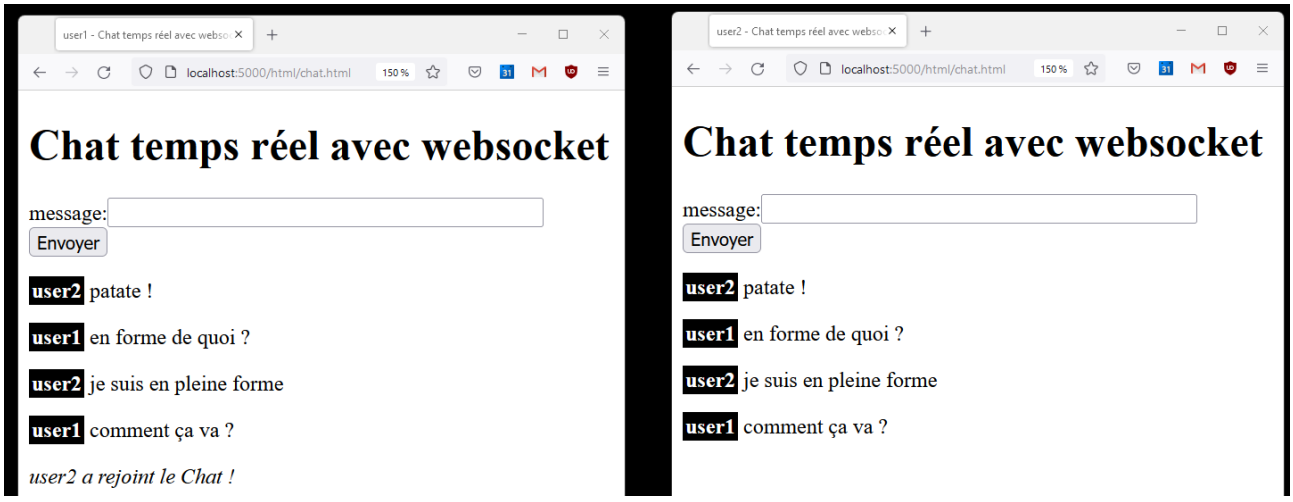
```
...
ws_server.on('connection', (ws) => {
 //console.log('New client connected!');

 ws.on('message', (data) => {
  //console.log("received message=" + data);
  let objData = JSON.parse(data);
  switch(objData.type){
   case 'chat_message' :
     // Dès qu'on reçoit un message, on récupère le pseudo de son auteur
    //et on le transmet aux autres personnes
    broadcast_json_message_to_others(ws,{ type: "chat_message" ,
                                          pseudo : ws.pseudo ,
                                          message : objData.message });
      break;
   case  'nouveau_client' :
    //Dès qu'on nous donne un pseudo,
    //on le stocke en variable de session et on informe les autres personnes
    ws.pseudo = objData.pseudo;
    broadcast_json_message_to_others(ws,{ type: "nouveau_client" ,
                                          pseudo : objData.pseudo });
     break;
   }
 });

 ws.on('close', () => console.log('Client has disconnected!'));
```

```
});

function broadcast_json_message_to_others(currWs,objData){
  broadcast_message_to_others(currWs,JSON.stringify(objData));
}

function broadcast_message_to_others(currWs,message){
  ws_server.clients.forEach((client) => {
    if(client != currWs)
      client.send(message);
  });
}

...
```

html/chat.html

```
<html>
   <head>
      <meta charset="utf-8" />        <title>Chat temps réel avec websockets</title>
      <style>
         #zone_chat strong {  color: white;      background-color: black ;        padding: 2px;      }
      </style>
   </head>
   <body>
      <h1>Chat temps réel avec websocket</h1>
      message:<input type="text" name="message" id="message"  size="50" autofocus />
      <input type="button" id="envoi_message" value="Envoyer" />
      <div id="zone_chat">
      </div>

      <script>
         var zoneChat = document.querySelector('#zone_chat');
         var zoneMessage = document.querySelector('#message');

         let socket = new WebSocket('ws://localhost:5000/ws');

         // On demande le pseudo, on l'envoie au serveur et on l'affiche dans le titre
         var pseudo = prompt('Quel est votre pseudo ?');
         socket.send(JSON.stringify({type:'nouveau_client', pseudo : pseudo}));
         document.title = pseudo + ' - ' + document.title;

         socket.onmessage= function(event) {
            let objData = JSON.parse(event.data);
            switch(objData.type){
               case 'chat_message' :
                  // Quand on reçoit un message de chat, on l'insère dans la page
                  insereMessage(objData.pseudo, objData.message);
                  break;
               case  'nouveau_client' :
                  // Quand un nouveau client se connecte, on affiche l'information
                  zoneChat.innerHTML='<p><em>' + objData.pseudo
```

```
                    + ' a rejoint le Chat !</em></p>'+zoneChat.innerHTML;
                break;
            }
        };

        // Lorsqu'on envoie le formulaire, on transmet le message et on l'affiche sur la page
        document.querySelector('#envoi_message').addEventListener('click',function () {
            var message = zoneMessage.value;

            // on transmet le message aux autres :
            socket.send(JSON.stringify({type : 'chat_message', message: message}));

            insereMessage(pseudo, message); // Affiche le message aussi sur notre page
            zoneMessage.value=''; zoneMessage.focus(); // Vide la zone de Chat et remet le focus dessus
            return false; // Permet de bloquer l'envoi "classique" du formulaire
        });

        // Ajoute un message dans la page
        function insereMessage(pseudo, message) {
            zoneChat.innerHTML='<p><strong>' + pseudo + '</strong> ' + message
                                + '</p>'+zoneChat.innerHTML;
        }
    </script>
  </body>
</html>
```



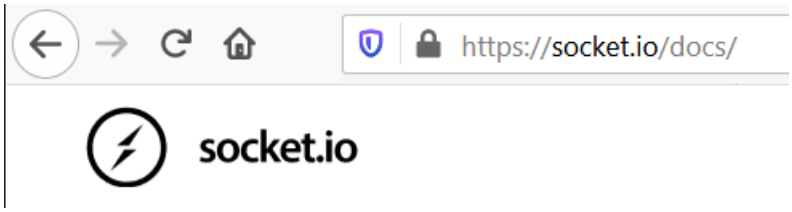
1.3. WebSockets nodeJs via socket.io

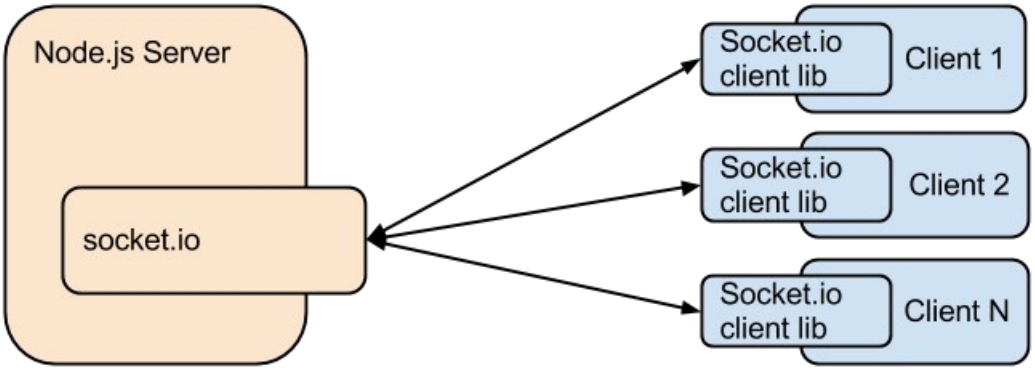
Socket.io est une **api javascript** qui encapsule et améliore la prise en charge des "**websockets**" .

Rappel : les **websockets** sont une technologie permettant d'établir des ***communications bidirectionnelles*** via un **canal construit au dessus du protocole HTTP** et cela permet au coté serveur d'envoyer spontanément des informations (ou événements) au client (fonctionnant dans un navigateur). C'est une des technologies de "**push**" .

Valeurs ajoutés par la bibliothèque javascript "**socket.io**" :

websocket (utilisé seul)	**socket.io**
protocole (lui même basé sur tcp et http) maintenant géré par la plupart des navigateurs et pouvant être manipulé en code javascript de bas niveau	**librairie javascript** complémentaire (à télécharger et utiliser)
fourni un canal de communication full-duplex de bas niveau	**fourni un canal abstrait de communication full-duplex (de plus haut niveau) basé sur des événements .**
proxy http et load-balancer ne sont pas gérés par les websockets ===> gros problème / limitation	**les communications peuvent êtres établies même en présence de proxy http et de load-balancer** (en mettant en oeuvre des mécanismes supplémentaires pour compenser les limitations des "websockets" généralement utilisées en interne)
ne gère pas le broadcasting	gère (si besoin) le broadcasting
pas d'option pour le "fallback"	comporte des options pour le "fallback"





documentation sur site officiel de Socket.io (présentation, concepts):

- https://socket.io/docs/

bibliothèque socket.io:

- https://www.npmjs.com/package/socket.io-client
- https://www.npmjs.com/package/@types/socket.io-client

---

npm install -s express

npm install -s socket.io

npm install -s ent

```
"dependencies": {
      "ent": "^2.2.0",
      "express": "^4.18.2",
      "socket.io": "^4.5.2"
   },
```

app.mjs

```
import express  from 'express';
import http from 'http';

import my_socket_io from './my_socket_io.mjs';

import { dirname } from 'path';
import { fileURLToPath } from 'url';
const __dirname = dirname(fileURLToPath(import.meta.url));

const  app  = express();

//les routes en /html/... seront gérées par express
//par de simples renvois des fichiers statiques du répertoire "/html"
app.use('/html', express.static(__dirname+"/html"));
app.get('/', function(req , res  ) {    res.redirect('/html/index.html');  });
```

```
const server = http.createServer(app);

const io = my_socket_io.initSocketIO(server);

server.listen(8383,function () {
   console.log("http://localhost:8383");
 });
```

my_socket_io.mjs

```
import ent from 'ent';// Permet de bloquer les caractères HTML
import  { Server } from 'socket.io';
function initSocketIO(expressServer){
   let io = new Server(expressServer);
   my_io_settings(io);
   return io;
}
function my_io_settings(io){
   //events: connection, message , disconnect and custom_event like nouveau_client
   io.on('connection', function (socket) {
      // Dès qu'on nous donne un pseudo,
      //on le stocke en variable de session et on informe les autres personnes
      socket.on('nouveau_client', function(pseudo) {
         pseudo = ent.encode(pseudo);
         socket.pseudo = pseudo;
         socket.broadcast.emit('nouveau_client', pseudo);
      });

      // Dès qu'on reçoit un message, on récupère le pseudo de son auteur
      //et on le transmet aux autres personnes
      socket.on('message', function (message) {
         message = ent.encode(message);
         socket.broadcast.emit('message', {pseudo: socket.pseudo, message: message});
      });
});
}
export default { initSocketIO };
```

html/index.html

```
<html>
  <head>
    <meta charset="utf-8" />       <title>Chat temps réel avec socket.io</title>
    <style>
      #zone_chat strong {  color: white;      background-color: black ;       padding: 2px;      }
    </style>
  </head>
  <body>
    <h1>Chat temps réel avec socket.io</h1>
    message:<input type="text" name="message" id="message"  size="50" autofocus />
    <input type="button" id="envoi_message" value="Envoyer" />
    <div id="zone_chat">
    </div>

    <script src="lib/socket.io.js"></script>
     <script>
      var zoneChat = document.querySelector('#zone_chat');
      var zoneMessage = document.querySelector('#message');

      // Connexion à socket.io :
      var socket = io.connect('http://localhost:8383');

      // On demande le pseudo, on l'envoie au serveur et on l'affiche dans le titre
      var pseudo = prompt('Quel est votre pseudo ?');
      socket.emit('nouveau_client', pseudo);
      document.title = pseudo + ' - ' + document.title;

      // Quand on reçoit un message, on l'insère dans la page
      socket.on('message', function(data) {
        insereMessage(data.pseudo, data.message)
      })

      // Quand un nouveau client se connecte, on affiche l'information
      socket.on('nouveau_client', function(pseudo) {
        zoneChat.innerHTML='<p><em>' + pseudo + ' a rejoint le Chat !</em></p>'
                           +zoneChat.innerHTML;
      })

      // Lorsqu'on envoie le formulaire, on transmet le message et on l'affiche sur la page
      document.querySelector('#envoi_message').addEventListener('click',function () {
        var message = zoneMessage.value;
        socket.emit( 'message', message);  // on transmet le message aux autres :

        insereMessage(pseudo, message); // Affiche le message aussi sur notre page
        zoneMessage.value=''; zoneMessage.focus(); // Vide la zone de Chat et remet le focus dessus
        return false; // Permet de bloquer l'envoi "classique" du formulaire
      });
```

```
        // Ajoute un message dans la page
        function insereMessage(pseudo, message) {
            zoneChat.innerHTML='<p><strong>' + pseudo + '</strong> ' + message
                                    + '</p>'+zoneChat.innerHTML;
        }
    </script>
  </body>
</html>
```

NB : ***le coté frontend à besoin de l'extension "coté cliente"*** ***lib/socket.io.js***

à télécharger depuis le site officiel de socket.io

même comportement que la version 'ws' .

ANNEXES

XI - Annexe – Tests (mocha , chai , ...)

1. Tests avec Mocha + Chai (env. nodeJs)

Mocha (+Chai) est la technologie de test préconisée pour nodeJs .
La technologie "mocha" ressemble beaucoup à jasmine . Elle doit être complétée par "chai" pour les assertions/vérifications (expect) .

Il est ainsi assez facile de tester unitairement un composant d'une application nodeJs.
En utilisant en plus le package "request" de façon à déclencher des requêtes HTTP, on peut également invoquer et tester un WS-REST construit avec node+express .

1.1. Structure et commandes

package.json
test/xy-spec.js
app/xy.js

Elément à ajouter/paramétrer dans package.json

```
"scripts": {
   "test": "./node_modules/.bin/mocha --reporter spec"
 },
```

//for nodemon xxx.js (watch mode) in place of node xxx.js
npm install -g nodemon

//for mocha test runner (in node):
npm install mocha --save-dev

//for chai expect/assert:
npm install chai --save-dev

//for rest ws test:
npm install request --save

//for rest app:
npm install express --save

Lancement des tests :

npm run test
ou bien
npm test

NB : Les exemples de code ci-après sont issus du tutorial
"*Getting Started with Node.js and Mocha – Semaphore*" accessible par recherche internet .

1.2. Test unitaire simple

Exemple de composant à tester :

app/converter.js

```
exports.rgbToHex = function(red, green, blue) {
  var redHex   = red.toString(16);
  var greenHex = green.toString(16);
  var blueHex  = blue.toString(16);
  return pad(redHex) + pad(greenHex) + pad(blueHex);
};

function pad(hex) {   return (hex.length === 1 ? "0" + hex : hex);
}

exports.hexToRgb = function(hex) {
  var red   = parseInt(hex.substring(0, 2), 16);
  var green = parseInt(hex.substring(2, 4), 16);
  var blue  = parseInt(hex.substring(4, 6), 16);
  return [red, green, blue];
};
```

exemple de test:

test/converter-spec.js

```
var expect    = require("chai").expect;
var converter = require("../app/converter");

describe("Color Code Converter", function() {
  describe("RGB to Hex conversion", function() {
    it("converts the basic colors", function() {
      var redHex   = converter.rgbToHex(255, 0, 0);
      var greenHex = converter.rgbToHex(0, 255, 0);
      var blueHex  = converter.rgbToHex(0, 0, 255);
      expect(redHex).to.equal("ff0000");
      expect(greenHex).to.equal("00ff00");
      expect(blueHex).to.equal("0000ff");
    });
  });

  describe("Hex to RGB conversion", function() {
    it("converts the basic colors", function() {
      var red   = converter.hexToRgb("ff0000");
      var green = converter.hexToRgb("00ff00");
      var blue  = converter.hexToRgb("0000ff");
      expect(red).to.deep.equal([255, 0, 0]);
      expect(green).to.deep.equal([0, 255, 0]);
      expect(blue).to.deep.equal([0, 0, 255]);
    });
  });
```

```
});
```

1.3. variantes syntaxiques pour chai

- should() n'est qu'une variante de expect .
- assert se rapproche du style des assertions java/JUnit .

via assert.

```
const { assert } = require('chai')

assert(val)
assert.fail(actual, expected)
assert.ok(val)                          // is truthy
assert.equal(actual, expected)          // compare with ==
assert.strictEqual(actual, expected)    // compare with ===
assert.deepEqual(actual, expected)      // deep equal check

assert.isTrue(val)
assert.isFalse(val)

assert.isNull(val)
assert.isNotNull(val)
assert.isUndefined(val)
assert.isDefined(val)
assert.isFunction(val)
assert.isObject(val)
assert.isArray(val)
assert.isString(val)
assert.isNumber(val)
assert.isBoolean(val)

assert.typeOf(/tea/, 'regexp') // Object.prototype.toString()
assert.instanceOf(chai, Tea)
assert.include([ a,b,c ], a)
assert.match(val, /regexp/)
assert.property(obj, 'tea') // 'tea' in object
assert.deepProperty(obj, 'tea.green')
assert.propertyVal(person, 'name', 'John')
assert.deepPropertyVal(post, 'author.name', 'John')

assert.lengthOf(object, 3)
assert.throws(function() { ... })
assert.throws(function() { ... }, /reference error/)
```

```
assert.doesNotThrow

assert.operator(1, '<', 2)
assert.closeTo(actual, expected)
```

via expect(...).to... (BDD syntax : Behavior Driven Development) :

```
const { expect } = require('chai')

expect(object)
  .to.equal(expected)
  .to.eql(expected)        // deep equality
  .to.deep.equal(expected) // same as .eql
  .to.be.a('string')
  .to.include(val)

  .be.ok(val)
  .be.true
  .be.false
  .to.exist

  .to.be.null
  .to.be.undefined
  .to.be.empty
  .to.be.arguments
  .to.be.function
  .to.be.instanceOf

  .to.be.gt(5)  // aka: .above .greaterThan
  .to.be.gte(5) // aka: .at.least
  .to.be.lt(5)  // aka: .below

  .to.respondTo('bar')
  .to.satisfy((n) => n > 0)

  .to.have.members([2, 3, 4])
  .to.have.keys(['foo'])
  .to.have.key('foo')
  .to.have.lengthOf(3)

expect(() => { ··· })
  .to.throw(/not a function/)
```

1.4. Test de service interne en typescript (avec Promise es2015)

```
import chai from 'chai';
import { DeviseDataService } from "../dao/devisedataService";
import { MemoryMapDeviseService } from "../dao/memoryMapDeviseService";
import { Devise } from '../model/devise';
let expect = chai.expect;

var deviseDataService : DeviseDataService =new MemoryMapDeviseService();

describe("internal deviseService", function() {

   it("euro for code EUR", function(done) {
     deviseDataService.findById("EUR")
     .then((deviseEur)=> {
            expect(deviseEur.nom).equals("euro");
            console.log("**" +JSON.stringify(deviseEur));
            done();//pour indiquer a mocha que le test unitaire est fini
            })
     .catch((err)=>console.log("erreur:"+err));
   });

   it("saveOrUpdate_et_getByIdEnchaine", function(done) {
     let nouvelleDev :Devise = { code : "Da1" , nom : "devise a1" , change : 123};
     //let nouvelleDev :Devise = { code : "Da1Wrong" , nom : "devise a1" , change : 123};
     deviseDataService.saveOrUpdate(nouvelleDev)
     .then((dEnregistree)=>{
       //...
       return deviseDataService.findById("Da1");
     })
     .then((deviseRelue)=>{
       expect(deviseRelue.nom).equals("devise a1");
       done();
     })
     .catch((err)=>{ console.log("err:" + err);
                done(err);//mieux que expect.fail("...")
                 })
   });
  });
});
```

1.5. Test de service interne en typescript (avec async/await)

```
import chai from 'chai';
import { DeviseDataService } from "../dao/devisedataService";
import { MemoryMapDeviseService } from "../dao/memoryMapDeviseService";
import { Devise } from '../model/devise';
let expect = chai.expect;

var deviseDataService : DeviseDataService =new MemoryMapDeviseService();

describe("internal deviseService", function() {

  describe("getAllDevises", function() {
    it("returning at least 4 devises", async function() {
      let devises: Devise[] = await deviseDataService.findAll();
      expect(devises.length).to.gte(4); //greater or equals
    });
  });

  describe("getDeviseByCode", function() {
   it("euro for code EUR", async function() {
      let deviseEur: Devise = await deviseDataService.findById("EUR");
      //console.log(JSON.stringify(deviseEur));
      expect(deviseEur.nom).equals("euro");
    });

   it("saveOrUpdate_et_getByIdEnchaine", async function() {
    try{
      let nouvelleDev :Devise = { code : "Da1" , nom : "devise a1" , change : 123};
      //let nouvelleDev :Devise = { code : "Da1Wrong" , nom : "devise a1" , change : 123};
      let dEnregistree = await deviseDataService.saveOrUpdate(nouvelleDev);
      let deviseRelue = await deviseDataService.findById("Da1");
      expect(deviseRelue.nom).equals("devise a1");
    }
    catch(err){
      console.log("err:" + err);
       throw err;
      }
  });

  });

});
```

--> plus d'appel à done() mais préfixe **async** pour la fonction codant le test unitaire

--> plus d'appel à done(err) mais **throw** err dans bloc try/catch au sein d'une fonction async

1.6. Eventuels pré et post traitements (before, after)

```
describe("internal deviseService", function() {

 before(function(done) {
  // runs before all tests :
  //insertion d'un jeu de données:
  sequelize.sync({logging: console.log})
                              .then(
                                    ()=>{
          console.log("sequelize is initialized");
          deviseDataService.saveOrUpdate(new DeviseObject("EUR" , "euro" , 1))
          .then(()=>deviseDataService.saveOrUpdate(new DeviseObject("USD" , "dollar" , 1.1)))
          .then(()=>deviseDataService.saveOrUpdate(new DeviseObject("GBP" , "livre" , 0.9)))
          .then(()=>deviseDataService.saveOrUpdate(new DeviseObject("JPY" , "yen" , 132)))
          .then(()=>{done()});
        }
                              ).catch( (err:any) => { console.log('An error occurred :', err);  });

 });

 describe("getAllDevises", function() {
  it("returning at least 4 devises", async function() {
    let devises: Devise[] = await deviseDataService.findAll();
    expect(devises.length).to.gte(4); //greater or equals
  });
 });

 describe("getDeviseByCode", function() {
  it("euro for code EUR", async function() {
    let deviseEur: Devise = await deviseDataService.findById("EUR");
    //console.log(JSON.stringify(deviseEur));
    expect(deviseEur.nom).equals("euro");
  });
 });
});
```

before()	exécuté une seule fois avant tous les tests de même niveau
after()	exécuté une seule fois après tous les tests de même niveau
beforeEach()	exécuté (plusieurs fois) avant chaque test de même niveau
afterEach()	exécuté (plusieurs fois) après chaque test de même niveau

1.7. Test de web service REST via "request"

NB :

- **request** est une api (module) de nodeJs qui permet d'**effectuer des appels HTTP** .
- **request** peut être utiliser au sein de test mais également au sein d'une application appelant en interne certains services vers une autre application (délégation) .
- **un test (d'intégration ou "end-to-end") basé sur request ne peut fonctionner que si le serveur à tester a été préalablement démarré** .

Exemple de service à tester :

app/server.js

```
var express = require("express");
var app = express();
var converter = require("./converter");

app.get("/rgbToHex", function(req, res) {
  var red   = parseInt(req.query.red, 10);
  var green = parseInt(req.query.green, 10);
  var blue  = parseInt(req.query.blue, 10);
  var hex = converter.rgbToHex(red, green, blue);
  res.send(hex);
});

app.listen(3000);
```

Exemple de test :

test/server-spec.js

```
var expect  = require("chai").expect;
var request = require("request");

describe("Color Code Converter API", function() {
  describe("RGB to Hex conversion", function() {
    var url = "http://localhost:3000/rgbToHex?red=255&green=255&blue=255";

    it("returns status 200", function(done) {
      request(url, function(error, response, body) {
        expect(response.statusCode).to.equal(200);
        done();//pour marquer la fin du test (réponse traitée apres appel asynchrone)
      });
    });

    it("returns the color in hex", function(done) {
      request(url, function(error, response, body) {
        expect(body).to.equal("ffffff");
        done();
      });
    });
  });
});
```

1.8. Test de web service rest via chai-http (ici en typescript)

chai-http est une extension de chai qui permet de :

- effectuer des assertions/vérifications de niveau "communication http"
- lancer si besoin le serveur nodeJs à tester

dans package.json :

```
...
"devDependencies": {
"@types/chai": "^4.1.7",
"@types/chai-http": "^4.2.0",
"@types/express": "^4.16.1",
"@types/mocha": "^5.2.7",
"chai": "^4.2.0",
"chai-http": "^4.3.0",
"mocha": "^6.1.4"
}
```

server.ts

```
import express from 'express';
export const app :express.Application = express();
import { apiRouter } from './api/apiRoutes';
import { initSequelize } from './model/global-db-model'
...
//ROUTES ORDINAIRES (apres PRE traitements , avant POST traitements)
app.use(apiRouter); //delegate REST API routes to apiRouter
...
export const server = app.listen(8282 , function () {
        console.log("http://localhost:8282");
        initSequelize();
});
```

test/xyz.spec.ts

```
import chai from 'chai';
import chaiHttp from 'chai-http';
import { app , server } from '../server';
import { Devise } from '../model/devise';
let expect = chai.expect;

// Configure chai :
chai.use(chaiHttp);
//chai.should();

describe("devise api", function() {

  before(function(done) {
    // runs before all tests :
    //insertion d'un jeu de données via http call:
    chai.request(app)
              .post('/devise')
              .send({code:"EUR" , nom : "euro" , change : 1 })
              .end((err, res) => { done(); });
  });

  after(function() {
    // runs after all tests : close server
    server.close();
  });

  describe("getDeviseByCode", function() {

    it("returns status 200 and a devise object with good name", function(done) {
      chai.request(app)
              .get('/devise/EUR')
              .end((err, res) => {
                 //res.should.have.status(200);
                 chai.expect(res).status(200);
                 let obj = res.body;
                 //obj.should.be.a('object');
                 chai.expect(obj).a('object');
                 let devise = <Devise> obj;
                 //console.log(JSON.stringify(devise));
                 chai.expect(devise.nom).equals("euro");
                 done();
               });
    });
  });
});
```

XII - Annexe – ORM Sequelize

1. ORM Sequelize (node)

"**Sequelize**" est une des technologies "ORM" disponibles dans l'écosystème nodeJs .
"Sequelize" est en 2018,2019 la **technologie "ORM"** la plus utilisée dans le monde "**nodeJs**"

"ORM" signifie "**O**bject **R**elational **M**apping" .
Dans le contexte nodeJs/Sequelize , des objets "javascript" pourront être mis en correspondance avec des enregistrements d'une des bases de données relationnelles utilisables :
- **mysql** ou **mariadb** *(via mysql2 ou mariadb)*
- **postgres** *(via pg pg-hstore)*
- **sqlLite** *(via sqlite3)*
- **sqlserver** *(de Miscrosoft) via tedious*

L'**api Sequelize** est ***asynchrone*** et s'appuie sur les "***Promises***" de es2015 et peut être invoquée via async/await de es2017 . L'utilisation de "Typescript" est possible et facultative .

Principe de fonctionnement de "Sequelize" :

1. **Paramétrage d'une connexion** à une base de données relationnelle (avec ***dialect=mysql*** par exemple)
2. **Paramétrage d'un modèle objet de persistance** (pendant "Sequelize" d'un "schéma relationnel") : ***Liste de classes d'objets persistants*** avec ***structures précises*** et éventuelles associations (1-n , n-n, ...)
3. **Utilisation des classes d'objets persistants définies au niveau du modèle** sequelize pour déclencher les opérations de persistance (CRUD) avec un code "orienté objet" et des ***requêtes SQL générées et déclenchées automatiquement*** .

Ces 3 phases peuvent être réparties au sein de fichiers/modules complémentaires :
- ***db-config.js*** et ***database.cfg.json*** *(config connexion à une base de données en mode "sequelize")*
- ***db-model.js*** *(définitions des classes persistantes (proche "tables") du modèle "sequelize")*
- ***my-sequelize-app.js*** *(utilisation des classes définies dans le modèle)*

1.1. Configuration d'une connexion à une base de données

db-config.js *(version simple avec URL de connexion)*

```
var Sequelize = require('sequelize');
const sequelize = new Sequelize('postgres://user:pass@example.com:5432/dbname');
//on exporte pour utiliser notre connexion depuis les autre fichiers :
var exports = module.exports = {};  exports.sequelize = sequelize;
```

version plus sophistiquée avec fichier de paramétrage :

database.cfg.json

```
{ "dev": {     "dialect": "mysql",  "host": "localhost", "port": 3306,
                "database": "minibank_db_node2",
                "user": "root",         "password": "root" },
  "prod": {     "dialect": "",   "user": "",      "database": " ",          "password": ""          }
}
```

db-config.js *(version classique)*

```
var Sequelize = require('sequelize');

var env="dev" ; //or "prod"
var confDb = require('./database.cfg.json')[env];
var password = confDb.password ? confDb.password : null;

// initialize database connection :
//sequelize = new Sequelize('database', 'username', 'password',
//                          {host: 'localhost', dialect: 'mysql',logging: false,});
var sequelize = new Sequelize(confDb.database, confDb.user, password,
                          { dialect: confDb.dialect,
                            port : confDb.port,
                          logging: false, // false or console.log,// pour voir les logs de sequelize
                          define: {
                                  timestamps: false
                                  }
                          }
                  );

sequelize.authenticate()
  .then(function() {
                          console.log('Connection has been established successfully.');
        }, function(err) {
                  console.log('Unable to connect to the database:', err);
        });

//on exporte pour utiliser notre connexion depuis les autre fichiers :
var exports = module.exports = {};  exports.sequelize = sequelize;
```

1.2. Configuration d'un modèle de persistance (sequelize)

Cette première approche se focalisera sur les définitions fondamentales , certains détails seront approfondis ultérieurement :

db-model.js

```
var sequelize = require('./db-config.js');
var Sequelize = require("sequelize");//Sequelize.STRING,Sequelize.INTEGER, ....

class Customer extends Sequelize.Model {};
Customer.init({
      /*id: {type: Sequelize.INTEGER, autoIncrement: true, primaryKey: true},*/
         lastName: { type: Sequelize.STRING(64),allowNull: false          },
         firstName: { type: Sequelize.STRING(64),allowNull: false          },
         phoneNumber: { type: Sequelize.STRING(16),allowNull: false    },
         email: { type: Sequelize.STRING(64),allowNull: true        }
      },
      { sequelize, modelName: 'customer' ,  freezeTableName: true });

var AddressOfCustomer = sequelize.define('addressOfCustomer', {
         idAddr: {type: Sequelize.INTEGER, autoIncrement: true, primaryKey: true},
         numberAndStreet: { type: Sequelize.STRING(64),allowNull: false        },
         zip: { type: Sequelize.STRING(64),allowNull: false          },
         town: { type: Sequelize.STRING(64),allowNull: false        }
         },
         { tableName: 'address_of_customer', timestamps: false , underscored : true});

var exports = module.exports = {};
exports.sequelize = sequelize;
exports.AddressOfCustomer = AddressOfCustomer;
exports.Customer = Customer;
```

NB : 2 façons de définir une classe de persistance :

```
class Xxx extends Sequelize.Model {};
   et Xxx.init({attrDefs } , { sequelize, modelName: 'xxx' });
```

ou bien

```
var Xxx = sequelize.define( { attrDefs} , { tableName= 'Xxx' , ... });
```

sachant que sequelize.define() appel .init() en interne

NB: si ***timestamps : true*** (par défaut) alors

```
CREATE TABLE IF NOT EXISTS "Xxx" (
 "id" SERIAL,
 "createdAt" TIMESTAMP WITH TIME ZONE NOT NULL,
 "updatedAt" TIMESTAMP WITH TIME ZONE NOT NULL,
 ...)
```

Nom de table par défaut selon paramètre freezeTableName :

Si modelName="customer" alors

default tableName = "customer**s**" *(with **s** suffix)* without freezeTableName: true

ou bien

default tableName = "customer" *(without s suffix)* with ***freezeTableName: true***

Clef primaire par défaut avec Sequelize pour chaque table :

id: {type: Sequelize.INTEGER or ... or Sequelize.SERIAL ,
autoIncrement: true, primaryKey: true},

Création des tables par défaut si elles n'existent pas déjà :

Type d'ordre SQL déclenché à l'initialisation de "Sequelize" :

```
CREATE TABLE IF NOT EXISTS `customer`
```

Conséquences :

- Si la table 'cutomer' existe déjà avec une structure différente ==> BUG , la TABLE ou bien la DATABASE doit être re-créée (drop , create) .
- En mode développement un script de création/réinitialisation de base peut être pratique et a souvent besoin d'être relancé :

```
DROP DATABASE IF EXISTS minibank_db_node2;
CREATE DATABASE minibank_db_node2 charset=utf8;
```

Nom de colonnes "snake_case" ou "camelCase" :

- Par défaut les noms des colonnes sont les mêmes que ceux mentionnés dans les définitions (souvent en "camelCase" (ex : *idAdr* , *numberAndStreet*).
- Si le paramètre **underscored** est fixé à la valeur **true** alors tous les noms de colonne d'une table seront en "snake_case" (ex: **id_addr** , **number_and_street**)

NB : Plein d'autres détails sont précisés dans la documentation officielle de "sequelize" .

1.3. Utilisation du modèle de persistance

my-sequelize-app.js

```
var MyModel = require('./db-model.js');

//MyModel.sequelize.sync()
MyModel.sequelize.sync({logging: console.log})
                        .then( ()=>{ doJobWithSequelize();})
                        .catch( (err) => { console.log('An error occurred :', err);  });
```

1.4. Insertions/créations :

```
MyModel.Customer.create({id:null, firstName: 'Jean', lastName: 'Bon',
                         phoneNumber : '0605040302',
                         email: 'jean.bon@charcuterie.fr'  })
.then((c) => { console.log("saved Customer :" + JSON.stringify(c));
        console.log("Customer auto-generated ID:", c.id);          }
  );
```

==>
saved Customer : { "id":1,"firstName":"Jean","lastName":"Bon",
"phoneNumber":"0605040302","email":"jean.bon@charcuterie.fr"}
Customer auto-generated ID: 1

1.5. Select/recherches :

Recherches multiples :

```
const findCriterias = {
      where: {lastName : 'Therieur'}, //on veux uniquement ceux qui ont lastName = 'Therieur'
      order: [['lastName', 'ASC']] //classer par ordre alphabétique sur le lastName
   };

//MyModel.Customer.findAll().then(adrs => {
MyModel.Customer.findAll(findCriterias).then(customers => {
      //on récupère ici un tableau "customers" contenant une liste de customers
      console.log("*** Customers de nom=Therieur :" + JSON.stringify(customers));
   })
.catch(function (e) {  console.log(e);    });
```

```
==>
*** Customers de nom=Therieur :
[{"id":2,"lastName":"Therieur","firstName":"Alex","phoneNumber":"0605040301",
"email":"alex.therieur@charcuterie.fr"},
{"id":3,"lastName":"Therieur","firstName":"Alain","phoneNumber":"0608040301",
"email":"alex.therieur@charcuterie.fr"}]
```

Recherche unique/précise :

```
MyModel.Customer.findOne( { where : { id : 1} } ).then(c => {
      //on récupère ici l'unique Customer recherchée par sont id (primary key)
      console.log("** Customer avec id=1 : " + JSON.stringify(c));
   }) ;
```

```
==>
** Customer avec  id=1 : {"id":1,"lastName":"Bon","firstName":"Jean",
"phoneNumber":"0605040302","email":"jean.bon@charcuterie.fr"}
```

Recherche selon clef primaire :

```
MyModel.Customer.findByPk(1).then(c => {
      console.log("** Customer avec id=1 : " + JSON.stringify(c));
   }).catch(function (e) {       console.log(e);    });
```

Recherche avec requête SQL brute spécifique (comme si sans ORM) :

```
const mySpecificRawSqlRequest = "SELECT firstName as prenom , lastName as nom ,
phoneNumber as telephone FROM customer WHERE id = $1";
const param_id_customer=1;
MyModel.sequelize.query(mySpecificRawSqlRequest,
                                {bind: [param_id_customer],
                                 type: MyModel.sequelize.QueryTypes.SELECT})
         .then(results => {
      console.log(">>>> Specific Request results="+JSON.stringify(results));
   }) ;
```

```
==>
>>>> Specific Request results=[{"prenom":"Jean","nom":"Bon","telephone":"0605040302"}]
```

1.6. Update/Mise à jour :

//exemple de requête d'update d'un Customer :

```
MyModel.Customer.update(
        {phoneNumber: '0606060606'},
        {where: {id: 1}}
   ).then(nbCustomersModifies => {
      console.log("nbCustomersModifies:" + nbCustomersModifies);
   }).catch(function (e) {
      console.log(e);
   });
```

==> nbCustomersModifies:1

1.7. Delete/suppression :

```
MyModel.Customer.destroy({  where: { id: 3 }}).then(() => {
  console.log("Customer avec id=3 supprimé");
}).catch(function (e) {
    console.log(e);
 });
```

```
//fonction supprimant tous les éléments d'une table selon le modelClassName "Customer" or ... :
function deleteAllPromise(modelClassName){
        return MyModel[modelClassName].destroy({ where: {} });
}
```

1.8. Enchaînement "CRUD" avec Promise ".then ... .then"

Rappel important :
Les fonctions de l'api "Sequelize" sont **asynchrones** et les valeurs de retour sont des "**Promise**" que l'on peut gérer par .then().then().catch().

Exemple d'enchaînement simple :

L'exemple suivant va enchaîner :
- la suppression de tous les anciens "customers"
- insertions d'un "customer"
- récupération du client enregistré dans variable c1 (avec c1.id = résultat auto_increment)
- idem pour c2 et c3
- la récupération de tous les "Customer" (une fois toutes les insertions effectuées)
- la mise à jour en base du numéro de téléphone du client c1
- la récupération des valeurs modifiées et enregistrées du client c1 (avec nouveau numéro de téléphone)

```
let c1,c2,c3;
deleteAllPromise("Customer").
then(()=>MyModel.Customer.create({id:null, firstName: 'Jean', lastName: 'Bon',
            phoneNumber : '0605040302',    email: 'jean.bon@charcuterie.fr'  }))
.then((c) => { c1 = c;
     return MyModel.Customer.create({id:null, firstName: 'Alex', lastName: 'Therieur',
           phoneNumber : '0605040301',    email: 'alex.therieur@charcuterie.fr'  });
})
.then((c) => { c2 = c;
          return MyModel.Customer.create({id:null, firstName: 'Alain', lastName: 'Therieur',
       phoneNumber : '0608040301',    email: 'alex.therieur@charcuterie.fr'  });
})
.then((c) => { c3 = c;  })
.then(() => /* implicit return without { } */ MyModel.Customer.findAll())
.then(customers => { console.log("*** Customers :" + JSON.stringify(customers));    })
.then(()=>MyModel.Customer.update({phoneNumber: '0606060606'},
                          {where: {id: c1.id}})
 )
.then(nbCustomersModifies => {
      console.log("nbCustomersModifies:" + nbCustomersModifies);
                return MyModel.Customer.findByPk(c1.id);
   })
.then(c => console.log("** updated Customer avec id=1 : " + JSON.stringify(c))    )
.catch((e) =>{  console.log(e); })
```

==>
**** Customers :*
*[{"id":4,"lastName":"Bon","firstName":"Jean",**"phoneNumber":"0605040302"**,"email":"jean.bon@charcuterie.fr"},"id":5,"lastName":"Therieur","firstName":"Alex","phoneNumber":"0605040301","email":"alex.therieur@charcuterie.fr"},"id":6,"lastName":"Therieur","firstName":"Alain","phoneNumber":"0608040301","email":"alex.therieur@charcuterie.fr"}]*
nbCustomersModifies:1
*** updated Customer avec id=1 : {"id":4,"lastName":"Bon","firstName":"Jean",*
***"phoneNumber":"0606060606"**,"email":"jean.bon@charcuterie.fr"}*

Même exemple plus lisible avec async/await de es2017 :

```
async function doJob1WithSequelizeAndAsyncAwait(){

try{
await deleteAllPromise("Customer");
let c1= await MyModel.Customer.create({id:null, firstName: 'Jean', lastName: 'Bon',
     phoneNumber : '0605040302',  email: 'jean.bon@charcuterie.fr'  });
let c2 = await MyModel.Customer.create({id:null, firstName: 'Alex', lastName: 'Therieur',
           phoneNumber : '0605040301',    email: 'alex.therieur@charcuterie.fr'  });
let c3 = await MyModel.Customer.create({id:null, firstName: 'Alain', lastName: 'Therieur',
        phoneNumber : '0608040301',    email: 'alex.therieur@charcuterie.fr'  });
const customers = await MyModel.Customer.findAll();
console.log("### Customers :" + JSON.stringify(customers));
let nbCustomersModifies = await MyModel.Customer.update({phoneNumber: '0606060606'},
                          {where: {id: c1.id}});
console.log("nbCustomersModifies:" + nbCustomersModifies);
c1= await MyModel.Customer.findByPk(c1.id);
console.log("## updated Customer avec id=1 : " + JSON.stringify(c1))
}
catch(e){
          console.log(e);
}

} //Rappel : await ne peut être appelé qu'au sein d'une fonction déclarée "async" et permet
d'attendre la valeur issue de la résolution d'une promesse (valeur de retour d'une fonction appelée)
```

1.9. Associations (jointures) : vue d'ensemble

- BelongsTo
- HasOne
- HasMany
- BelongsToMany

1.10. Associations "1-n" / "n-1" classique (foreignKey)

Exemple "plusieurs Operations bancaires pour un Compte" :

```
var sequelize = require('./db-config.js');
var Sequelize = require("sequelize"); //Sequelize.STRING,Sequelize.INTEGER, ....

class Account extends Sequelize.Model {};
Account.init({
    number: {type: Sequelize.INTEGER, autoIncrement: true, primaryKey: true},
   label: { type: Sequelize.STRING(64),allowNull: false   },
   /*solde*/balance: { type: Sequelize.DOUBLE ,allowNull: false  , validate : { min: -1500 }   }
}, { sequelize, modelName: 'account' ,  freezeTableName: true });

class Operation extends Sequelize.Model {};
Operation.init({
   number: {type: Sequelize.INTEGER, autoIncrement: true, primaryKey: true},
   label: { type: Sequelize.STRING(64),allowNull: false   },
   amount: { type: Sequelize.DOUBLE ,allowNull: false            },
  dateOp: { type: Sequelize.DATEONLY ,allowNull: false          }
}, { sequelize, modelName: 'operation' ,  freezeTableName: true });

Operation.belongsTo(Account); //une opération est un détail d'un seul compte
//--> on pourra accéder au compte lié à l'opération courante via op.account .

Account.hasMany(Operation);  //un compte peut comporter plusieurs opérations
//--> on pourra parcourir les opérations liées à un compte via compteXy.operations
//--> on pourra ajouter une operation à un compte via compteXy.addOperation(opZz) ;

var exports = module.exports = {};  exports.sequelize = sequelize;
exports.Account = Account;
exports.Operation = Operation;
```

Remarques :

Effet sur le modèle relationnel :

Une seule des 2 instructions suivantes (***Operation.belongsTo(Account);*** ou bien ***Account.hasMany(Operation);***) suffit à générer une clef étrangère dans la table "operation(s)" .

Le nom par défaut de la clef étrangère (ici dans la table operation) est '***accountNumber***'
soit = ***modelName associé*** + ***pkName du modèle (table) référencé(e)*** .

Effet de "as :" sur le modèle objet (javascript) :

```
//Account.hasMany(Operation , { as : "lastOperations" } );
```

as ="roleName" correspond à une vision orientée objet de l'association (role UML)
et correspondra à un attribut de l'objet javascript permettant de naviguer d'un niveau à l'autre
NB: **la valeur par défaut est le modelName référencé (avec un suffixe "s" si "many").**
la valeur de as est considérée par sequelize comme un "alias" qu'il faut régulièrement préciser
(exemple : MyModel.Account.findAll({ include: [{model: MyModel.Operation , as: "lastOperations" }] }))
==>pour faire simple modelName commençant par une minuscule et pas trop de "as:"

Exemple de code exploitant l'association "n-1" / "1-n" :

```
let ac1 ;
...
.then( () => MyModel.Account.create({number:null, label: 'bank_account_xx', balance : 100.0  }) )
.then( (ac) => { ac1=ac;
        return MyModel.Operation.create({number:null, label: 'achat x1',
                                                amount : -50.0 , dateOp:"2019-05-23"  });
        })
.then((newOp) => ac1.addOperation(newOp))
...
```

NB : ***.addOperation()*** existe du fait de l'instruction Account.***hasMany***(Operation);

```
...
.then(()=> MyModel.Account.findAll( { include: [{model: MyModel.Operation} ] }))
.then(accounts => {
      //console.log("*** accounts =" + JSON.stringify(accounts));
      accounts.forEach((account) => {
         //console.log("** account =" + JSON.stringify(account));
         console.log(`* account ${account.number} ${account.label} ${account.balance} `);
               account.operations.forEach((op) => {
                  console.log(`****** operation ${op.number} ${op.label}
                                 ${op.amount} ${op.dateOp}`);
                  });
      });
  })
....
```

NB :
{ include: [{model: MyModel.Operation}] } permet une sorte de "***fetch JOINED operations***" pour chaque opération remontée par findAll() .
et ***account.operations*** existe du fait de la déclaration Account.***hasMany***(Operation); au sein du modèle.

2. Utilisation de Sequelize v5.x avec Typescript

config/database.cfg.ts

```
export default{
"dev": {
        "dialect": "mysql",
        "host": "localhost",
        "port": 3306,
        "database": "deviseApiDb",
        "user": "root",
        "password": "root"
        },
"prod": {
        "dialect": "mysql",
        "host": "devise.db.service",
        "port": 3306,
        "database": "deviseApiDb",
        "user": "root",
        "password": "root"
         }
}
```

config/db-config.ts

```
import dbCfg from './database.cfg';
export interface IDbConfig {
        dialect: "mssql" | "mysql" | "postgres" | "sqlite" | "mariadb",
        host: string,
        port: number,
        database : string
        user: string
        password : string ;
}

let mode = process.env.MODE;
//env variable MODE=dev or MODE=prod when launching node
if(mode == undefined) mode = "dev";
//console.log("in db-config, mode="+mode);
export const confDb : IDbConfig =
            (mode == "dev") ? dbCfg.dev as any : dbCfg.prod as any;
console.log("in db-config, confDb.host="+confDb.host);
if(confDb.port === undefined) {
  if(confDb.dialect=="mysql") {
          confDb.port=3306;
  }
}
```

model/devise.ts

```
//GENERIC PART (SEQUELIZE OR NOT)
export interface Devise  {
  code :string ;
  nom :string ;
  change :number ;
}
//exemples: ("USD" , "dollar" , 1) ,  ("EUR" , "euro" , 0.9)

//real class for instanciation ,  with constructor .
export class DeviseObject implements Devise {
  constructor(public code:string = "?" ,   public nom:string = "?",  public change:number= 0){
  }
}
```

Le fichier ci-dessus est volontairement indépendant de Sequelize !!!

La partie spécifique "Sequelize" est placée dans le fichier annexe suivant :

model/sq-devise.ts

```
import { Devise } from './devise';
import { Sequelize, Model, DataTypes, BuildOptions } from 'sequelize';
/* import { HasManyGetAssociationsMixin, HasManyAddAssociationMixin,
HasManyHasAssociationMixin, Association, HasManyCountAssociationsMixin,
HasManyCreateAssociationMixin } from 'sequelize'; */


//SqDevise est une interface mixant la structure de données "Devise" au "Model" Sequelize :
export interface /*DeviseModel*/ SqDevise extends Model, Devise {
}
 // Need to declare the static model so `findOne` etc. use correct types.
 export type DeviseModelStatic = typeof Model & {
   new (values?: object, options?: BuildOptions): SqDevise /*DeviseModel*/;
 }
 // fonction exportée (à appeler) définissant la structure de la table (dans base de données) :
 export function initDeviseModel(sequelize: Sequelize):DeviseModelStatic{
 const DeviseDefineModel = <DeviseModelStatic> sequelize.define('devise', {
     code: {type: DataTypes.STRING(32), autoIncrement: false, primaryKey: true},
     nom: { type: DataTypes.STRING(64),allowNull: false          },
     change: { type: DataTypes.DOUBLE,allowNull: false          }
     }, {    freezeTableName: true , });
 return DeviseDefineModel;
}
```

NB :

- avec Sequelize v4 , il fallait utiliser @types/sequelize (avec XxxAttributes , XxxInstance , ...)
- depuis Sequelize v5, la partie "définitions typescript" est déjà intégrée dans "sequelize" et il faut faire avec la documentation officielle "http://docs.sequelizejs.com/manual/typescript"
- besoin de `@types/node` ,`@types/validator` , `@types/bluebird` dans package.json :

```
"devDependencies": {
"@types/bluebird": "^3.5.27",
"@types/express": "^4.16.1",
"@types/node": "^12.0.8",
"@types/validator": "^10.11.1",
}
```

model/global-db-model.ts

```
import { Sequelize , Model }from "sequelize";
import { DeviseModelStatic , initDeviseModel } from './sq-devise';
import { confDb  } from "../config/db-config"
//import { PaysModelStatic , initPaysModel } from "./sq-pays";

export class MyApiModels {
   public devises! : DeviseModelStatic;
   //public pays! : PaysModelStatic;
 }

 export class MySqDatabase {
   private models: MyApiModels;
   private sequelize: Sequelize;
   public dbname: string = "unknown";

   constructor() {
     this.models = new MyApiModels();
     let model: any;
     let sqOptions = {
          dialect: confDb.dialect,
          port : confDb.port,
          host : confDb.host,
```

```
        logging: /*console.log*/false, // false or console.log,// permet de voir les logs de sequelize
        define: {
           timestamps: false
        }
    };
    var password = confDb.password ? confDb.password : "";
    this.sequelize = new Sequelize(confDb.database, confDb.user, password, sqOptions);
    this.dbname = confDb.database;

    this.models.devises= initDeviseModel(this.sequelize);
    //this.models.pays= initPaysModel(this.sequelize) ;
  }

  getModels() {      return this.models;    }
  getSequelize() {      return this.sequelize;    }
}

export const database: MySqDatabase = new MySqDatabase();
export const models = database.getModels();
export const sequelize: Sequelize = database.getSequelize();

export function initSequelize(){
  sequelize.sync({logging: console.log})
        .then( ()=>{    console.log("sequelize is initialized");   }
        ).catch( (err:any) => { console.log('An error occurred :', err);  });
}
```

Utilisation au sein de **dao/sqDeviseService**

```
import { DeviseDataService } from './deviseDataService'
import { Devise } from '../model/devise';
import { SqDevise, DeviseModelStatic } from '../model/sq-devise';
import { models } from '../model/global-db-model';
import { NotFoundError, ConflictError } from '../api/apiErrorHandler';
//"strictNullChecks": false in tsconfig.json
```

```
export class SqDeviseService implements DeviseDataService{

  deviseModelStatic : DeviseModelStatic = models.devises;

  constructor(){}

  findById(code: string): Promise<Devise> {
    return new Promise<Devise>((resolve: Function, reject: Function) => {
      this.deviseModelStatic.findByPk(code)
       .then((obj: SqDevise) => {
         //returning null by default if not Found
         if(obj!=null)
            resolve(obj);
         else
           reject(new NotFoundError("devise not found with code="+code));
        })
        .catch((error: Error) => { reject(error);});
     });
  }

  findAll(): Promise<Devise[]> {
    return new Promise((resolve,reject) => {
       this.deviseModelStatic.findAll()
      .then((objects: SqDevise[]) => { resolve(objects);  })
       .catch((error: Error) => {  reject(error);  });
    });
  }


  insert(d: Devise): Promise<Devise> {
    return new Promise((resolve,reject) => {
       this.deviseModelStatic.create(d)
       .then((obj: SqDevise) => { resolve(obj); })
       .catch((error: any) => {   reject(error);  });
   });
  }
```

```
  saveOrUpdate(d: Devise): Promise<Devise> {
    return new Promise((resolve,reject) => {
      //.upsert() is appropriate for saveOrUpdate if no auto_incr
      this.deviseModelStatic.upsert(d)
      .then((ok: Boolean) => {  resolve(d); })
      .catch((error: any) => {  reject(error); });
   });
  }

  deleteById(codeDev: string): Promise<void> {
     return new Promise((resolve,reject) => {
        this.deviseModelStatic.destroy( { where: { code : codeDev} } )
        .then(() => {  resolve(); })
        .catch((error: any) => {   reject(error);  });
    });
  }
}
```

2.1. compléments si associations

model/sq-devise.ts

```
...
import { HasManyGetAssociationsMixin, HasManyAddAssociationMixin,
HasManyHasAssociationMixin, Association, HasManyCountAssociationsMixin,
HasManyCreateAssociationMixin } from 'sequelize';
...
export interface /*DeviseModel*/ SqDevise extends Model, Devise {
 getPays: HasManyGetAssociationsMixin<Pays>;
 addPays: HasManyAddAssociationMixin<Pays, number>;
 hasPays: HasManyHasAssociationMixin<Pays, number>;
 countPays: HasManyCountAssociationsMixin;
}
...
```

NB : sqDeviseEuro.**addPays**(objPaysFr) ou bien sqDeviseEuro.**addPays**(idPaysFr)

dans **model/global-db-model.ts**

```
...
 export class MySqDatabase {
   ...
```

```
  constructor() {
    ....
   this.models.devises= initDeviseModel(this.sequelize);
   this.models.pays= initPaysModel(this.sequelize);
   this.models.devises.hasMany(this.models.pays);
  }
 ....
 }
 ...
```

Exemple d'utilisation :

```
async function doAssociationJob(){
 try{
   let devC1 : SqDevise =
    await models.devises.findByPk("CC1", { include :[models.devises.associations.pays]});
  if(devC1==null){
              devC1= await models.devises.create({code: 'CC1',name: 'MonaieC1',change : 1234 });
              let pays1forDevC1 :Pays = await models.pays.create({name:"Pays1",capitale:"Cap1"});
              await devC1.addPays(pays1forDevC1);
              console.log(JSON.stringify(pays1forDevC1));
              let pays2forDevC1 :Pays = await models.pays.create({name:"Pays2",capitale:"Cap2"});
              await devC1.addPays(pays2forDevC1);
              console.log(JSON.stringify(pays2forDevC1));
             }
              //console.log(devC1.code, devC1.name, devC1.change);
              console.log(JSON.stringify(devC2)); //devise avec sous partie .pays

             }
      catch(e){
            console.log(JSON.stringify(e));
         }
}
```

XIII - Annexe – apidoc (pour api rest)

1. Api doc (swagger-ui-express et swagger-jsdoc)

NB: L'ancienne technologie "**http://apidocjs.com**" **est de moins en moins utilisée .**
La technologie swagger-ui-express est nettement plus populaire .

package.json

```
…
"dependencies": {
   "express": "^5.1.0",
   "swagger-jsdoc": "^6.2.8",
   "swagger-ui-express": "^5.0.1"
  },
...
```

XIV - Annexe – Utilitaires (grunt , gulp , ...)

1. GRUNT

1.1. Grunt (Javascript Task Runner)

"**Grunt**" s'appuyant lui même sur npm pour le téléchargement/installation , permet d'**automatiser des tâches de développement répétitives** telles que :
- **nettoyage** de certains répertoires et fichiers temporaires (via *grunt-contrib-clean*)
- **analyse/vérification** du code javascript (via *grunt-contrib-jshint*)
- éventuelle **pré-compilations** (ex : scss → css ,
  ts (typeScript/anagular2) → js ,
  …) via *grunt-typescript* , ...
- éventuelle concaténation de fichiers javascript (via *grunt-contrib-concat*)
- éventuelle **compression** (*minification*) du code (via *grunt-contrib-uglify ou ...)*
- éventuels déclenchements automatiques dès qu'un fichier a changé *(via grunt-contrib-watch)*
- **lancement de tests** *(via grunt-karma* ou *....)*

la configuration de "**Grunt**" s'effectue via un fichier javascript de configuration (fonction de configuration javascript pilotant des modules "**npm/nodeJs**" et avec données/paramètres de syntaxe proche JSON) .

1.2. Configuration et utilisation de Grunt

Installation (en mode global) du lanceur de Grunt en ligne de commande :

```
npm install -g grunt-cli
```

Installation (locale au projet) de quelques plugins souvent utiles :

```
npm install  grunt  grunt-contrib-uglify  grunt-karma  --save-dev
```
ou bien édition de package.json + npm update

Installation (locale au projet) de quelques autres plugins souvent utiles :

fichier ***package.json***

```
{
 "name": "karma-through-grunt",
 "version": "0.0.1",
 "description": "...",
 "main": "...",
 "scripts": {
   "test": "echo \"Error: no test specified\" && exit 1"
 },
 "author": "...",
 "devDependencies": {
   "karma-jasmine": "~0.3.6",
```

```
    "karma-chrome-launcher": "~0.2.2",
    "karma-jasmine-html-reporter": "~0.2.0",
    "grunt-contrib-clean" : "~0.7.0" ,
    "grunt-contrib-jshint" : "~0.12.0"
  }
}
```

npm update

Configuration de GRUNT via le fichier **gruntfile.js** :

```
module.exports = function(grunt) {
  // Configuration de Grunt
  grunt.initConfig({

          //NB: npm update (after change dependencies in package.json)
          pkg: grunt.file.readJSON('package.json') ,

          clean: {
                // Deletes all old .js files, but skips old min.js files
                js: ["path/to/dir/old/*.js", "!path/to/dir/old/*.min.js"] ,

                // delete all files and directories here
                build: ["build/xx/yy", "dist" , "dest"],
          } ,

          jshint: {
                        all: ['gruntfile.js', 'js/**/*.js', 'test/**/*.js']
                        // options in .jshintrc file
                        },

           uglify: {
                        my_target: {
                          files: [
                                { src: 'js/*js', dest: 'dest/common.js'},
                                { src: 'test/unit/*js', dest: 'dest/test.js'}
                        ]
          }
      } ,

      karma: {
         unit: {
            configFile: 'karma.conf.js'
         }
      }
  });

  // A very basic logging task :
  grunt.registerTask('basic_log', '', function() {
    grunt.log.write('Logging some stuff...').ok();
  });

  // Définition des tâches/plugins Grunt :
```

```
  grunt.loadNpmTasks('grunt-contrib-clean');
  grunt.loadNpmTasks('grunt-contrib-jshint');
  grunt.loadNpmTasks('grunt-contrib-uglify');
  grunt.loadNpmTasks('grunt-karma');

  grunt.registerTask('default', [ 'basic_log' , 'clean' , 'jshint' ,'uglify', 'karma']);
};
```

Exemple de contenu du fichier .jshintrc

```
{  "curly": true,
  "eqnull": true,
  "eqeqeq": true,
  "undef": true,
  "globals": {
    "jQuery": true
  }
}
```

Lancement de GRUNT (depuis le répertoire contenant gruntfile.js) :

grunt

2. GULP

Ancien exemple (pour angular 2) :

gulpfile.js

```
const gulp = require("gulp");
const del = require("del");
const tsc = require('gulp-typescript');
const sourcemaps = require('gulp-sourcemaps');
const ignore = require('gulp-ignore');
const tscProject = tsc.createProject("tsconfig.json");

gulp.task('default', ['build']);

gulp.task('build', ['copy_resources','compile','copy-libs' ], function(){
    console.log("Building the project ...");
});

gulp.task('clean', function () {
```

```
  return del('dist/**/*');
});

// TypeScript compile (mode "dev" avec sourcemaps )
gulp.task('compile', ['tslint'], function () {
    var tsResult =
    gulp.src('src/**/*.ts')
    //tscProject.src() //prend en compte l'option "rootDir" de tsconfig.json
    .pipe(sourcemaps.init())
    .pipe(tscProject(/*tsc.reporter.defaultReporter()*/)) ;
    //.pipe(gulp.dest('dist')); //si pas suite avec sourcemaps.write

    return tsResult.js //.js en plus depuis la v3
        .pipe(sourcemaps.write(".", {sourceRoot: '/src'}))
        .pipe(gulp.dest('dist'));
});



//Copy all resources that are not TypeScript files into build/dist directory (.html , .css , .json, ...)
gulp.task('copy_resources', function (){
    return gulp.src(["src/**/*", "!**/*.ts"])
          .pipe(gulp.dest("dist"));
});

gulp.task("copy-libs",[ ] ,  function (){
    return gulp.src([
            'reflect-metadata/Reflect.js',
            'zone.js/dist/zone.js',
            'core-js/client/shim.min.js',
            'systemjs/dist/system-polyfills.js',
            'systemjs/dist/system.src.js',
            'rxjs/**',
            '@angular/core/bundles/core.umd.js',
            '@angular/common/bundles/common.umd.js',
            '@angular/compiler/bundles/compiler.umd.js',
            '@angular/platform-browser/bundles/platform-browser.umd.js',
```

```
        '@angular/platform-browser-dynamic/bundles/platform-browser-dynamic.umd.js',
        '@angular/http/bundles/http.umd.js',
        '@angular/router/bundles/router.umd.js',
        '@angular/forms/bundles/forms.umd.js',
        '@angular/upgrade/bundles/upgrade.umd.js'
      ], {cwd: "node_modules/**"}) /* Glob required here. */
      .pipe(ignore.exclude([ "**/*.map" , "**/*.ts" , "**/*.txt" , "**/*.md" , "**/*.json"]))
      .pipe(gulp.dest("dist/lib"));
});
```

Voici finalement la configuration "npm" nécessaire pour "gulp":

package.json

```
…
"scripts": {
…
 "build": "gulp clean && gulp build"
},
…
 "devDependencies": {
    ...
    "del": "^2.2.0",
    "gulp": "^3.9.1",
    "gulp-sourcemaps": "^2.2.0",
    "gulp-typescript": "^3.1.2",
    "gulp-ignore": "^2.0.2",
   ...
  },
...
```

Commande à lancer (dans un terminal depuis la racine du projet) :

une seule fois :

```
npm i -g gulp
npm i -g gulp-cli
npm install
```

régulièrement :

gulp clean

gulp build

…

ou bien indirectement

npm run build

XV - Annexe – WEB Services "REST"

1. Généralités sur Web-Services REST

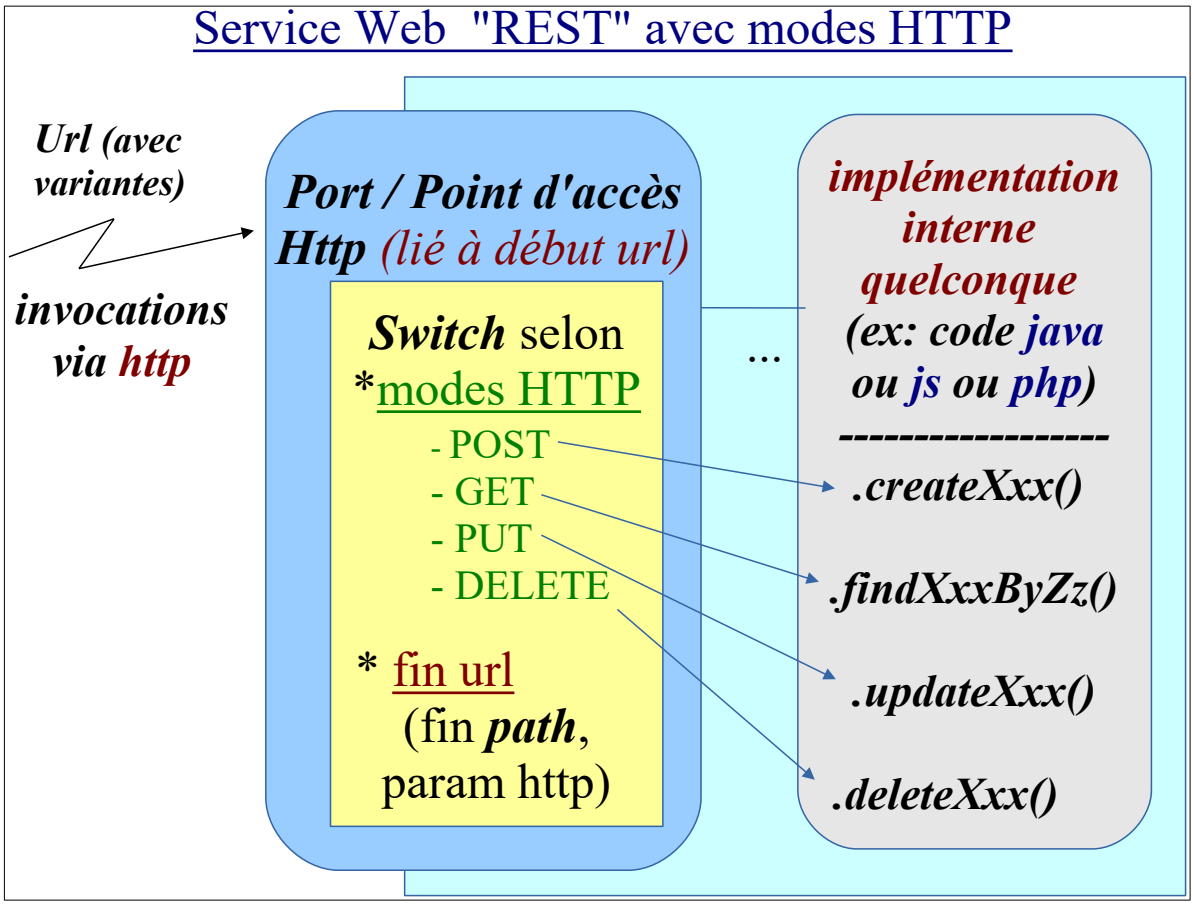
2 grands types de services WEB: **SOAP/XML** et **REST/HTTP**

WS- (SOAP / XML)*

- "**Payload**" systématiquement en **XML** *(sauf pièces attachées / HTTP)*
- **Enveloppe SOAP** en XML *(header facultatif pour extensions)*
- **Protocole de transport au choix (HTTP, JMS, ...)**
- Sémantique quelconque (*appels méthodes*) , **description WSDL**
- **Plutôt** orienté Middleware SOA **(arrière plan)**

REST (HTTP)

- **"Payload" au choix (XML , HTML , JSON, ...)**
- Pas d'enveloppe imposée
- **Protocole de transport = toujours HTTP.**
- Sémantique "**CRUD**" (***modes http*** **PUT,GET,POST,DELETE)**
- **Plutôt** orienté IHM Web/Web2 **(avant plan)**



Points clefs des Web services "REST"

Retournant des données dans un format quelconque ("XML" , "JSON" et éventuellement "txt" ou "html") les web-services "REST" offrent des résultats qui nécessitent généralement peu de re-traitements pour être mis en forme au sein d'une IHM web.

Le format "au cas par cas" des données retournées par les services REST permet peu d'automatisme(s) sur les niveaux intermédiaires.

Souvent associés au format "JSON" les web-services "REST" conviennent parfaitement à des appels (ou implémentations) au sein du langage javascript .

La relative simplicité des URLs d'invocation des services "REST" permet des appels plus immédiats (*un simple href="..." suffit en mode GET pour les recherches de données*) .

La compacité/simplicité des messages "JSON" souvent associés à "REST" permet d'obtenir d'assez bonnes performances .

REST = style d'architecture (conventions)

REST est l'acronyme de **Representational State Transfert**.
C'est un **style d'architecture** qui a été décrit par ***Roy Thomas Fielding*** dans sa thèse *«Architectural Styles and the Design of Network-based Software Architectures»*.

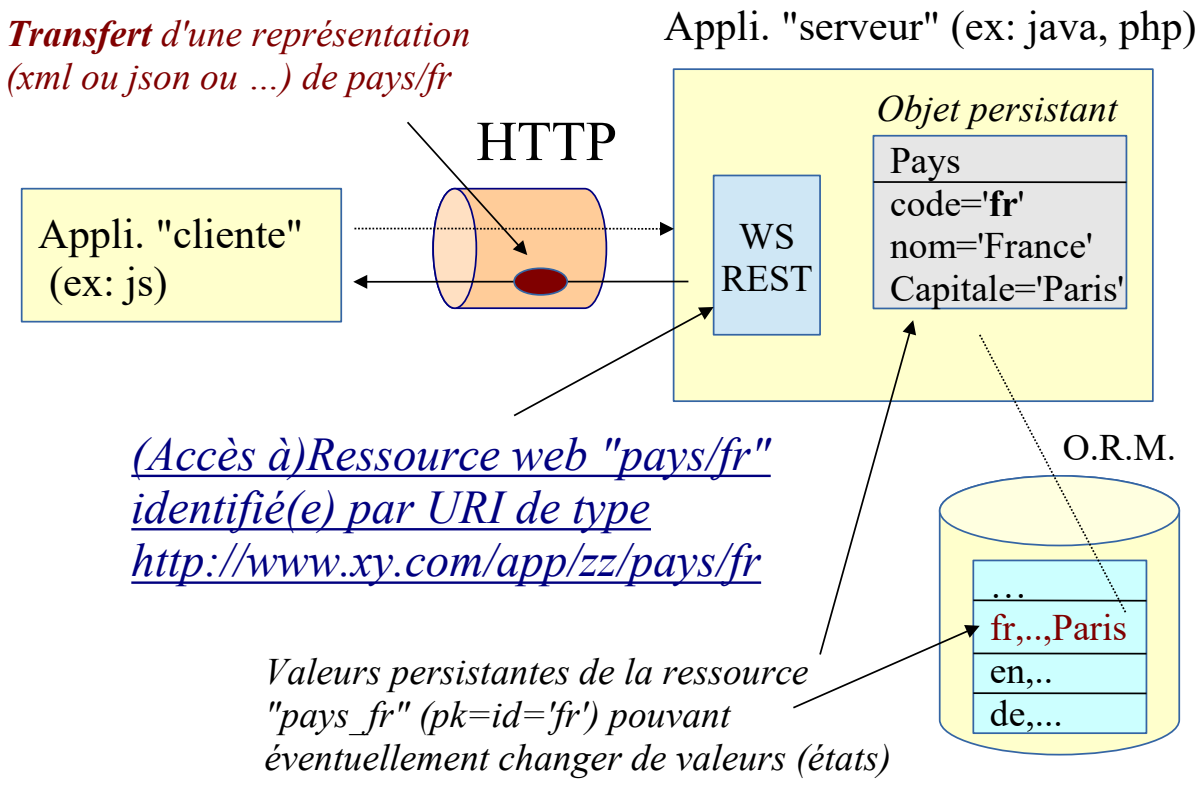
L'information de base, dans une architecture REST, est appelée **ressource**.
Toute information (à sémantique stable) qui peut être nommée est une ressource: un article , une photo, une personne , un service ou n'importe quel concept.

Une ressource est identifiée par un **identificateur de ressource**. Sur le web ces identificateurs sont les **URI** (Uniform Resource Identifier).

NB: dans la plupart des cas, une ressource REST correspond indirectement à un enregistrement en base (avec la *clef primaire* comme partie finale de l'uri "identifiant").

Les composants de l'architecture REST manipulent ces ressources en **transférant à travers le réseau** (via HTTP) des **représentations de ces ressources**.
Sur le web, on trouve aujourd'hui le plus souvent des représentations au format **HTML , XML ou JSON**.

REST : transferts de représentations de ressources



REST et principaux formats (xml,json)

Une invocation d'URL de service REST peut être accompagnée de données (en entrée ou en sortie) pouvant prendre des formats quelconques :
text/plain , text/html , application/xml , application/json , …

Dans le cas d'une lecture/recherche d'informations , le format du résultat retourné pourra (selon les cas) être :

- **imposé (en dur) par le code du service REST** .

- **au choix (xml , json) et précisé par une partie de l'url**

- **au choix (xml , json) et précisé par le champ "Accept :" de l'entête HTTP de la requête**. (exemple: Accept: application/json) .

Dans tous les cas, la réponse HTTP devra avoir son format précisé via le champ habituel ***Content-Type:*** *application/json* de l'entête.

Format JSON (JSON = *JavaScript Object Notation*)

Les 2 principales caractéristiques de JS0N sont :

- Le principe de clé / valeur (map)
- L'organisation des données sous forme de tableau

Les types de données valables sont :

- tableau
- objet
- chaîne de caractères
- valeur numérique (entier, double)
- booléen (true/false)
- null

```
[
  {
    "nom": "article a",
    "prix": 3.05,
    "disponible": false,
    "descriptif": "article1"
  },
  {
    "nom": "article b",
    "prix": 13.05,
    "disponible": true,
    "descriptif": null
  }
]
```

une liste d'articles

une personne

```
{
  "nom": "xxxx",
  "prenom": "yyyy",
  "age": 25
}
```

REST et méthodes HTTP (verbes)

Les **méthodes HTTP** sont utilisées pour indiquer la **sémantique des actions demandées** :

- **GET** : **lecture/recherche** d'information
- **POST** : **envoi** d'information
- **PUT** : **mise à jour** d'information
- **DELETE** : **suppression** d'information

Par exemple, pour récupérer la liste des adhérents d'un club, on peut effectuer une requête de type **GET** vers la ressource **http://monsite.com/adherents**

Pour obtenir que les adhérents ayant plus de 20 ans, la requête devient http://monsite.com/adherents?ageMinimum=20

Pour supprimer numéro 4, on peut employer une requête de type **DELETE** telle que **http://monsite.com/adherents/4**

Pour envoyer des informations, on utilise **POST** ou **PUT** en passant les informations dans le corps (invisible) du message HTTP avec comme URL celle de la ressource web que l'on veut créer ou mettre à jour.

Exemple concret de service REST : "Elevation API"

L'entreprise "***Google***" fourni gratuitement certains services WEB de type REST.
"***Elevation API***" est un service REST de Google qui renvoie l'altitude d'un point de la planète selon ses coordonnées (latitude ,longitude) .
La documentation complète se trouve au bout de l'URL suivante :

https://developers.google.com/maps/documentation/elevation/?hl=fr

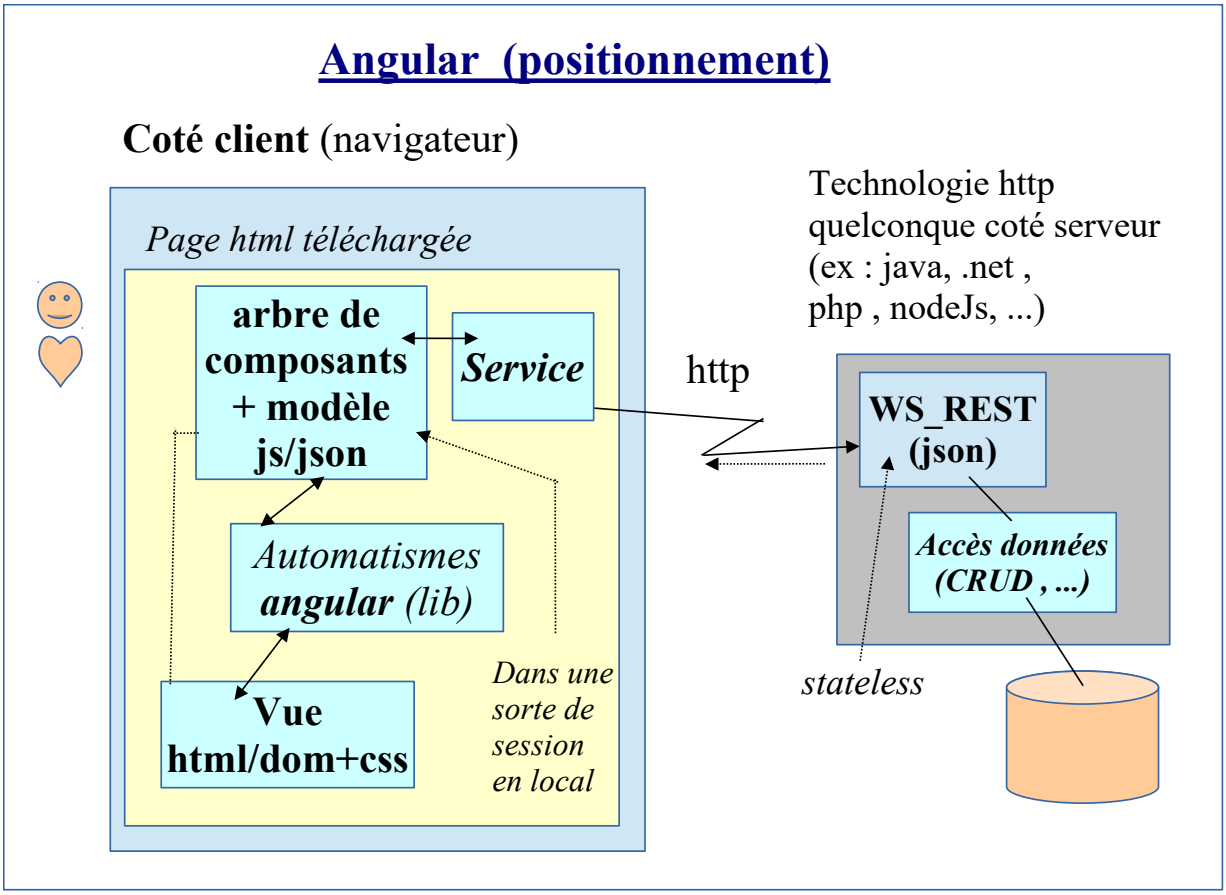
Sachant que les coordonnées du Mont blanc sont :
Lat/Lon : 45.8325 N / 6.86417 E *(GPS : 32T 334120 5077656)*
Les invocations suivantes (du service web rest "api/elevation")

http://maps.googleapis.com/maps/api/elevation/json?locations=45.8325,6.86417
http://maps.googleapis.com/maps/api/elevation/xml?locations=45.8325,6.86417

donne les résultats suivants "json" ou "xml":

```
{  "results" : [
     {
       "elevation" : 4766.466796875,
       "location" : {
         "lat" : 45.8325,
         "lng" : 6.86417
       },
       "resolution" : 152.7032318115234
     }
   ],   "status" : "OK"
}
```

```
?xml version="1.0" encoding="UTF-8"?>
<ElevationResponse>
 <status>OK</status>
 <result>
  <location>
   <lat>45.8325000</lat>
   <lng>6.8641700</lng>
  </location>
  <elevation>4766.4667969</elevation>
  <resolution>152.7032318</resolution>
 </result>
</ElevationResponse>
```



Conventions sur URL / Path des ressources REST

Type requêtes	HTTP *Method*	*URL ressource(s) distante(s)*	*Request body*	*Réponse JSON*
Recherche multiple	**GET**	**…/product** **…/product?crit1=v1&crit2=v2**	vide	Liste/tableau d'objets
Recherche par id	**GET**	**…/product/idRes** *(avec idRes=1,...)*	vide	Objet JSON
Ajout (seul)	**POST**	**…/product**	Objet JSON	Objet JSON avec id quelquefois calculé (incr)
Mise à jour (seule)	**PUT**	**…/product/idRes** **ou** **…/product**	Objet JSON avec **.id**	Objet JSON mis à jour
SaveOr Update	*POST*	*.../product*	*Objet JSON*	*Objet JSON ajouté (auto_incr id) ou modifié*
suppression	**DELETE**	**…/product/idRes**	vide	Statut et message
Autres	**...**	**…/product-action/opXy/....**	...	...

1.1. Statuts HTTP (code d'erreur ou ...)

Catégories de code/statut HTTP :

1xx	Information (rare)
2xx (ex : 200)	Succès
3xx	Redirection
4xx	Erreur du client
5xx (ex : 500)	Erreur du serveur

Principaux codes/statuts en cas de succès ou de redirection:

200 , OK	Requête traitée avec succès. La réponse selon méthode de requête utilisée
201 , Created	Requête traitée avec succès et création d'un document.
204 , No Content	Requête traitée avec succès mais pas d'information à renvoyer.
301 , *Moved Permanently*	Document déplacé de façon permanente
304 , *Not Modified*	Document non modifié depuis la dernière requête

Principaux codes d'erreurs :

400 , Bad Request	La syntaxe de la requête est erronée (ex : invalid argument)
401 , Unauthorized	Une authentification est nécessaire pour accéder à la ressource.
403, Forbidden	authentification effectuée mais manque de droits d'accès (selon rôles, ...)
404 , Not Found	Ressource non trouvée.
409 , Conflict	La requête ne peut être traitée en l'état actuel.
...	*liste complète sur https://fr.wikipedia.org/wiki/Liste_des_codes_HTTP*
500 , Internal Server Error	Erreur interne (vague) du serveur (ex ; bug , exception , ...) .
501, Not Implemented	Fonctionnalité réclamée non supportée par le serveur
503 , Service Unavailable	Service temporairement indisponible ou en maintenance.

1.2. Variantes classiques

Réponses plus ou moins détaillées (simple "http status" ou bien "message json")

Lorsqu'un serveur répond à une requête en mode POST , il peut soit :

*- retourner le "https status" **201/CREATED** et une réponse JSON comportant toute l'entité sauvegardée coté serveur avec souvent l'id (clef primaire) automatiquement généré ou incrémenté*
*{ **"id" : "a345b6788c335d56"** , "name" : "toto" , ... }*

*- se contenter de renvoyer le "http status" **201/CREATED** avec aucun message de réponse mais avec le champ **Location: /type_entite/idxy** comportant au moins l'id de la resource enregistrée au sein de l'entête HTTP de la réponse .*
*L'application cliente pourra alors effectuer un second appel en mode GET avec une fin d'URL en **/type_entite/idxy** si elle souhaite récupérer tous les détails de l'entité sauvegardée .*

- combiner les 2 styles de réponses (champ Location ET réponse JSON)

Lorsqu'un serveur répond à une requête en mode DELETE , il peut soit :

- se contenter de renvoyer le "http status" **204/NO_CONTENT** et aucun message

- retourner le "https status" **200/0K** et une réponse JSON de type
{ "message" : "resource of id ... successfully deleted" }

Lorsqu'un serveur répond à une requête en mode PUT , il peut soit :

- se contenter de renvoyer le "http status" **204/NO_CONTENT** et aucun message

*- retourner le "https status" **200/0K** et une réponse JSON comportant toutes les valeurs de l'entité mise à jour du coté serveur , exemple:*
{ "id" : "a345b6788c335d56" , "name" : "titi" , ... }

*On peut éventuellement envisager que le serveur réponde **par défaut** aux modes PUT et DELETE par un simple **204/NO_CONTENT** et qu'il réponde par **200/OK + un message JSON** si le paramètre http optionnel **?v=true** ou **?verbose=true** est présent en fin de l'URL de la requête .*

Identifiant de la resource à modifier en mode PUT placé en fin d'URL ou bien dans le corps de la requête HTTP, ou bien les deux.

Lorsqu'un serveur reçoit une requête de mise à jour en mode PUT , l'id de l'entity peut soit être précisée en fin d'URL , soit être précisée dans les données json de la partie body et si l'information est renseignée des 2 façons elle ne doit pas être incohérente .

Le serveur peut éventuellement faire l'effort de récupérer l'id de l'une ou des deux façons envisageables et peut renvoyer ***400/BAD_REQUEST*** si l'id de l'entité à mettre à jour n'est pas renseigné ou bien incohérent.

1.3. Safe and idempotent REST API

Une Api "Rest" désigne un ensemble de Web-services liés à un certain domaine fonctionnel (ex : gestion des stocks ou facturation ou ...)

Un appel "HTTP" vers une api-rest est dit "***safe***" s'il n'engendre pas de modifications du coté des ressources du serveur (**"safe" = "readonly"**) .

En mathématique , une fonction est dite "idempotente" si plusieurs appels successifs avec les mêmes paramètres retournent toujours le même résultat.

Au niveau d'une **api-rest** , une **invocation HTTP** (***ex : GET , PUT ou DELETE***) est dite "**idempotente**" si **plusieurs appels successifs avec les mêmes paramètres engendrent un même "état résultat" au niveau du serveur** .

Mais la réponse HTTP peut cependant varier .

Exemple : premier appel à "delete xyz/567" --> return "200/OK" ou "204/NO_CONTENT"

et second appel à "delete xyz/567"--> return 404 / notFound

mais dans les 2 cas , la ressource de type "xyz" et d'id=567 est censée ne plus exister .

Le DELETE est donc généralement considéré comme idempotent .

	safe	idempotent
GET *(et HEAD,OPTIONS)*	y	y
PUT	n	y
DELETE	n	y
POST	n	n

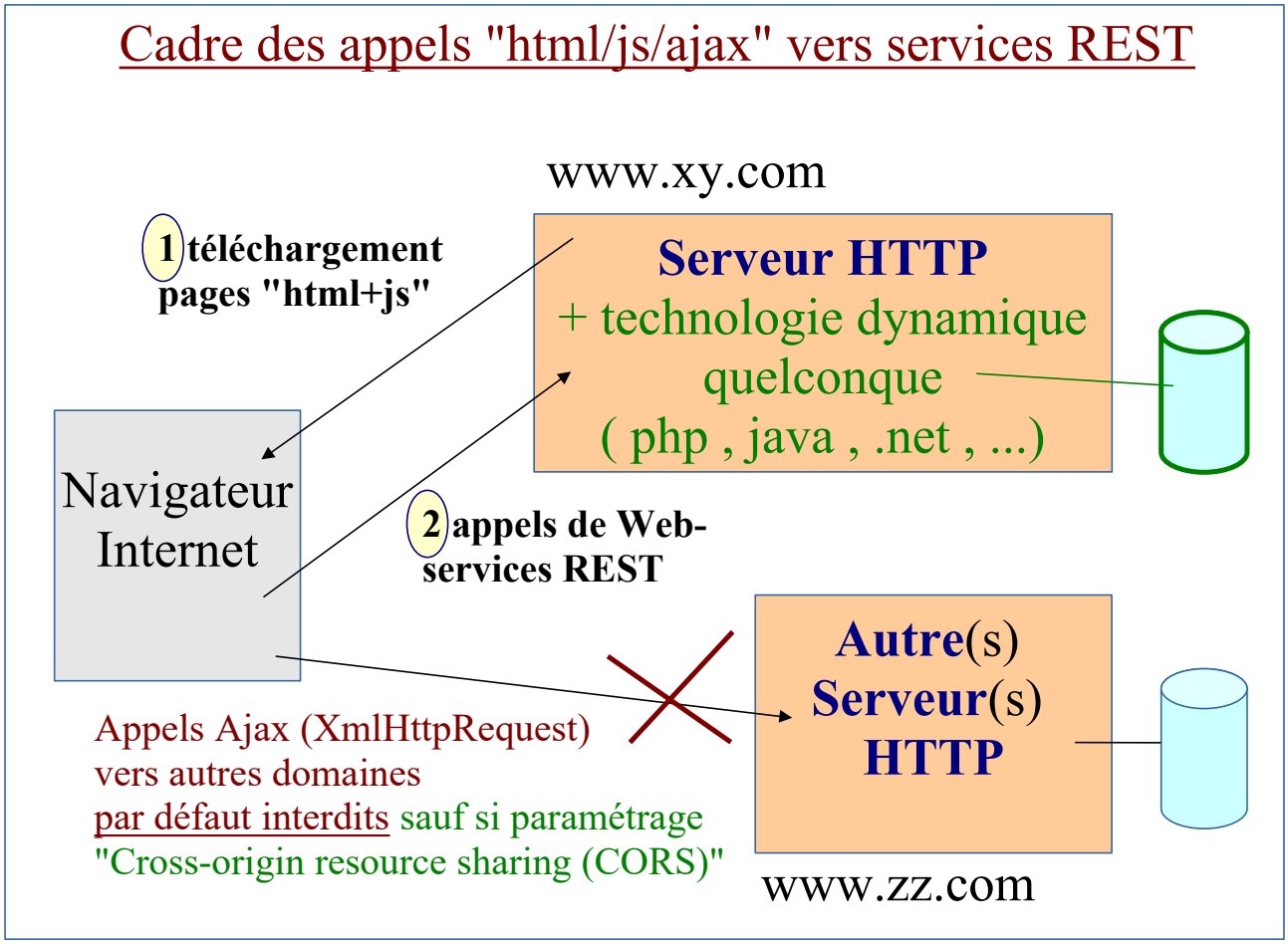
Intérêt de l'impotence comportementale du coté serveur :

Une application cliente doit souvent passer par des intermédiaires pour véhiculer une requête HTTP jusqu'au serveur . Certains mécanismes intermédiaires considèrent "internet / http" comme pas fiable à 100 % et vont quelquefois effectuer plusieurs retransmissions d'une requête si la première tentative échoue . il vaut mieux donc que le serveur se comporte de manière idempotente dans un maximum de cas .

Bien que le vocabulaire "~~idempotence~~" ne soit pas du tout approprié , **il est tout de même conseillé de retourner des réponses HTTP dans un format assez homogène vers le client** pour que celui-ci soit simple à programmer (pas trop de if ... else ...)

Dans tous les cas , bien documenter "comportements & réponses" d'une apit rest .

2. Limitations Ajax sans CORS



3. CORS (Cross Origin Resource Sharing)

CORS=Cross Origin Resource Sharing

CORS est une **norme du W3C** qui précise certains **champs** à placer dans une **entête HTTP** qui serviront à échanger entre le navigateur et le serveur des informations qui serviront à décider si une requête sera ou pas acceptée.

(utile si domaines différents) , dans requête simple ou bien dans pré-échange préliminaire quelquefois déclenché en plus :

Au sein d'une requête "demande autorisation" envoyée du client vers le serveur :

Origin: http://www.xy.com

Dans la "réponse à demande d'autorisation" renvoyée par le serveur :

Access-Control-Allow-Origin: http://www.xy.com
Ou bien
Access-Control-Allow-Origin: * *(si public)*
→ *requête acceptée*

Si absence de "Access-Control-Allow-Origin :" ou bien valeur différente
---> requête refusée

CORS=Cross Origin Resource Sharing (2)

NB1: toute requête "CORS" valide doit absolument comporter le champ "***Origin :***" dans l'entête http. Ce champ est toujours construit automatiquement par le navigateur et jamais renseigné par programmation javascript.
Ceci ne protège que partiellement l'accès à certains serveurs car un "méchant hacker" utilise un "navigateur trafiqué".
Les mécanismes "CORS" protège un peu le client ordinaire (utilisant un vrai navigateur) que dans la mesure ou la page d'origine n'a pas été interceptée ni trafiquée (l'utilisation conjointe de "https" est primordiale) .

NB2 : Dans le cas (très classique/fréquent) , où la requête comporte "***Content-Type: application/json***" (ou ***application/xml*** ou …) , la norme "CORS" (considérant la requête comme étant "pas si simple") impose un **pré-échange préliminaire** appelé "**Preflighted request/response**" .

Paramétrages CORS à effectuer coté serveur

L'application qui coté serveur, fourni quelques Web Services REST , peut (et généralement doit) autoriser les requêtes "Ajax / CORS" issues d'autres domaines ("*" ou "www.xy.com").
Attention: ce n'est pas une "sécurité coté serveur" mais juste **un paramétrage autorisant ou pas à rendre service à d'autres domaines et en devant gérer la charge induite** *(taille du cluster, consommation électrique, ...)* .

```
// Exemple : CORS enabled with express/node-js :
app.use(function(req, res, next) {
  res.header("Access-Control-Allow-Origin", "*"); //"*" ou "xy.com , ..."
  res.header("Access-Control-Allow-Methods",
            "POST, GET, PUT, DELETE, OPTIONS"); //default: GET, ...
  res.header("Access-Control-Allow-Headers",
     "Origin, X-Requested-With, Content-Type, Accept , Authorization");
  next();
});
```

Paramétrage "CORS" avec Spring-mvc

```
import org.springframework.web.bind.annotation.CrossOrigin;
...
@RestController
@CrossOrigin(origins = "*")
//@CrossOrigin(origins = { "http://localhost:4200" ,
//                         "http://www.partenaire-particulier.com" })
@RequestMapping(value="/rest/products" , headers="Accept=application/json")
public class ProductCtrl {...
}
```

et

```
@Configuration
public class WebSecurityConfig extends WebSecurityConfigurerAdapter {
...
  @Override
  protected void configure(final HttpSecurity http) throws Exception {
        http.authorizeRequests()
            .antMatchers("/",    "/favicon.ico",      "/**/*.png",
                    "/**/*.gif",      "/**/*.svg",  "/**/*.jpg",
                            "/**/*.html",      "/**/*.css",     "/**/*.js").permitAll()
            .antMatchers("/devise-api/public/**").permitAll()
            .antMatchers("/devise-api/private/**").authenticated()
             .and().cors() //enable CORS (avec @CrossOrigin sur class @RestController)
            .and().csrf().disable()
            .exceptionHandling().authenticationEntryPoint(unauthorizedHandler)
            .and()
            .sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS)
            .and()
            .addFilterBefore(jwtAuthenticationFilter,
                            UsernamePasswordAuthenticationFilter.class);
        }
...
}
```

XVI - Annexe – Sécurité - WEB Services "REST"

1. Api Key

Un web service hébergé par une entreprise et rendu accessible sur internet a un certain coût de fonctionnement (courant électrique , serveurs , ...) .

Pour limiter des abus (ex : appel en boucle) ou bien pour obtenir un paiement en contre partie d'une bonne qualité de service , un web service public est souvent invocable que si l'on renseigne une "api_key" (au niveau de l'URL ou bien au niveau de l'entête la requête HTTP).

Une "**api_key**" est très souvent de type "**uuid/guid**" .

Critères d'une api_key :

- lié à un abonnement (gratuit ou payant) , ex : compte utilisateur / compte d'entreprise
- ne doit idéalement pas être diffusé (à garder secret)
- souvent lié à un compteur d'invocations (limite selon prix d'abonnement)
- doit pouvoir être administré (régénéré si perdu/volé , ...)
  et les modifications doivent pouvoir être immédiatement ou rapidement prises en compte.

Exemple :

Le site **https://fixer.io** héberge un web service REST permettant de récupérer les taux de change (valeurs de *"USD"* , *"GBP"* , *"JPY"* , ... vis à vis de "*EUR*" par défaut).

Début 2018, ce web service était directement invocable sans "api_key" .

Courant 2018, ce web service est maintenant invocable qu'avec une "**api_key**" **liée à un compte utilisateur** *"gratuit" ou bien "payant"* selon le mode d'abonnement (options, fréquence d'invocation, ...).

URL d'appel sans "api_key" : **http://data.fixer.io/api/latest**

Réponse :

```
{
"success":false,
"error":{"code":101,"type":"missing_access_key",
        "info":"You have not supplied an API Access Key. [Required format:
                access_key=YOUR_ACCESS_KEY]"
    }
```

```
}
```

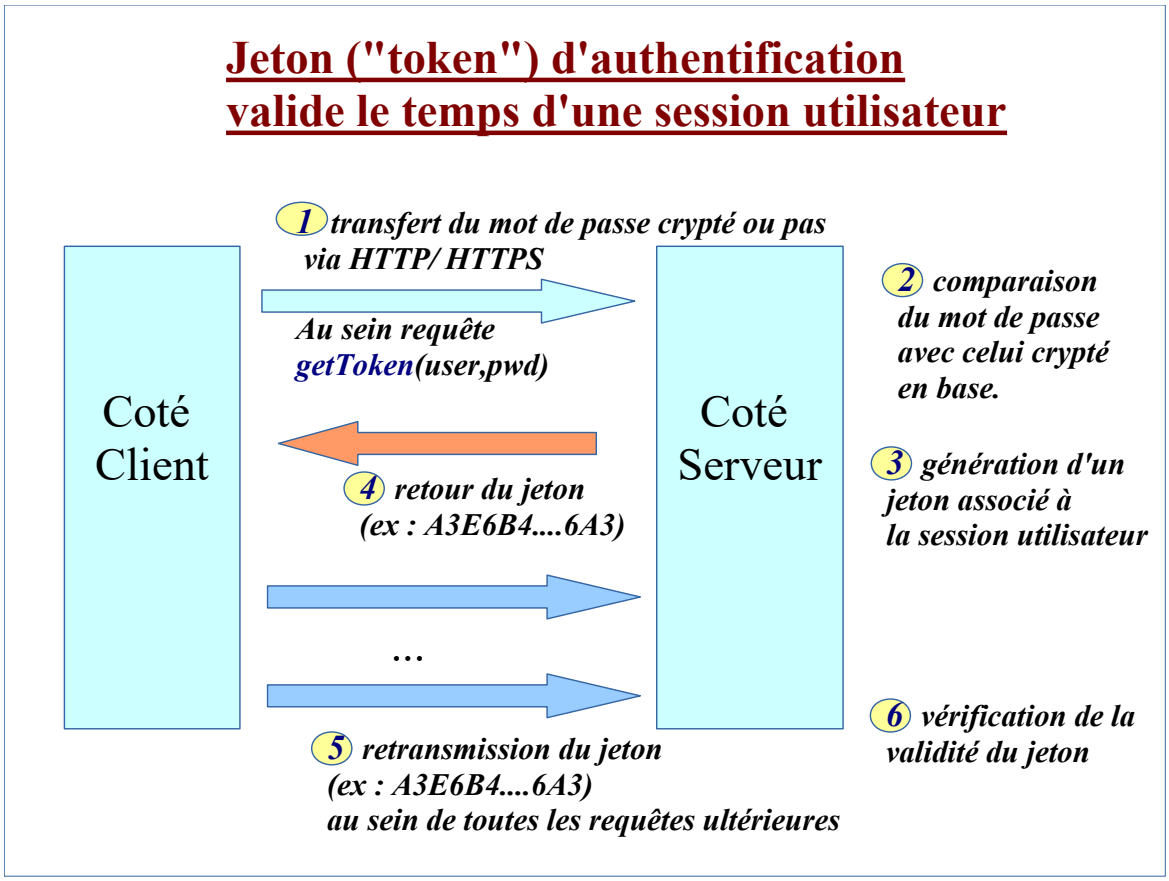
URL d'invocation avec api_key valide :

http://data.fixer.io/api/latest?access_key=*26ca93ee7.....aaa27cab235*


```
{
"success":true, "timestamp":1538984646, "base":"EUR", "date":"2018-10-08",
"rates":
{"AED":4.224369,...,"DKK":7.460075,"DOP":57.311592,"DZD":136.091172,"EGP":20.596249,
"ERN":17.250477,"ETB":31.695652,"EUR":1,"FJD":2.46956,"FKP":0.88584,"GBP":0.879667,.
...,"JPY":130.858498,....,"USD":1.15005,...,"ZWL":370.724343}
}
```


2. Token d'authentification

2.1. Tokens : notions et principes



Plusieurs sortes de jetons/tokens

Il existe plusieurs sortes de jetons (normalisés ou pas).

Dans le cas le plus simple, un **jeton est généré aléatoirement** (ex : **uuid** ou …) et **sa validation consiste essentiellement à vérifier son existence** en tentant de le récupérer quelque part *(en mémoire ou en base)* et éventuellement à vérifier une date et heure d'expiration.

JWT (Json Web Token) est un **format particulier de jeton** qui **comporte 3 parties** (une entête technique , un paquet d'informations en clair (ex : username , email , expiration, …) au format JSON et une signature qui ne peut être vérifiée qu'avec la clef secrète de l'émetteur du jeton.

2.2. Bearer Token (au porteur) / normalisé HTTP

Bearer token (jeton au porteur) et transmission

Le champ ***Authorization:*** normalisé d'une entête d'une requête HTTP peut comporter une valeur de type ***Basic …*** ou bien ***Bearer …***

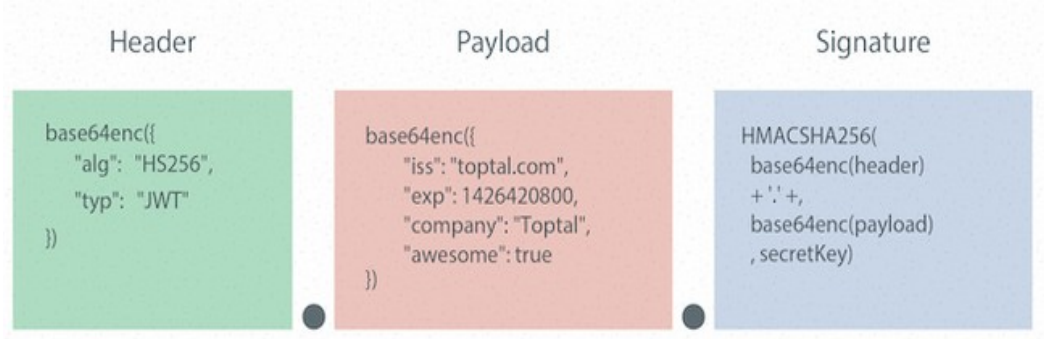
Le terme anglais "**Bearer**" signifiant "au **porteur**" en français indique que la simple possession d'un jeton valide par une application cliente devrait normalement , après transmission HTTP, permettre au serveur d'autoriser le traitement d'une requête (après vérification de l'existence du jeton véhiculé parmi l'ensemble de ceux préalablement générés et pas encore expirés).

NB: Les "bearer token" sont utilisés par le protocole "O2Auth" mais peuvent également être utilisés de façon simple sans "O2Auth" dans le cadre d'une authentification "sans tierce partie" pour API REST.

NB2 : un "bearer token" peut éventuellement être au format "JWT" mais ne l'est pas toujours (voir rarement) en fonction du contexte.

2.3. JWT (Json Web Token)

Structure jeton "JWT / Json Web Token"



Exemple:
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJ0b3B0YWwuY29tIiwiZXhwIjoxNDI2NDIwODAwLCJodHRwOi8vdG9wdGFsLmNvbS9qd3RfY2xhaW1zL2lzX2FkbWluIjp0cnVlLCJjb21wYW55IjoiVG9wdGFsIiwiYXdlc29tZSI6dHJ1ZX0.yRQYnWzskCZUxPwaQupWkiUzKELZ49eM7oWxAQK_ZXw

NB : "iss" signifie "issuer" (émetteur) , "iat" : issue at time
"exp" correspond à "date/heure expiration" . Le reste du "payload" est libre (au cas par cas) (ex : "company" et/ou "email" , ...)

XVII - Annexe – Bibliographie, Liens WEB + TP

1. Bibliographie et liens vers sites "internet"

https://npmtrends.com/	Comparaison du nombre de téléchargements
https://www.npmjs.com/	Référentiel officiel
https://nodejs.org/en/	Site officiel nodejs

2. TD et TP (quelques énoncés , propositions)

2.1. TD "premiers pas"

- Créer quelque part (ex : `c:\tp` ou ...) un nouveau répertoire `myNodeServer` qui deviendra un projet Node.js
- Dans une fenêtre CMD, se placer dans le répertoire vide `myNodeServer` et lancer la commande interactive `npm init`
  - Appuyer tout le temps sur `enter` pour valider les valeurs par défaut
  - Visualiser le fichier `package.json` créé
- Lancer la commande `npm install --save express`
- Visualiser le nouveau contenu du fichier `package.json` (effet de `--save`)
- Visualiser les éléments téléchargés dans `node_modules`
- Supprimer le sous-répertoire `node_modules` (et tout son contenu)
- Lancer la commande `npm install` et visualiser les éléments re-téléchargés dans `node_modules` (en fonction des dépendances exprimées dans le fichier `package.json`).
- Au sein du répertoire `myNodeServer`, coder le fichier `helloWorld.js` comme suit :

```
console.log("hello world");
```

- Depuis une invite de commande (CMD), lancer `node helloWorld.js` et visualiser le résultat attendu (pas très spectaculaire mais bon début).

2.2. Module "calcul.js" ou "calcul.mjs"

- Coder un module ***calcul.js*** ou ***calcul.mjs*** comportant les fonctions **carre(x)** et **racineCarree(x)** qu'il faudra exporter
- importer et appeler ces fonctions depuis le module principal (main.js ou server.js ou ...)

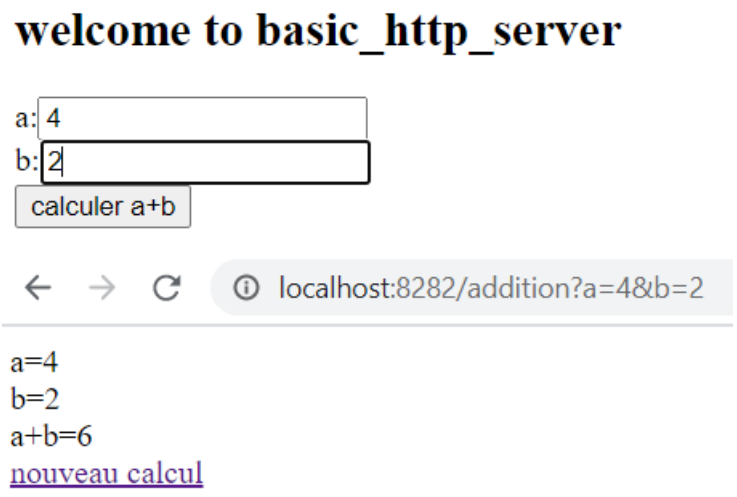
2.3. Micro-TP (bases de express)

- Créer dans `C:\tp-js` ou ailleurs un nouveau répertoire `myNodeServer`
- Se placer dans le répertoire `myNodeServer` (dans une console "CMD")
- Créer un début de `package.json` via `npm init`
- ajouter "type" : "module" dans package.json
- Installer le "package" Express via `npm install -s express`
- Au sein du répertoire `myNodeServer`, coder `basic_http_server.js` en s'inspirant de l'exemple `first_express_server.js` (page 16 environ du support de cours)
- Depuis une invite de commande (CMD), lancer `node basic_http_server.js` et saisir l'URL http://localhost:8282 dans un navigateur pour visualiser le résultat
- Modifier un peu le texte affiché dans `basic_http_server.js`
- Arrêter et relancer le server (`Ctrl-C` puis `node basic_http_server.js`) pour visualiser les changements
- Lancer la commande `npm install -g nodemon` de façon à installer (en mode global) l'extension nodemon.
- Arrêter le server (`Ctrl-C`) et relancer le via nodemon `nodemon basic_http_server.js`
- Modifier un peu le texte affiché dans `basic_http_server.js` et **visualiser directement** les changements dans le navigateur **dès que le fichier est sauvegardé**.

Petites améliorations facultatives (mais conseillées) :

- Faire en sorte que l'on puisse saisir les valeurs de `a` et `b` (à additionner) dans un formulaire HTML (en mode GET et avec `action="addition"`)
- Faire en sorte que l'on puisse ré-effectuer un autre calcul via un lien hypertexte
-

Résultats escomptés :



Le code suivant (`basic_http_server.js`) assure bien les fonctionnalités souhaitées :

```
//var express = require('express');
import express from 'express' ;
var app = express();

//route principale:
app.get('/', function(req, res , next) {
res.setHeader('Content-Type', 'text/html');
res.write("<html> <body>");
res.write('<h2>welcome to basic_http_server</h2>');
res.write('<!-- <a href="addition?a=5&b=6">5+6</a><br/> -->');
res.write("<form method='get' action='addition'>");
res.write("a:<input name='a'/><br/>");
res.write("b:<input name='b'/><br/>");
res.write("<input type='submit' value='calculer a+b'/>");
res.write("</form>");
res.write("</body></html>");
res.end();
});

//route pour addition?a=5&b=6
app.get('/addition', function(req, res , next) {
a = Number(req.query.a);b = Number(req.query.b);
resAdd = a+b;
res.setHeader('Content-Type', 'text/html');
res.write("<html> <body>");
res.write('a=' + a + '<br/>');res.write('b=' + b + '<br/>');
res.write('a+b=' + resAdd + '<br/>');
res.write('<a href="/">nouveau calcul</a>');
res.write("</body></html>");
res.end();
});

app.listen(8282 , function () {
console.log("http://localhost:8282");
});
```

2.4. Eventuel TP sur template-engine (handlebars , EJS ou pug)

1) utiliser express-generator pour générer un début de projet "***my-mvc-app***"
   en utilisant handlebar ou ejs ou … (page 41 ou ...)
2) lancer **npm install**
3) coder un traitement de type "**calcul de racine carrée**" (s'inspirer des pages 42,… 47)
4) tester

2.5. TP "Web service REST" sur Devise

En s'inspirant de l'exemple "***server.js***" + "***produit-api-routes-memory.js***" (près de la page 51)
Coder une api REST gérant des Devises monétaires .
"**server.js**" utilisera "***devise-api-routes-memory.js"***

Le fichier **`html/index.html`** servira à effectuer des tests en mode **GET**

```
<a href="../devise-api/public/devise"> toutes devises</a><br/>
<a href="../devise-api/public/devise?changeMini=1.05"> devises avec changeMini=1.05</a><br/>
```

```
<a href="../devise-api/public/devise/EUR"> devise EUR</a><br/>
```

Début de **devise-api-route-memory.js**

```
//var express = require('express');
import express from 'express';
const apiRouter = express.Router();

var allDevises = [];

allDevises.push({ code : 'EUR' , nom : 'Euro' , change : 1.0 });
allDevises.push({ code : 'USD' , nom : 'Dollar' , change : 1.1 });
allDevises.push({ code : 'JPY' , nom : 'Yen' , change : 123 });
allDevises.push({ code : 'GBP' , nom : 'Livre' , change : 0.9 });

function findDeviseInArrayByCode(devises,code){
	var devise=null;
	for(let i in devises){
		if(devises[i].code == code){
			devise=devises[i]; break;
		}
	}
	return devise;
}

function removeDeviseInArrayByCode(devises,code){
	var delIndex;
	for(let i in devises){
		if(devises[i].code == code){
			delIndex=i; break;
		}
	}
	if(delIndex){
		devises.splice(delIndex,1);
	}
}

function findDevisesWithChangeMini(devises,changeMini){
	var selDevises=[];
	for(let i in devises){
		if(devises[i].change >= changeMini){
			selDevises.push(devises[i]);
		}
	}
	return selDevises;
}
/*
points d'entrées souhaitées :
http://localhost:8282/devise-api/public/devise/EUR

http://localhost:8282/devise-api/public/devise (returning all devises)
http://localhost:8282/devise-api/public/devise?changeMini=1.05

http://localhost:8282/devise-api/private/role-admin/devise en mode post
```

```
avec { "code" : "mxy" , "nom" : "monnaieXy" , "change" : 123 } dans req.body

http://localhost:8282/devise-api/private/role-admin/devise en mode PUT
avec { "code" : "USD" , "nom" : "Dollar" , "change" : 1.123 } dans req.body

http://localhost:8282/devise-api/private/role-admin/devise/EUR en mode DELETE
*/
```

Tester avec Postman les modes POST, PUT, DELETE

2.6. TP "Middleware" d'authentification élémentaire

req.headers.xyz pour récupérer dans entête http et *req.query.xyz* pour récupérer param url
---
Au sein de **server.js** , ajouter via **app.use()** un **middleware** qui aura le comportement suivant :
- récupérer la valeur d'un parametre "*x-username*" qui sera recherché à la fois en tant que paramètre url et en tant que champ de l'entête HTTP (on gardera la valeur non nulle trouvée)
- récupérer la valeur d'un parametre "*x-password*" qui sera recherché à la fois en tant que paramètre url et en tant que champ de l'entête HTTP (on gardera la valeur non nulle trouvée)
- récupérer la valeur d'un parametre "*x-role*" qui sera recherché à la fois en tant que paramètre url et en tant que champ de l'entête HTTP (on gardera la valeur non nulle trouvée)
- ce middleware retournera une erreur 401
 si la valeur de *x-password n'est pas égale à "pwd' + x-username*
- ce middleware retournera une erreur 403
 si la valeur de *x-role* n'est pas égale à '*user*' ni '*admin*'
- ce middleware stockera les valeurs de *x-username* et *x-role* dans l'objet **req** (request)
 et appelera next() quand tout va bien

Tester (avec ou sans postman)

NB : sur un vrai projet d'entreprise , jamais de *x-password* en clair et en amont "analyse de jeton JWT ou autres"

2.7. TP "Accès base de données"

Restructurer le code de myNodeServer pour que les devises soient stockées dans une base de données (**mongoDB** ou **sqlLite** ou autre)

2.8. TP "DAO avec Promise puis async/await"

En partant d'un code de ce type (à adapter en fonction du type de base de données utilisé) :

//http://localhost:8282/devise-api/public/devise-conversion?montant=50&source=EUR&cible=USD

```
apiRouter.route('/devise-api/public/devise-conversion')
.get( function(req , res  , next ) {
        let montant = Number(req.query.montant);
        let codeDeviseSource = req.query.source;
        let codeDeviseCible = req.query.cible;
        //on demande à mongodb les détails de la devise source
        PersistentDeviseModel.findOne( { _id : codeDeviseSource} ,
```

```
                                function (err,deviseSource){
            if(err!=null || deviseSource==null )
              res.status(404).send({ message:"devise source pas trouvee"})      ;
            else
            //callback avec deviseSource si tout va bien
            //2 nd appel pour récupérer les détails de la devise cible:
            PersistentDeviseModel.findOne( { _id : codeDeviseCible} ,
                    function (err,deviseCible){
                    if(err!=null || deviseCible==null )
                            res.status(404).send({ message:"devise cible pas trouvee"})          ;
                    else {
                     //callback avec deviseCible si tout va bien
                     var montantConverti = montant * deviseCible.change / deviseSource.change;
                     res.send ( { montant : montant ,
                                  source :codeDeviseSource ,
                                  cible : codeDeviseCible ,
                                  montantConverti : montantConverti});
                     }
                    });
            });
})
```

1. Tester en premier une version adaptée du code si dessus avec des callbacks
2. Dupliquer la route précédente et en garder une version de souvenir en commentaire
3. Reprogrammer cette route en s'appuyant sur des appels de méthodes retournant des "Promise"
4. Tester la version "Promise"
5. Dupliquer la route "Promise" et en garder une version de souvenir en commentaire
6. Reprogrammer cette route en s'appuyant sur async/await
7. Tester la version "async/await"

Solutions possibles (à adapter selon type de base de données utilisé) :

```
/*
//valeur en retour Promise<Devise>
function getDeviseByCode(codeDevise){
 return new Promise ((resolve,reject)=> {
   PersistentDeviseModel.findById( codeDevise ,
    function(err,devise){
    if(err || devise==null)
        reject({ err : 'not found'});
    else
        resolve(devise);
   });
 });
```

```
}
*/
```

```
/*
//exemple URL: http://localhost:8282/devise-api/public/devise-conversion?montant=50&source=EUR&cible=USD
apiRouter.route('/devise-api/public/devise-conversion')
.get( function(req , res  , next ) {
        let montant = Number(req.query.montant);
        let codeDeviseSource = req.query.source;
        let codeDeviseCible = req.query.cible;
        let deviseSourceLocal = null;
        //on demande à mongodb les détails des devises source et cible
        //PersistentDeviseModel.findOne( { _id : codeDeviseSource} )
        devise_dao_mongoose.getDeviseByCode(codeDeviseSource)
          .then((deviseSource)=>{deviseSourceLocal = deviseSource;
                  //return PersistentDeviseModel.findOne( { _id : codeDeviseCible} )
                return devise_dao_mongoose.getDeviseByCode(codeDeviseCible)})
          .then((deviseCible)=>{ let montantConverti = montant * deviseCible.change / deviseSourceLocal.change;
                  res.send ( { montant : montant ,
                                          source :codeDeviseSource ,
                                          cible : codeDeviseCible ,
                                          montantConverti : montantConverti});
          })
          .catch((erreur)=>{ res.status(404).send({ message:"devise toujours pas trouvee"})      ;});
})
*/
```

```
/*
//exemple URL: http://localhost:8282/devise-api/public/devise-conversion?montant=50&source=EUR&cible=USD
apiRouter.route('/devise-api/public/devise-conversion')
.get( async function(req , res  , next ) {
        let montant = Number(req.query.montant);
        let codeDeviseSource = req.query.source;
        let codeDeviseCible = req.query.cible;
        //on demande à mongodb les détails des devises source et cible
   try{
      //appels via await au sein d'une fonction préfixée par async
                  //const deviseSource = await PersistentDeviseModel.findOne( { _id : codeDeviseSource} );
                  //const deviseSource = await devise_dao_mongoose.getDeviseByCode(codeDeviseSource);
                  //const deviseCible =  await PersistentDeviseModel.findOne( { _id : codeDeviseCible});
                  //const deviseCible =  await devise_dao_mongoose.getDeviseByCode(codeDeviseCible);
```

```
                const [deviseSource,deviseCible]=await Promise.all([
                        devise_dao_mongoose.getDeviseByCode(codeDeviseSource) ,
                        devise_dao_mongoose.getDeviseByCode(codeDeviseCible)
                ]);
                let montantConverti = montant * deviseCible.change / deviseSource.change;
        res.send ( { montant : montant ,
                                source :codeDeviseSource ,
                                cible : codeDeviseCible ,
                                montantConverti : montantConverti});
    }catch(e){
            res.status(404).send({ message:"devise inconnue pas trouvee qui existe pas"})   ;
    }
});
*/
```

2.9. TP sur fileSystem

Lire un fichier **produits.csv**

```
#num;label;prix
1;cahier;4.6
2;stylo;2.3
3;classeur;5.7
```

Calculer la moyenne , le minimum et le maximum des prix

Générer en sortie un fichier **stats.json**

```
{ "min" : 2.3 , "max" : 5.7 , "average" : 4 }
```

2.10. TP sur webSockets / socket.io

Programmer une petite appli (hyper simplifiée) sur le thème "ventes aux enchéres" et affichage en temps réel de la "meilleur offre" .

2.11. Autres TD et TP

- Au moins un **test unitaire** avec **mocha + chai**
- ...