

# TD WEB DEV BACK END

Joseph AZAR – Oct 2022


## NodeJS

1. Quel est l'avantage d'utiliser node.js ?
2. Qu'est-ce que REPL dans Node.js ?
3. Est-ce que "console" est un objet global ?
4. Quelles sont les principales fonctionnalités fournies par npm? (citez 2)
5. En quoi NodeJS est-il différent des autres frameworks JavaScript (Angular, React, Vue) ?
6. Regardez cette vidéo sur la boucle d'événements NodeJs :  
[https://www.youtube.com/watch?v=g25LIAllcbE&t=3s&ab\\_channel=Louisititi](https://www.youtube.com/watch?v=g25LIAllcbE&t=3s&ab_channel=Louisititi)  
La boucle d'événements peut-elle être bloquée ? si oui comment ?
7. "non bloquant signifie que rien sur un serveur Node n'est bloqué par quoi que ce soit d'autre."  
La déclaration ci-dessus est-elle correcte ? Élaborer.
8. Pour chaque cas d'utilisation ci-dessous, expliquez si vous utiliseriez NodeJS ou non:
  - a. J'ai besoin de créer une application de création de rapports. Le backend devait prendre quelques Go de données brutes, structurer les données en groupes, élaborer des statistiques basées sur les groupes et renvoyer des informations pour générer des graphiques de tendance.
  - b. J'ai besoin de créer une application de chat en temps réel.
  - c. J'ai besoin de créer un outil de collaboration en temps réel. Les outils de collaboration (collaboration tools) sont des applications qui mettent l'accent sur la gestion des projets et des tâches. Ex: Trello, Airtable, Asana, Basecamp, Wrike et Monday.
  - d. J'ai besoin de créer des applications monopage complexes (Single Page Application ⇒ SPA). Dans les SPA, l'intégralité de l'application tient essentiellement sur une seule page pour fournir une expérience similaire à celle d'une application de bureau.
  - e. J'ai besoin de créer une application Web qui classe les images téléchargées par l'utilisateur à l'aide de l'intelligence artificielle. L'utilisateur télécharge une image d'un chat ou d'un chien et l'application doit répondre par (chat, chien, ni l'un ni l'autre)
  - f. J'ai besoin de construire une API REST.
9. Expliquez la différence entre l'installation des packages npm locaux et globaux (local & global npm installation).
10. Que sont libuv et V8.
11. Expliquez comment fonctionne Node.js ?
12. Qu'est-ce qu'un code bloquant dans Node.js ?
13. Pourquoi Node.js est-il monothread (single threaded)?
14. Combien de types de fonctions d'API existe-t-il dans Node.js ?
15. Qu'est-ce que package.json ?
16. Donnez un exemple de la différence entre CommonJS et ESM.

17. La fonction require est synchrone ou asynchrone ?
18. Qu'entendez-vous par programmation événementielle ?
19. Que sont les versions LTS de Node.js ?
20. Quelle est la relation entre Node.js et V8 ?
21. Comment les modules Node.js sont disponibles en externe ?
22. Quelle est la différence entre require(x) et import x (ES6) dans NodeJS ?
23. Qu'est-ce que "export default" en NodeJS ?
24. Qu'entendez-vous par objets globaux dans Node.js ? Pouvez-vous donner quelques exemples.
25. Quelle est la différence entre dependencies, devDependencies et peerDependencies dans le fichier npm package.json ?

# Javascript Asynchrone

1. Quelle est la sortie de ce code ?



```
setTimeout(function() {
  console.log('Test 0');
});

console.log('Test 1');

for (let i = 0; i < 10000; ++i)
{
  doSomeStuff();
}

console.log('Test 2');

function doSomeStuff() {
  return 1 + 1;
}
```

2. Quelle est la différence entre setInterval et setTimeout ?
3. "setTimeout et setInterval sont les seules fonctions natives du JavaScript pour exécuter du code de manière asynchrone."  
La déclaration ci-dessus est-elle correcte ?
4. Pourquoi avons-nous besoin d'une exécution asynchrone ?
5. Quelle est la sortie de ce code ?



```
let fs = require('fs');
console.log('A');
fs.readFile('test.txt', 'utf8', function(error, data) {
  if (error) {
    throw error;
  }

  console.log('B');
});
console.log('C');
```

6. Qu'est-ce que le rappel (Callback)?
7. Quelle est la fonction pour arrêter un compteur d'intervalles (interval timer)?
8. L'extrait de code ci-dessous est synchrone. Il appelle callbackA une fois et callbackB trois fois. Vous devez modifier ce code pour le rendre asynchrone avec les règles suivantes :

callbackA ne doit être appelé qu'une seule fois, après 4 secondes.  
callbackB doit être appelé trois fois avec un intervalle de 2 secondes.

```
function myToDoTask(callbackA, callbackB) {
  callbackA();

  callbackB();
  callbackB();
  callbackB();
}

module.exports = myToDoTask;
```

9. "Le code asynchrone repose sur la fourniture de rappels (callbacks)."

La déclaration ci-dessus est-elle correcte ?

10. Considérez le code suivant. Appelez func3 après func1 et func2 après func3.

```
function func1(callback) {
  setTimeout(function() {
    callback('FUNC 1');
  }, 2000);
}

function func2(callback) {
  setTimeout(function() {
    callback('FUNC 2');
  }, 4000);
}

function func3(callback) {
  setTimeout(function() {
    callback('FUNC 3');
  }, 1000);
}
```

11. Considérez le code suivant. Nous voulons appeler la fonction "terminer" après l'exécution de func1 et func2. Quelles modifications devez-vous ajouter au code pour ce faire ?

```

func1(function() {
  console.log("FUNC 1");
});

func2(function() {
  console.log("FUNC 2");
})

function terminer() {
  console.log('FUNC 1 et FUNC 2 sont finies d'être exécutées');
}

```

12. Quel est le nom du problème que vous pouvez rencontrer lors de l'utilisation des rappels comme dans les questions 10 et 11 ? Comment pouvez-vous résoudre ce problème ?
13. Utilisez la classe Promise pour convertir le code asynchrone ci-dessous de basé sur le rappel (callback-based) en basé sur des promesses (promises-based).

```

function func1(callback){
  setTimeout(function() {
    let i = Math.random()
    if (i<=0.5){
      callback(null, 'SUCCESS');
    }else{
      callback('ERROR')
    }
  }, 2000);
}

func1((error,result)=>{
  if (error) {
    console.log(error)
  }else{
    console.log(result)
  }
});

```

14. Refaire l'exercice 10 en utilisant le chaînage de promesses (Promises chaining).  
Remarque: enchaîner les promesses est la raison même pour laquelle nous avons des promesses en premier lieu. C'est un bon moyen d'indiquer à JavaScript la prochaine chose à faire après la fin d'une tâche asynchrone

15. Qu'est-ce qui ne va pas dans le code suivant, et comment le corriger ?

```
function monPromesse() {
  var promise = func1();

  promise.then(function(data) {
    console.log(data);
  });

  return promise;
}
```

16. Quelle est la sortie du code ci-dessous ? Pourquoi pensez-vous que nous avons obtenu cette sortie ?

```
function func() {
  return new Promise(function(resolve, reject) {
    reject();
  });
}

let promise = func();

promise
  .then(function() {
    console.log('Success A');
  })
  .then(function() {
    console.log('Success B');
  })
  .then(function() {
    console.log('Success C');
  })
  .catch(function() {
    console.log('Error A');
  })
  .then(function() {
    console.log('Success D');
  });
```

17. Parfois, vous avez plusieurs tâches asynchrones à effectuer et vous devez démarrer quelque chose lorsque chaque tâche est terminée. Lorsque vous utilisez des promesses, vous pouvez le faire avec Promise.all. Essayez de résoudre l'exercice 11 en utilisant Promise.all.

18. Quel est l'avantage d'utiliser async/await par rapport aux promesses ?

19. Créez une fonction qui appelle myFunc en utilisant async/await.



```
function myFunc() {  
  return new Promise(function(resolve, reject) {  
    setTimeout(resolve, 500, 'SALUT');  
  });  
}
```

20. Comment l'erreur est-elle gérée lors de l'utilisation de async/await ?

21. Quelle est la sortie de ce code ? pourquoi pensez-vous que nous avons obtenu cette sortie ?



```
async function myFunc() {  
  return 'SALUT';  
}  
  
console.log(await myFunc());
```

# Backend & REST API

## Partiel 2021

Les questions ci-dessous sont posées lors de l'examen de 2021.

### Questions sur le backend

- Répondez par Vrai ou Faux.
  - REST ==> REsources State Transfer: \_\_\_\_\_
  - REST a un URI pour accéder aux ressources au moyen d'un modèle requête-réponse: \_\_\_\_\_
  - REST est un modèle architectural pour le développement de systèmes de communication client-serveur: \_\_\_\_\_
  - REST prend en charge les requêtes asynchrones, éliminant ainsi la nécessité d'une interaction client-serveur constante: \_\_\_\_\_
- Pouvez-vous citer ce qui constitue les composants de base d'une réponse HTTP?

- Nous avons une table "clients" et nous souhaitons créer les URL des différentes opérations pour notre API. Remplissez le tableau ci-dessous:

Opération	Méthode HTTP	URL
sélectionner un client en utilisant son nom d'utilisateur	GET	/clients/{username}
Insérer un nouvel client		
Sélectionnez tous les clients		
Sélectionnez un client à l'aide de son adresse e-mail		
Supprimer un client		

- Quelle réponse obtenons-nous lorsque nous envoyons cette requête:

<http://localhost:3000/api/clients/projects?project=12>

```
app.get('/api/clients', (req, res) => {res.send("A")});
app.post('/api/clients/projects', (req, res) => {res.send("B")});
app.get('/api/clients/:id', (req, res) => {res.send("C")});
app.get('/api/clients/projects', (req, res) => {res.send("D")});
```

- Quel est le nom du modèle de conception suivant et quel est le rôle des fichiers dans chaque dossier?

```
├── models
│   └── user.model.js
├── routes
│   └── user.route.js
├── services
│   └── user.service.js
└── controllers
    └── user.controller.js
```

Nom du modèle de conception (design pattern name): .....

models	
routes	
services	
controllers	



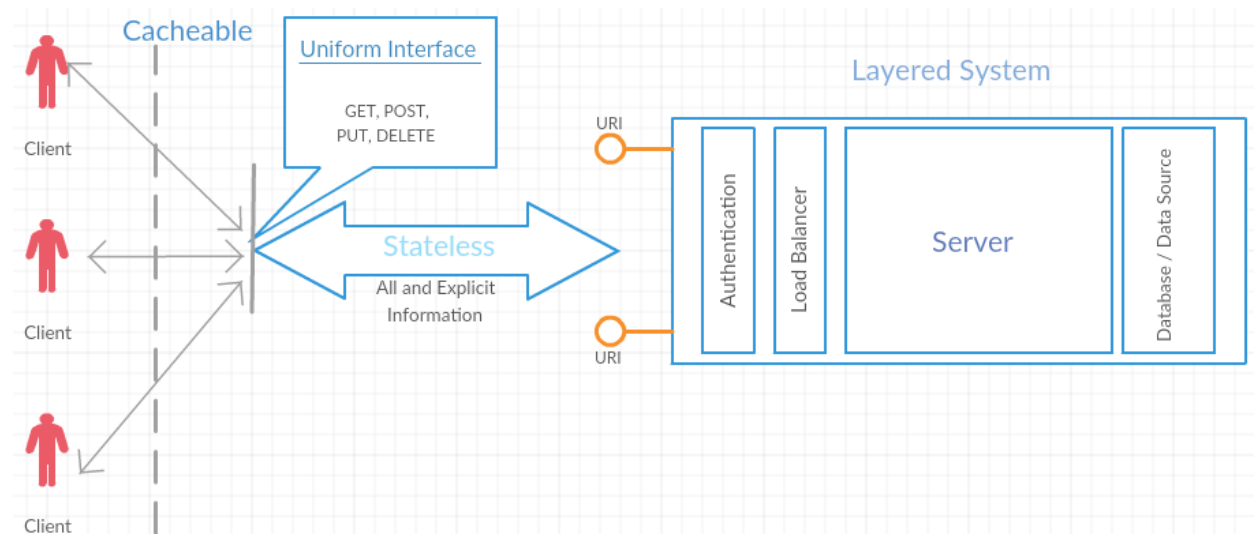
6. "L'API Web est une interface de programmation d'application (API) pour un serveur Web ou pour un navigateur Web. Ainsi, l'API Web est un concept ou une méthodologie permettant d'accéder à n'importe quelle API (disponible sur le Web) via le protocole HTTP. Il existe de nombreuses catégories d'API, SOAP, XML-RPC, JSON-RPC, REST, etc."

La déclaration ci-dessus est-elle correcte ?

7. "Les concepts REST ont été soumis sous forme de thèse de doctorat par Roy Fielding. Le principe fondamental de REST est d'utiliser le protocole HTTP pour la communication de données (entre systèmes hypermédias distribués), et il s'articule autour du concept de ressources où chaque composant considéré comme une ressource, et ces ressources sont accessibles par les interfaces communes à l'aide de méthodes HTTP"

La déclaration ci-dessus est-elle correcte ?

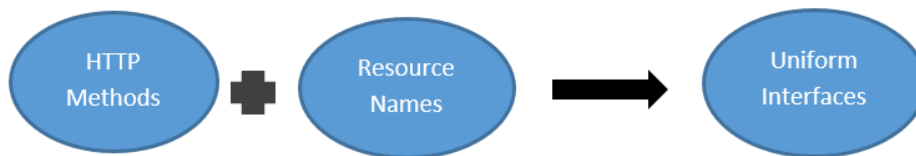
8. Le schéma suivant illustre les contraintes REST dans une application Web/Internet typique. Quelles sont les contraintes REST que vous pouvez extraire de ce diagramme et expliquer chacune d'entre elles.



9. La contrainte "statelessness", dans le contexte REST, signifie que toutes les requêtes client au serveur portent toutes les informations comme explicites (énoncées), de sorte que le serveur comprend les requêtes, les traite comme indépendantes, et ces requêtes client maintiennent le serveur indépendant de tout stockage contextes. La contrainte "statelessness" impose des restrictions importantes sur le type de communications autorisées entre les services et les consommateurs, pour atteindre ses objectifs de conception. Faites une petite recherche sur le Web et découvrez quels sont les avantages et les inconvénients de "statelessness" dans la conception REST.

10. Il existe quatre principes directeurs suggérés par Roy Fielding qui constituent les contraintes nécessaires pour satisfaire l'interface uniforme, et ils sont les suivants :

- Identification des ressources (Identification of resources)
- Manipulation des ressources (Manipulation of resources)
- Messages autodescriptifs (Self-descriptive messages)
- L'hypermédia comme moteur d'état applicatif (Hypermedia as the engine of application state)



Pouvez-vous décrire chacun de ces 4 points ?

11. "Hypermedia as the Engine of Application State (HATEOAS) est l'une des contraintes les plus critiques ; sans y répondre, les services ne peuvent pas être qualifiés de services RESTful."

La déclaration ci-dessus est-elle correcte ?

12. Le diagramme suivant illustre un exemple de réponse JSON d'un serveur sans HATEOAS. Mettez à jour la réponse afin que le serveur prenne en compte la contrainte HATEOAS.

### SANS HATEOAS

```
1 Request:
2 [Headers]
3 user: jim
4 roles: USER
5 GET: /items/1234
6 Response:
7 HTTP 1.1 200
8 {
9   "id" : 1234,
10  "description" : "FooBar TV",
11  "image" : "fooBarTv.jpg",
12  "price" : 50.00,
13  "owner" : "jim"
14 }
```

13. Quels sont les objectifs architecturaux de REST?

14. Expliquer le concept de middleware dans Node.js ?

15. `app.use(bodyParser.json());` est utilisé pour prendre en charge l'encodage JSON et `app.use(bodyParser.urlencoded({ extended: true }));` est utilisé pour prendre en charge les URL encodées telles que `content-type: application/x-www-form-urlencoded`.

La déclaration ci-dessus est-elle correcte ?

16. Quels sont les arguments disponibles pour une fonction de gestionnaire de route (route handler) Express JS ?

17. Voici les principaux types de middleware :

- Application-level Middleware
- Router-level Middleware
- Error-handling Middleware
- Built-in Middleware
- Third-party Middleware

Pouvez-vous donner un exemple pour chacun ?

18. Que sont les moteurs de template (template engines)? pouvez-vous donner deux exemples?

19. Comment rendre un HTML simple dans Express ? (citez deux approches)

20. Existe-t-il une différence entre `res.send` et `return res.send` dans Express.js ?