

## Remerciements

En premier lieu, je tiens à remercier mon maître de stage, M. Philippe CORNICHET. Un grand merci pour son accueil chaleureux au sein de l'entreprise Harmonic, ainsi que pour sa patience et ses précieux conseils. Il a su me faire confiance lors de cette aventure dans le monde professionnel et a partagé ses connaissances de manière très pédagogique. Je le remercie aussi pour sa disponibilité et la qualité de son encadrement en entreprise.

Je tiens également à remercier vivement mes supports techniques M Gildas CANCOUET, M Bruno DERRIEN et M Jean-Luc LEBLANC pour leur enthousiasme et le partage de leur expertise au quotidien. Ils m'ont donné les conseils les plus importants au niveau technique ainsi que fonctionnel dans les moments les plus difficiles. Ils m'ont aussi montré mes points faibles à améliorer et m'ont orienté vers les technologies utilisées dans l'industrie.

Je saisir également cette occasion pour adresser mes remerciements au corps professoral et administratif de l'Institut National des Sciences Appliquées Centre Val de Loire, pour la qualité de l'enseignement offert, le soutien de l'équipe administrative et pour m'avoir donné une opportunité de faire un stage de 4 mois qui m'a aidé à acquérir plus d'expériences dans ma future profession.

Je désire aussi remercier les professeurs de l'INSA Centre Val de Loire, qui m'ont fourni les outils nécessaires au bon déroulement de mon stage. Je tiens particulièrement à remercier M. Maxime BAVENCOFFE, pour son soutien tout au long de ce stage.

Un grand merci à toute l'équipe de l'entreprise Harmonic, pour l'intérêt qu'ils m'ont porté, l'esprit du travail en équipe et en particulier Mlle Cinderella DUBOIS pour sa gentillesse et son aide pour la partie administrative.

Enfin, je remercie aussi toutes les personnes qui m'ont conseillé et relu lors de la rédaction de ce rapport de stage.

## Résumé

Dans le cadre de mes études d'ingénieur en informatique à l'INSA Centre Val de Loire, ce stage s'est effectué au sein de la société Harmonic entre le mois de mai et août 2021. L'objectif du stage était de réaliser des missions techniques pendant une période significative, afin d'appliquer toutes les connaissances et compétences acquises au cours de mes études.

Ma mission de 4 mois a été divisée en plusieurs parties principales concernant le développement une interface Restful Api pour le VIBE CP9000, un encodeur utilisé par les Broadcaster pour la transmission d'événements sportifs, avec le but final de pouvoir intégrer au management system.

D'abord, j'ai développé un Restful API en m'inspirant d'autres API existantes utilisant NodeJs, une plateforme logicielle en JavaScript qui permet d'exécuter des scripts côté serveur. Ensuite, j'étais responsable de développer un outil de gestion des identifiant. En fin, il faut l'intégrer au cadre hérité et générer un document api pour le test de validation. Avec des étapes comme celle-là, je dois concevoir mon propre script pour les fonctions de l'api.

Le challenge de ce projet était de bien comprendre la structure de données et le fonctionnement de la machine CP9000 pour le mieux personnaliser l'api. Par manque du temps et des difficultés dans la période de confinement à cause du COVID 19, je n'ai pas pu réaliser toutes les idées. Pourtant, au final, je suis arrivé à fournir quelques solutions qui répondent aux exigences définies au début de mon stage.

**Mot clés :** Développement Back-end, REST API, JavaScript, Node.js, Native Addon Napi, C/C++, Session id, JWT, Oauth2.0, HTML, Cmake, HTTP/HTTPS, Linux, JIRA, Confluence

## Abstract

As part of my studies at INSA Center Val de Loire, this internship took place within the company Harmonic between May and August 2021. The objective of the internship was to carry out technical missions for a significant period, in order to apply all the knowledge and skills acquired during my studies.

My 4 month mission was divided into several main parts concerning the development of a Restful API interface for the VIBE CP9000, an encoder used by Broadcasters for the transmission of sporting events, with the final goal of being able to integrate into the management system.

First, I developed a Restfull API taking inspiration from other existing APIs using NodeJs, a JavaScript-based software platform that allows server-side scripting to be executed. Then, I was responsible for developing an identifier management tool. Finally, I have to integrate it into the legacy framework and generate an API document. With steps like this I have to design my own script for the API functions.

The challenge of this project was to understand the data structure and the operation of the CP9000 machine in order to customize the API. Due to lack of time and difficulties in the lockdown period due to COVID 19, I could not realize all the ideas. However, in the end, I managed to provide some solutions that meet the requirements defined at the start of my internship.

**Keywords :** Back-end development, REST API, JavaScript, Node.js, Native Addon Napi, C / C ++, Session id, JWT, Oauth2.0, HTML, Cmake, HTTP/HTTPS, Linux, JIRA, Confluence

## Table des matières

<b>Remerciements</b>	<b>1</b>
<b>Résumé</b>	<b>2</b>
<b>Abstract</b>	<b>2</b>
<b>Table des figures</b>	<b>5</b>
<b>Introduction</b>	<b>6</b>
<b>1 L'entreprise</b>	<b>7</b>
1.1 Présentation de l'entreprise . . . . .	7
1.2 Harmonic France site Rennes . . . . .	7
1.3 Les membres concernés . . . . .	8
<b>2 Contexte et description détaillé du projet</b>	<b>8</b>
2.1 Contexte du projet de stage . . . . .	8
2.2 Outils de travail . . . . .	9
2.3 Déroulement du stage . . . . .	10
<b>3 Détails techniques des solutions</b>	<b>11</b>
3.1 Développement web et les protocoles de communication . . . . .	11
3.1.1 Développement web back-end . . . . .	11
3.1.2 Aperçu rapide de l'IP (Internet Protocol) . . . . .	11
3.1.3 Introduction de API REST . . . . .	13
3.2 Développement API REST avec Node.js et Express.js . . . . .	14
3.2.1 NodeJS et Framework ExpressJS . . . . .	14
3.2.2 Le première démo . . . . .	15
3.3 C++ Addons . . . . .	17
3.3.1 Outil de compilation . . . . .	18
3.3.2 Demo des addons dans un NodeJs . . . . .	19
3.4 Gestion des identifiants . . . . .	21
3.4.1 User session . . . . .	22

3.4.2	OAuth2.0 . . . . .	24
3.4.3	JSON Web Token (JWT) . . . . .	26
3.5	La gestion des erreurs . . . . .	30
3.6	Gestion des droits d'utilisateur . . . . .	31
3.7	Intégration de API REST au framework héritage CP9000 . . . . .	32
3.7.1	La structure du CP9000 . . . . .	32
3.7.2	Manipulation des structures de donnée NCCP. . . . .	33
3.7.3	GET/SET adresse IP et port UDP source en SMPTE2022-6 . . . . .	34
3.8	Documentation Api . . . . .	35
<b>4</b>	<b>Conclusion</b>	<b>36</b>
<b>Références</b>		<b>36</b>

## Table des figures

1	Présence d'Harmonic sur le monde . . . . .	7
2	Les solutions technologies données par Harmonic . . . . .	8
3	Implémentation API dans CP9000 . . . . .	9
4	La fonctionnement du serveur proxy inverse . . . . .	12
5	Implémenter un serveur proxy inverse avec Nginx . . . . .	12
6	Exemple endpoints utilisant dans notre projet . . . . .	13
7	Le principe de l'architecture REST . . . . .	14
8	Event-Loop NodeJS . . . . .	14
9	La structure du démo . . . . .	15
10	Requête GET et PUT au serveur par 2 clients différents . . . . .	17
11	Le fonctionnent les addons natifs dans Node.js . . . . .	18
12	Structure des dossiers avec addons . . . . .	18
13	Intergration Nedb dans le projet . . . . .	21
14	Intégration stratégie passport-local . . . . .	22
15	Intégration du fonction crypto username/password . . . . .	23
16	Page de connexion redirigée fournie par Google lors de la tentative d'accès à un itinéraire protégé . . . . .	24
17	Flux de travail OAuth2.0 avec Google . . . . .	25
18	Génération de paires de clés dans un projet node.js . . . . .	26
19	Exemple un jeton JWT . . . . .	27
20	Exemple de connexion en tant qu'administrateur pour obtenir toutes les données . . . . .	29
21	Faire des erreurs lors de la connexion/de l'utilisation de l'API . . . . .	29
22	Implémenter le gestionnaire d'erreurs . . . . .	30
23	Implémenter des rôles différents . . . . .	31
24	Utilisation d'un jeton pour l'utilisateur et pour l'administrateur afin d'accéder à la route restreinte . . . . .	31
25	Architecture de supervision de CP9000 détaillée . . . . .	32
26	Génération de paires de clés dans un projet node.js . . . . .	33
27	Test de SET et GET de l'ipstream ID 1 . . . . .	35
28	Documentation API avec Swagger . . . . .	35

## Introduction

Harmonic est une entreprise spécialisée dans le traitement et le service vidéo. Dans le cadre de la machine Vibe CP9000, un encodeur qui serve à la transmission des données dans nombreux des services streamings, l'entreprise a envie d'entraîner le stagiaire en construisant une interface API pour gérer toutes les configurations de la machine. Le stagiaire doit comprendre comment construire une API à partir de zéro et adapter cette API à une structure existante qui a été construite et stabilisée depuis longtemps. De cette façon, le stagiaire pourra approcher de plus en plus des activités dans le domaine web service et ainsi renforcer les compétences essentielles d'un ingénieur du développement cross-platform comme programmation Linux, C/C++, développement web avec JavaScript.

Pendant mon stage de 4 mois, j'assurais le rôle de développeur et me concentre sur une application plus concrète : de disposer à terme d'une REST API pour permettre la supervision complète du CP9000 et son intégration dans les nouveaux systèmes de management des clients d'Harmonic. Cette REST API doit permettre l'authentification des opérateurs avant d'offrir l'accès à la supervision. Il existe de nombreux Framework de supervision modernes (exemple SPRING) qui offrent toutes les briques de bases pour construire des REST API. Cependant, le CP9000 n'utilise pas une distribution Linux classique mais un kernel Linux temps réel et un root file system générés spécifiquement pour cette plateforme et ce produit. Il est donc très compliqué et coûteux (délai de développement, charge CPU et empreinte mémoire) de redesigner l'interface de supervision avec l'un de ces Framework. Une solution plus légère et pragmatique est donc envisagée :

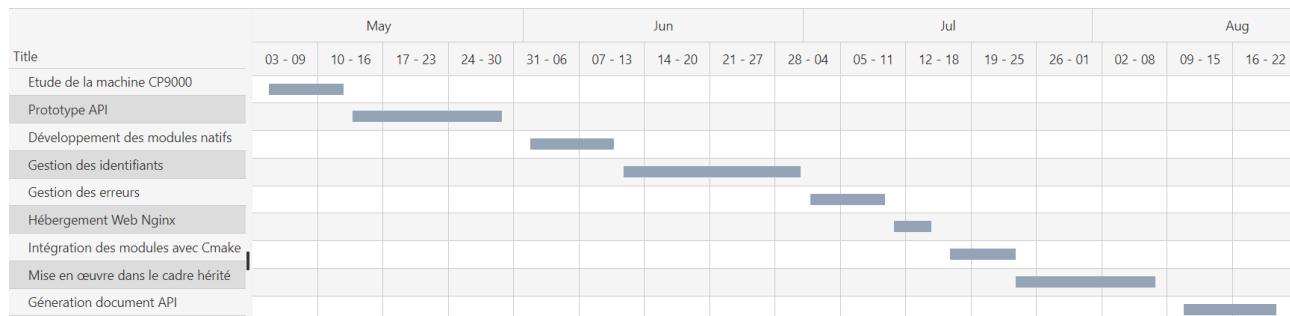
1. Gestion https pour l'interface web et la REST API via un serveur Nginx configuré en proxy.
2. Instanciation de Node.js et d'un nombre restreint de modules pour la REST API avec un wrapper pour l'interface avec le Framework de supervision existant.

Le point (1) ainsi que linstanciation de Node.js du point (2) étaient disponibles au début du stage. Le stage vise à prototyper complètement le point (2). Mon objectif est donc de chercher des solutions faisables pour la telle problématique et réaliser des tests nécessaires et la démonstration.

Ce rapport est alors constitué de 4 sessions principales :

- Présentation de l'entreprise
- Description détaillée du projet
- Points techniques avec des solutions et des démos
- Conclusion de stage

Ci-dessous, le diagramme de Gantt qui représente les différentes étapes de mon travail durant mon stage de 4 mois.



## 1 L'entreprise

Ce chapitre présente l'entreprise Harmonic France où j'effectue mon stage. J'ai effectué mon stage à distance pour le site de Rennes, mais Harmonic est prestataires répartis sur 25 sites dans le monde et en France, la société possède 3 sites : Issy-les-Moulineaux, Brest et Rennes.

### 1.1 Présentation de l'entreprise

Harmonic Inc. est une société technologique américaine qui développe et commercialise des produits de routage vidéo, de serveur et de stockage pour les entreprises qui produisent, traitent et distribuent du contenu vidéo pour la télévision et Internet (broadcast et OTT) via des technologies innovantes de Cloud et SaaS (Software as a Service).

Avoir été créée dans la Silicon Valley il y a 25 ans, Harmonic est l'une des entreprises leader dans la transformation de la vidéo ainsi que dans la fourniture de câbles et médias. Qu'il s'agisse de simplifier le streaming en utilisant le cloud ou le logiciel en tant que service, ou en aidant les câblo-opérateurs à déployer des services gigabit de nouvelle génération, la société permet à ses clients de monétiser et de diffuser leur contenu sur chaque écran.



FIGURE 1 – Présence d'Harmonic sur le monde

La société est implanté sur 25 sites dans de multiples pays avec plus de 5000 partenaires mondiaux, dépense environ 85 millions de dollars chaque année dans le département R&D et emploie plus de 1 500 ingénieurs dans le monde qui travaillent pour :

- Grandes entreprises internationales.
- PME technologiques [PME : petites et moyennes entreprises].

Harmonic peut être considéré comme un leader du marché des services médiatiques et solutions technologiques.

### 1.2 Harmonic France site Rennes

En France, la société possède 3 sites : Issy-le-Moulineaux, Brest et Rennes. Le site de Rennes est le plus grand site de l'entreprise en France. Il est responsable de l'ingénierie, du développement logiciel et du service client. C'est ici que naissent de nombreuses solutions technologiques impressionnantes.

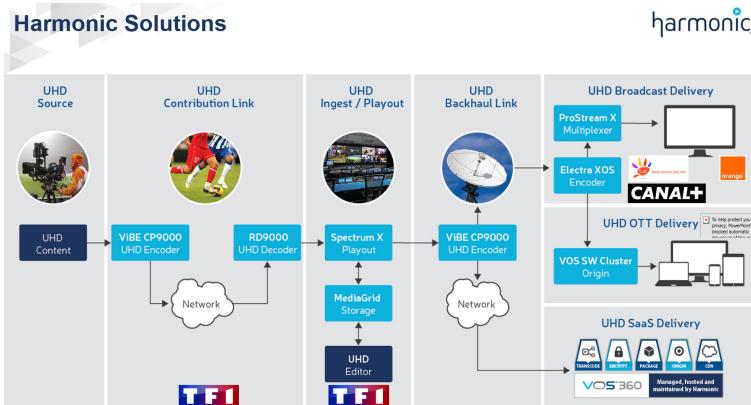


FIGURE 2 – Les solutions technologies données par Harmonic

Ce site est l'endroit où je fais mon stage mais malheureusement en raison du covid 19, la plupart des employés doivent travailler à domicile. Après 3 jours d'intégration, cela fut aussi le cas pour moi.

### 1.3 Les membres concernés

Ceux sont les personnes avec lesquelles j'ai travaillé au cours de mon stage de 4 mois.

Philippe CORMICHET	Maître de stage
Gildas CANCOUET	Ingénieur
Bruno DERRIEN	Ingénieur
Jean-Luc LEBLANC	Ingénieur

## 2 Contexte et description détaillé du projet

### 2.1 Contexte du projet de stage

Le ViBE CP9000 est un encodeur utilisé par les Broadcaster à travers le monde pour la transmission d'événements sportifs. L'objet du stage est le développement d'une interface Restful API pour le ViBE CP9000 afin de pouvoir l'intégrer au management system des clients.

- Définition de l'API en s'inspirant d'autres API existantes
- Participation au développement et à l'intégration
- Test et validation de l'API

CP9000 est un encodeur avec beaucoup de fonctions et une grande complexité, mais dans ce stage, nous nous concentrerons uniquement sur la création d'une interface api pour pouvoir interagir avec les données dans le cadre de supervision hérité. Le diagramme suivant montre simplement comment nous pouvons implémenter cette API dans CP9000.

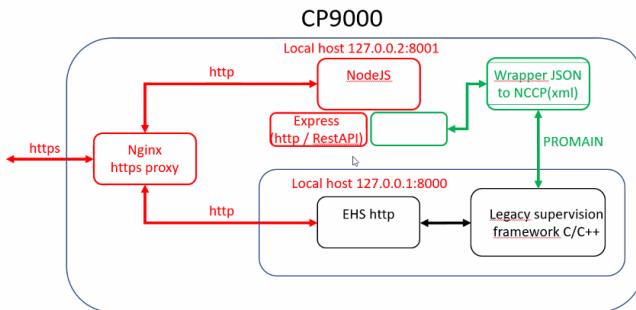


FIGURE 3 – Implémentation API dans CP9000

Ce processus sera divisé en 2 temps, d'abord, nous devons créer une API de repos en utilisant NodeJs et Express, puis utiliser nginx pour héberger cette API. Ensuite, nous devons créer une fonction wrapper pour échanger des informations entre cette API et les données NCCP stockées dans un fichier .xml.

## 2.2 Outils de travail

### Matériels physiques

Pour ce faire, le stagiaire dispose d'un ordinateur linux sur site pour la programmation et d'un ordinateur portable Windows qui peut être connecté à un ordinateur portable linux via ssh (MPutty). L'objectif est de simplifier le travail à distance pour le stage.

### NodeJs et ExpressJs

Node.js est un environnement d'exécution open source et multiplateforme permettant d'exécuter du code JavaScript en dehors d'un navigateur. Nous utilisons souvent Node.js pour créer des services back-end tels que des API dans Web App, qui sont pilotés par des événements côté serveur et écrits en JavaScript.

Express est un petit framework qui repose sur la fonctionnalité de serveur Web de Node.js pour simplifier ses API et ajouter des fonctionnalités utiles. Il facilite l'organisation des fonctionnalités de votre application avec le middleware et le routage. Il ajoute des utilitaires utiles aux objets HTTP de Node.js.

npm (Node Package Manager) est un gestionnaire de packages pour le langage de programmation JavaScript. npm est le gestionnaire de packages par défaut pour l'environnement d'exécution JavaScript Node.js. Il se compose d'un client en ligne de commande et d'une base de données en ligne de packages qui seront utilisés pour construire un projet Node.js.

### Nginx

NGINX est un logiciel libre de serveur Web (ou HTTP) ainsi qu'un proxy inverse. L'utilisation la plus fréquente de NGINX est de le configurer comme un serveur Web classique pour servir des fichiers statiques et comme un proxy pour les requêtes dynamiques.

### Logiciel PuTTY

PuTTY est un émulateur de terminal, une console série et une application de transfert de fichiers réseau. Dans le cadre de notre projet, il est utilisé sur ordinateur Windows pour pouvoir accéder au terminal sur ordinateur linux en distanciel via ssh.

### Cross-compiler sur srv0359

Les étudiants se verront attribuer un châssis pour effectuer le stage. Cela n'aura pas trop d'impact si nous construisons simplement l'application en JavaScript car nous pouvons enregistrer les modules directement sur

ce châssis. Cependant, lorsque l'on ajoute des fonctions liées au C/C++, il est nécessaire d'avoir un poste de travail pour s'occuper de la fonction de compilation.

### **SVN (Subversion)**

SVN est utilisé pour gérer et suivre les modifications apportées au code et aux actifs à travers les projets. La configuration SVN a besoin de deux éléments principaux : le serveur, qui contient toutes les versions de tous les fichiers sources et une copie locale des fichiers, qui se trouve sur l'ordinateur local. Les utilisateurs modifient les fichiers de travail sur leur local. Ils valident ensuite leurs modifications sur le serveur SVN. Chaque fois qu'un utilisateur valide un changement, SVN le gère et l'enregistre en créant une nouvelle version.

### **VSCode (Visual Studio Code)**

VSCode est un éditeur de code source créé par Microsoft pour Windows, Linux. La force de VSCode est sa capacité de personnalisation. Sur VSCode, nous pouvons installer des extensions, il existe de nombreuses extensions gratuites dans VSCode, qui peuvent aider à la mise en évidence, au formatage du code au débogage, au client API, etc.

## **2.3 Déroulement du stage**

Les présentations du produit (usage, architecture) et de l'environnement de développement ont été réalisées en plusieurs passes au court du stage en fonction des besoins. J'ai réalisé plusieurs POC qui ont permis d'évaluer les avantages et inconvénients des différentes solutions technique (base de données locale à la REST API ou pas, technique de wrapping de module C++). J'ai retenu l'environnement de travail Windows pour prendre en main la REST API et réaliser les première POC car :

- Il était plus simple à prendre en main (IDE tout intégré disponible)
- Il me permettait de travailler plus efficacement (En local sur mon laptop Windows pour éviter les problème d'accès aux ressources de l'entreprise dans le cadre du télétravail).

### 3 Détails techniques des solutions

#### 3.1 Développement web et les protocoles de communication

##### 3.1.1 Développement web back-end

Le développement Web est la création et la maintenance de sites Web ; c'est le travail qui se fait en coulisse pour donner à un site Web une belle apparence, fonctionner rapidement et bien fonctionner avec une expérience utilisateur transparente. Les développeurs Web le font en utilisant une variété de langages de codage. Les langues qu'ils utilisent dépendent des types de tâches qu'ils effectuent et des plates-formes sur lesquelles ils travaillent. Le domaine du développement Web est généralement divisé en frontend (côté utilisateur) et back-end (côté serveur). Plongeons dans les détails.

**Back-End Development** Le développeur backend conçoit ce qui se passe dans les coulisses. C'est là que les données sont stockées, et sans ces données, il n'y aurait pas de frontend. Le backend du Web se compose du serveur qui héberge le site Web, d'une application pour l'exécuter et d'une base de données pour contenir les données.

Le développeur backend utilise des programmes informatiques pour s'assurer que le serveur, l'application et la base de données fonctionnent parfaitement ensemble. Ce type de développeur doit analyser les besoins d'une entreprise et fournir des solutions de programmation efficaces. Pour faire toutes ces choses incroyables, ils utilisent une variété de langages côté serveur, comme PHP, Python et en particulier JavaScript, qui sera l'objectif principal de mon stage.

##### 3.1.2 Aperçu rapide de l'IP (Internet Protocol)

Comme plus loin, Je serai plus précis sur le fonctionnement d'une application côté serveur, il est préférable de bien comprendre certains protocoles de communication Web qui seront liés au fonctionnement de mon travail. Internet Protocol est une famille de protocoles de communication de réseaux informatiques conçus pour être utilisés sur Internet. Les protocoles IP s'intègrent dans la suite des protocoles Internet et permettent un service d'adressage unique pour l'ensemble des terminaux connectés.

##### TCP et UDP

Il existe deux types de trafic IP (Internet Protocol). Ce sont TCP ou Transmission Control Protocol et UDP ou User Datagram Protocol. TCP est orienté connexion – une fois qu'une connexion est établie, les données peuvent être envoyées dans les deux sens. UDP est un protocole Internet plus simple et sans connexion. Plusieurs messages sont envoyés sous forme de paquets en morceaux à l'aide d'UDP.

Propriétés	TCP	UDP
Connexion	TCP est un protocole orienté connexion.	User Datagram Protocol est un protocole sans connexion
Flux de données	Les données sont lues sous forme de flux d'octets, aucune indication distinctive n'est transmise aux limites du message de signal (segment).	Les paquets sont envoyés individuellement et leur intégrité n'est vérifiée que si elles arrivent. Les paquets ont des limites définies qui sont respectées à la réception, ce qui signifie qu'une opération de lecture sur le socket récepteur produira un message entier tel qu'il a été envoyé à l'origine.

## Serveur proxy inverse

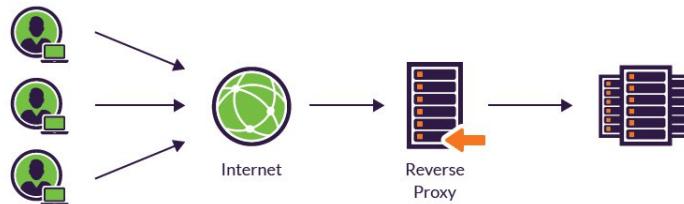


FIGURE 4 – La fonctionnement du serveur proxy inverse

Un serveur proxy inverse est un point de connexion intermédiaire positionné à la périphérie d'un réseau. Il reçoit les demandes de connexion HTTP initiales, agissant comme le point de terminaison réel. Un proxy inverse fonctionne :

- Réception d'une demande de connexion utilisateur
- Terminer une négociation TCP, mettre fin à la connexion initiale
- Connexion au serveur d'origine et transfert de la demande

### Application dans le contexte du projet

Pendant mon projet, mon équipe utilisera Nginx comme serveur web HTTP pour héberger mon application. De plus, il est également utilisé comme proxy inverse et équilibré de charge pour gérer le trafic entrant et le distribuer vers des serveurs en amont plus lents, des serveurs de bases de données hérités aux micro services. Comme mon application utilise Node.js, c'est un point supplémentaire pour utiliser Nginx car il partage ces caractéristiques architecturales avec Nginx.

Cependant, Node.js présente aussi quelques points faibles et vulnérabilités qui peuvent rendre les systèmes basés sur Node.js sujets à des sous-performances ou même à des plantages. Les problèmes surviennent plus fréquemment lorsqu'une application Web basée sur Node.js connaît une croissance rapide du trafic. Pour tirer le meilleur parti de Node.js, vous devez mettre en cache le contenu statique, créer un proxy et équilibrer la charge entre plusieurs serveurs d'applications et gérer les conflits de ports entre les clients, Node.js et les assistants. NGINX peut être utilisé à toutes ces fins, ce qui en fait un excellent outil pour le réglage des performances de Node.js.

Face à un site à fort trafic, nous pouvons augmenter les performances des applications en mettant un serveur proxy inverse devant votre serveur Node.js. Cela protège le serveur Node.js de l'exposition directe au trafic Internet et permet une grande flexibilité dans l'utilisation de plusieurs serveurs d'applications, dans l'équilibrage de charge entre les serveurs et dans la mise en cache du contenu.

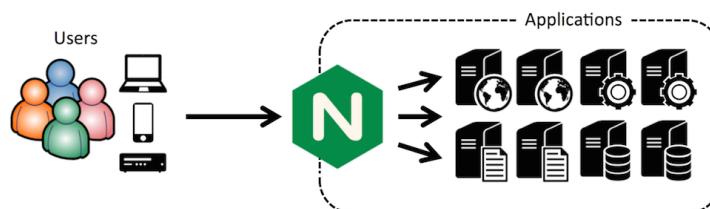


FIGURE 5 – Implémenter un serveur proxy inverse avec Nginx

L'utilisation de NGINX en tant que serveur proxy inverse Node.js présente des avantages spécifiques, notamment :

- Simplification de la gestion des priviléges et des attributions de ports
- Servir plus efficacement les images statiques

- La gestion des plantages de Node.js avec succès
- Atténuer les attaques DoS

### 3.1.3 Introduction de API REST

**API WEB** API est une interface de programmation d'applications pour un serveur Web ou un navigateur Web. Une API Web côté serveur est une interface de programmation composée d'un ou de plusieurs points de terminaison exposés publiquement à un système de message de demande-réponse défini, généralement exprimé en JSON ou XML, qui est exposé via le Web, le plus souvent au moyen d'un HTTP basé sur serveur Web.

**Information structurée JSON (JavaScript Object Notation)** JSON est un format de fichier standard ouvert et un format d'échange de données qui utilise du texte lisible par l'homme pour stocker et transmettre des objets de données constitués de paires attribut-valeur et de tableaux (ou d'autres valeurs sérialisables). Il s'agit d'un format de données commun avec une gamme variée de fonctionnalités dans l'échange de données, y compris la communication d'applications Web avec des serveurs.

```

1  {
2    {
3      "id": "981681",
4      "info": ["ABC", "abc"]
5    },
6    {
7      "id": "291234",
8      "info": "DEF"
9    }
10 }
```

**Endpoints** Endpoints sont des aspects importants de l'interaction avec les API Web côté serveur, car ils spécifient où se trouvent les ressources accessibles par des logiciels tiers. Généralement l'accès se fait via une URI sur laquelle sont postées les requêtes HTTP, et dont la réponse est donc attendue. Les API Web peuvent être publiques ou privées, cette dernière nécessitant un token d'accès. nécessitant un token d'accès.

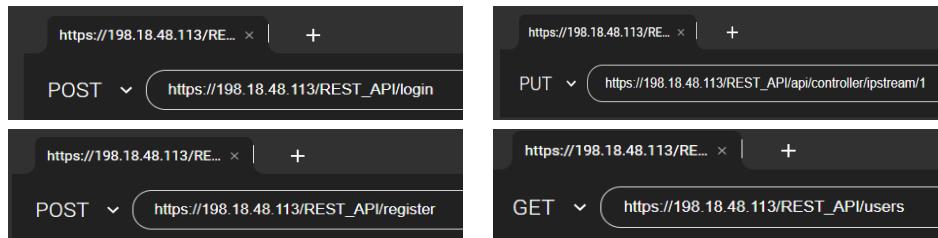


FIGURE 6 – Exemple endpoints utilisant dans notre projet

**Le protocole REST** REST (REpresentational State Transfer) constitue un style architectural et un mode de communication fréquemment utilisé dans le développement de services Web. REST utilise les méthodes HTTP et il s'articule autour d'une ressource où chaque composant est une ressource et une ressource est accessible par une interface commune à l'aide de méthodes standard HTTP.

Les quatre méthodes HTTP suivantes sont couramment utilisées dans l'architecture basée sur REST.

- HTTP GET est utilisé pour fournir un accès en lecture seule à une ressource.
- HTTP PUT est utilisé pour créer une nouvelle ressource.
- HTTP DELETE est utilisé pour supprimer une ressource.
- HTTP POST est utilisé pour mettre à jour une ressource existante ou créer une nouvelle ressource.

Un service Web est un ensemble de protocoles et de normes ouverts utilisés pour échanger des données entre des applications ou des systèmes. Les applications logicielles écrites dans divers langages de programmation et exécutées sur diverses plates-formes peuvent utiliser des services Web pour échanger des données sur des réseaux informatiques comme Internet d'une manière similaire à la communication interprocessus sur un seul ordinateur. Cette interopérabilité est due à l'utilisation de standards ouverts.

Les services Web basés sur l'architecture REST sont appelés services Web RESTful. Ces webservices utilisent des méthodes HTTP pour implémenter le concept d'architecture REST. Un service Web RESTful définit généralement un URI, Uniform Resource Identifier un service, qui fournit une représentation des ressources telles que JSON et un ensemble de méthodes HTTP

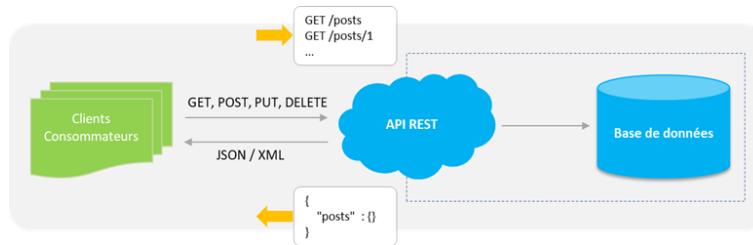


FIGURE 7 – Le principe de l'architecture REST

L'**API REST** nous offre certains avantages comme :

- Les services Web RESTful sont faciles à exploiter à l'aide de la plupart des outils
- REST est conçu pour une utilisation sur un Internet/Web ouvert. Il constitue un meilleur choix pour les applications Web et à plus forte raison pour les plateformes en Cloud.
- Le protocole REST associe des bonnes performances à un coût moindre à long terme

## 3.2 Développement API REST avec Node.js et Express.js

### 3.2.1 NodeJS et Framework ExpressJS

**NodeJS** NodeJS est entièrement piloté par les événements. Fondamentalement, le serveur se compose d'un thread traitant un événement après l'autre.

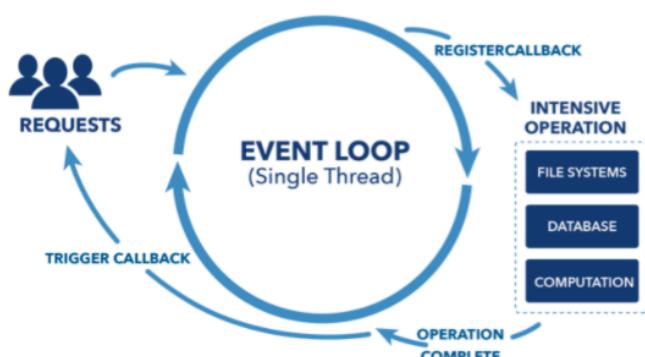


FIGURE 8 – Event-Loop NodeJS

Une nouvelle demande entrante est un type d'événement. Le serveur commence à le traiter et lorsqu'il y a une opération d'E/S bloquante, il n'attend pas qu'elle se termine et enregistre à la place une fonction de rappel. Le serveur commence alors immédiatement à traiter un autre événement (peut-être une autre demande). Lorsque l'opération IO est terminée, c'est un autre type d'événement, et le serveur le traitera (c'est-à-dire continuera à travailler sur la demande) en exécutant le rappel dès qu'il en aura le temps.

Le principal avantage d'un serveur Node.js est qu'il peut gérer beaucoup plus de trafic qu'un serveur conventionnel. Les autres serveurs sont multithreads, ce qui signifie que chaque client, lorsqu'il est connecté au site Web, obtient son propre thread. Chaque thread s'occupe des requêtes faites par son client. Avec Node.js,

alors qu'il n'y a qu'un seul thread (la boucle d'événement - event loop), lorsqu'une requête est faite, la boucle d'événement peut la passer à une autre fonction à exécuter puis quand elle obtient terminer et renvoyé à l'utilisateur l'interaction se termine. Il n'est pas nécessaire d'avoir une connexion ouverte (thread) à tout moment car elle est pilotée par les événements(event-drivent structure). Cela signifie qu'un serveur Node.js peut accueillir plus de clients, avec la même quantité de mémoire, qu'un autre serveur multithread.

**ExpressJS** ExpressJs est un Framework d'applications Web gratuit et open source pour Node.js. Il est utilisé pour concevoir et créer une API Web plus rapidement et plus facilement que d'utiliser uniquement Node. ExpressJs fournit de nombreuses fonctionnalités sous la forme de fonctions qui peuvent être facilement utilisées n'importe où dans le programme.

- Le middleware est une partie du programme qui a accès à la base de données, à la demande du client et aux autres middlewares. Il est principalement responsable de l'organisation systématique des différentes fonctions d'Express.js.
- Il fournit également un mécanisme de routage très avancé qui aide à préserver l'état de la page Web à l'aide de leurs URL.
- Il fournit également des moteurs de modèles qui permettent aux développeurs de créer du contenu dynamique sur les pages Web en créant des modèles HTML côté serveur.
- ExpressJS facilite le débogage en fournissant un mécanisme de débogage qui a la capacité d'identifier la partie exacte de l'application Web qui a des bogues.

### 3.2.2 Le première démo

Tout d'abord, nous commençons par concevoir une API simple avec des fonctions qui simulent les exigences du produit final mais le rendent plus simple. La première chose à faire est de créer une base de données simple à travers laquelle nous pouvons tester les fonctionnalités de cette api.

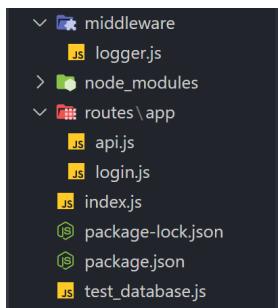


FIGURE 9 – La structure du démo

Considérons que nous avons un fichier de base de données .js d'utilisateurs et d'informations à récupérer lors de l'utilisation des méthodes HTTP. J'ai les utilisateurs et les données suivants dans un fichier testdatabase.js :

```

1 // test_database.js
2
3 const datas = [
4     { id: 1, info: 'ABC' },
5     ...
6 ];
7 const users = [
8     { id: '1', username: 'admin', password: 'admin' },
9     ...
10 ];
11 module.exports = {datas, users};

```

Notez que bien qu'il ne soit pas possible d'utiliser un fichier javascript comme base de données (le fichier .js ne peut pas être modifié pendant l'exécution du nœud de serveur), dans un souci de simplifier la démo, je l'ai

configuré. Cette conception de fichier a une structure similaire à un fichier `.json`. Sur la base de ces informations, nous allons fournir les API RESTful suivantes :

URI	Méthode HTTP	POST body	Résultat
/api	GET	NULL	Afficher toutes les données
/api/<id>	GET	NULL	Afficher un élément existant
/api	POST	JSON String	Ajouter un nouveau élément
/api/<id>	PUT	JSON String	Modifier un élément existant

## Le première demo

Tout d'abord, pour démarrer l'application correctement, j'affecte express en tant que variable à utiliser facilement plus tard.

```

1 //index.js
2
3 const express = require('express');
4 const logger = require('./middleware/logger');
5 const PORT = process.env.PORT || 8001;
6 const datas = require('./test_database');
7 const app = express();
8
9 //Logger middleware
10 app.use(logger);
11 // Body-parser Middleware
12 app.use(express.json());
13 app.use(express.urlencoded({ extended: false }));
14
15 //Login Routes
16 app.use('/login', require('./routes/app/login'));
17 //API Routes
18 app.use('/api', require('./routes/app/api'));
19
20 app.listen(PORT, '127.0.0.2', () =>
21   console.log(`Server started on port ${PORT}`)
22 );

```

Nous utilisons `app.listen` pour démarrer le serveur 127.0.0.2 sur le port 8001. En utilisant `express.router()`, un moyen de séparer les routes pour faciliter la gestion des applications, je peux attacher 2 autres routes, `/api` et `/login` et les appeler avec `app.use()`.

Comme il s'agit de la première démo, je ne me soucie pas de l'authentification de l'utilisateur mais me concentre uniquement sur les fonctions principales de l'Api, donc la route `/login` n'a pas trop de différences notables par rapport à la route `/api`. C'est pourquoi j'ai décidé de le sauter dans mon explication. De plus, j'utilise également d'autres middleware comme un logger personnalisé `app.use(logger)` pour notifier sur le terminal lorsqu'une requête est envoyée au serveur et un middleware d'analyse corporelle intégrée dans Express pour analyser les corps des requêtes entrantes dans un middleware avant les gestionnaires. Et pour l'api de route, qui est au centre de la démo, j'ai configuré les méthodes HTTP et utilisé `express.Router()` pour l'appeler dans l'application principale. Dans chacun de ces itinéraires, j'effectue des opérations en utilisant les informations de la requête pour communiquer avec la base de données.

Par exemple, dans cette route GET, je récupère d'abord l'identifiant de l'URL, puis je recherche dans la base de données pour voir s'il existe. Si c'est le cas, je récupère les données correspondant à cet identifiant et les envoie à l'utilisateur au format json avec le statut HTTP 200. Sinon, j'enverrai un message introuvable avec le statut HTTP 404.

```

1 //api.js
2
3 // Get all datas
4 router.get('/', (req, res) => {
5   res.json(database.datas); // return as .json
6 });
7 // Get single data
8 router.get('/:id', (req, res) => {

```

```

9  const finding_function = (data) => data.id === parseInt(req.params.id);
10 const existed = database.datas.some(finding_function);
11 if (existed) {
12   const resultInfo = database.datas.filter(finding_function);
13   res.status(200).json(resultInfo);
14 } else {
15   res.status(404).json({
16     msg: 'Data with the id ${req.params.id} not existed',
17   });
18 }
19 });
20
21 // Modify an element
22 router.put('/:id', (req, res) => {
23   const finding_function = (data) => data.id === parseInt(req.params.id);
24   const existed = database.datas.some(finding_function);
25   if (existed) {
26     const updateInfo = req.body;
27     database.datas.forEach((data) => {
28       if (data.id === parseInt(req.params.id)) data.info = updateInfo.info;
29     });
30     const editedDatas = database.datas;
31     res.status(200).json({
32       msg: 'Updated data',
33       editedDatas,
34     });
35   } else {
36     res.status(404).json({
37       msg: 'Data with the id ${req.params.id} not existed',
38     });
39   }
40 });

```

Toute cette méthode se fera de manière asynchrone dans la fonction callback, qui prend 2 paramètres req et res. Par cela, il suivra la norme NodeJS et ne bloquera pas le serveur en attendant qu'une demande d'un utilisateur se termine. Un autre côté client, une autre personne peut envoyer la requête au serveur par exemple avec cette route PUT pour modifier un certain élément en ajoutant un corps à sa requête.

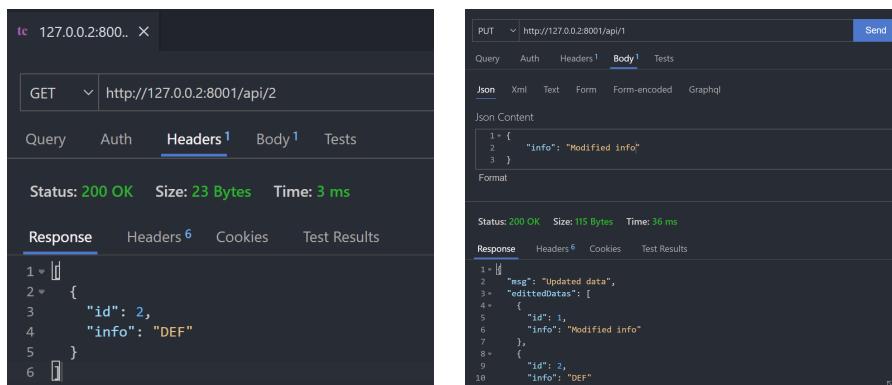


FIGURE 10 – Requête GET et PUT au serveur par 2 clients différents

### 3.3 C++ Addons

Node.js est un environnement d'exécution JavaScript multiplateforme qui s'exécute sur le moteur V8, qui est en fait écrit en C++, il ne fait aucun doute que Node.js lui-même a la capacité de fonctionner avec du code C/C++. C'est exactement ce que font les addons NodeJS.

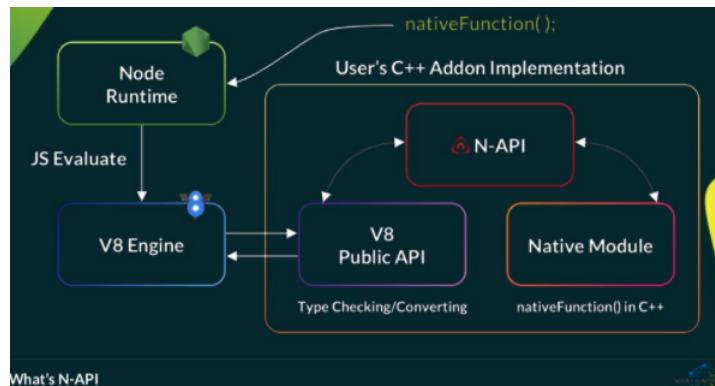


FIGURE 11 – Le fonctionnement les addons natifs dans Node.js

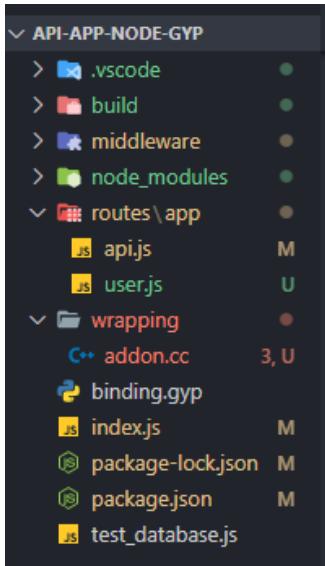


FIGURE 12 – Structure des dosiers avec addons

En bref, vous pouvez penser que les addons fonctionnent comme une fonction C++ qui sera appelée en javascript mais toujours compilée dans un environnement C++.

- Les addons Node.js sont des objets partagés liés dynamiquement, écrits en C++.
- Il peut être chargé dans Node.js à l'aide de la fonction require() et également utilisé comme s'il s'agissait d'un module Node.js ordinaire.
- Il est principalement utilisé pour fournir une interface entre JavaScript exécuté dans Node.js et les bibliothèques C/C++.

Il y a plusieurs raisons pour lesquelles nous choisissons d'utiliser des addons C++ dans un projet Node.js comme :

- Nous pouvons intégrer une bibliothèque tierce écrite en C/C++ et l'utiliser directement dans Node.js.
- Il donne également la possibilité d'utiliser des bibliothèques C++ dans Node.js.
- Il donne la possibilité de faire des calculs intensifs, parallèles et de haute précision car les performances de C++ sont bien meilleures sur des valeurs de calcul plus importantes.

Il existe trois options pour implémenter des addons : Node-API, NAN ou utilisation directe des bibliothèques internes V8, libuv et Node.js. Dans le but de pouvoir s'intégrer au Framework hérité C/C++, ma direction est de commencer par le chemin le plus bas vers le chemin le plus pratique, puis je choisirai celui qui est le plus approprié pour continuer.

### 3.3.1 Outil de compilation

Pour cette partie, je ne mettrai pas trop l'accent sur le fonctionnement des différentes outils car c'est trop compliqué. Au lieu de cela, je vais me concentrer sur la façon dont nous les utilisons.

**Node-gyp** Une fois le code source écrit, il doit être compilé dans le fichier binaire addon.node. Pour ce faire, créez un fichier appelé binding.gyp au niveau supérieur du projet décrivant la configuration de construction du module à l'aide d'un format similaire avec .json. Ce fichier est utilisé par node-gyp, un outil écrit spécifiquement pour compiler les addons Node.js.

```

1 // binding.gyp
2 {
3     "targets": [
4         {
5             "target_name": "addon",
6             "sources": ["./wrapping/addon.cc"]
7         }
8     ]
9 }
10 }
```

**Cmake** En termes du résultat, Cmake est similaire à node-gyp, la différence est que Cmake sera plus facile à intégrer dans l'environnement linux de CP9000.

```

1 // CMakeLists.txt
2
3 project (cmake-addon)
4 include_directories(${CMAKE_JS_INC})
5 file(GLOB SOURCE_FILES "./addon/addon.cc")
6 add_library(${PROJECT_NAME} SHARED ${SOURCE_FILES} ${CMAKE_JS_SRC})
7 set_target_properties(${PROJECT_NAME} PROPERTIES PREFIX "" SUFFIX ".node")
8 target_link_libraries(${PROJECT_NAME} ${CMAKE_JS_LIB})
9
10 # Include Node-API wrappers
11 execute_process(COMMAND node -p "require('node-addon-api').include"
12                  WORKING_DIRECTORY ${CMAKE_SOURCE_DIR}
13                  OUTPUT_VARIABLE NODE_ADDON_API_DIR)
14 string(REGEX REPLACE "[\r\n]+\" \" ${NODE_ADDON_API_DIR} ${NODE_ADDON_API_DIR}")
15
16 target_include_directories(${PROJECT_NAME} PRIVATE ${NODE_ADDON_API_DIR})
```

Après avoir appelé la commande pour générer le fichier addon.node compilé, le résultat sera placé dans le répertoire build/Release/. Une fois construit, l'addon binaire peut être utilisé depuis Node.js en faisant require() dans le module addon.node construit.

```

1 // test.js
2
3 const binding_test = require('./build/Release/binding_test');
4 try console.log(binding_test.binding());
5 catch(err) throw err;
```

### 3.3.2 Demo des addons dans un NodeJs

A propos de cette partie, je vais décrire 2 façons de créer des addons natifs dans un projet NodeJs : V8 et NAPI. En fait, il y a d'autres méthodes, mais j'ai choisi ces deux parce qu'elles sont les plus typiques dans l'industrie.

**Moteur JavaScript V8** V8 est le moteur JavaScript et WebAssembly haute performances open source de Google, écrit en C++. Il est souvent utilisé dans Chrome et dans l'application Node.js. V8 permet à n'importe quelle application C++ d'exposer ses propres objets et fonctions au code JavaScript. L'utilisation directement du module interne v8 me permet de créer un fonction C++ pour rappeler dans la partie JavaScript.

```

1 // addons.cc avec v8
2
3 void Compare(const FunctionCallbackInfo<Value> &args)
4 {
5     Isolate *isolate = args.GetIsolate();
6     if (args.Length() < 2) // Check the number of arguments passed.
7     {
8         // Throw an Error that is passed back to JavaScript
9         isolate->ThrowException(Exception::TypeError(
```

```

10         String::NewFromUtf8(isolate, "Must be 2 arguments entered").ToLocalChecked());
11     return;
12 }
13 if (!args[0]->IsNumber() || !args[1]->IsNumber()) // Check the argument types
14 {
15     isolate->ThrowException(Exception::TypeError(
16         String::NewFromUtf8(isolate, "Argument type musts be number").ToLocalChecked()));
17     return;
18 }
19 bool compare = args[0].As<Number>()->Value() == args[1].As<Number>()->Value();
20 args.GetReturnValue().Set(compare);
21 };
22 void Initialize(Local<Object> exports)
23 {
24     NODE_SET_METHOD(exports, "compare", Compare);
25     MyObject::Init(exports);
26 }
27
28 NODE_MODULE(NODE_GYP_MODULE_NAME, Initialize)

```

Bien que l'utilisation de la v8 soit le meilleur moyen de personnaliser nos fonctions pour développer des applications compliquées, ce n'est pas vraiment pratique pour un cas d'utilisation simple. Et comme la v8 est mise à jour presque deux fois par mois par Google, il est très difficile de maintenir ou de mettre à jour de manière stable notre programme. L'une des autres alternatives est donc N-API.

**N-API** NAPI fournit une API ABI-stable qui peut être utilisée pour développer des modules complémentaires natifs pour Node.js, simplifiant la tâche de création et de prise en charge de ces modules complémentaires dans Node.js. Donc, j'ai retravaillé la fonction de comparaison et ajouté quelques fonctions supplémentaires dans ces nouveaux add-ons.

```

1 //addon.cc avec Napi
2
3 static Napi::Boolean Compare(const Napi::CallbackInfo &info)
4 {
5     // 2 args input, verify if they're the same value
6     Napi::Env env = info.Env();
7     if (info.Length() != 2)
8         throw Napi::Error::New(env, "Method 'Compare' must be taking 2 arguments");
9     if (!info[0].IsNumber() || !info[1].IsNumber())
10         throw Napi::Error::New(env, "Method 'Compare' must be taking number value arguments");
11     return Napi::Boolean::New(env, info[0].ToNumber() == info[1].ToNumber());
12 }
13
14 static Napi::Object Init(Napi::Env env, Napi::Object exports)
15 {
16     exports.Set(Napi::String::New(env, "compare"),
17                 Napi::Function::New(env, Compare));
18     return exports;
19 }
20 NODE_API_MODULE(addon, Init)

```

**Appel des add-ons dans JavaScript** La fonction ci-dessus vérifiera les arguments entrés et renverra un booléen. Nous pouvons le tester en JavaScript en exécutant : `require('./build/Release/addon')`. Puis, en utilisant `addon.compare(<args>)`, nous pouvons appeler cette fonction dans notre route.

```

1 // api.js
2
3 // Get single data
4 router.get('/:id',(req, res) => {
5     //Compare id - Generate resulted array - Pass id and data as string
6     let result = addon.genArray(2);
7     for (let i = 0; i < database.datas.length; i++) {
8         if(addon.compare(parseInt(req.params.id), database.datas[i].id)) {
9             ...

```

### 3.4 Gestion des identifiants

La gestion des identités est une partie extrêmement importante de la conception d'une application. Passport.js est donc l'un des meilleurs outils pour cela. Mais d'abord, nous devons parler de la base de données que nous utiliserons dans les prochaines étapes.

Comme mentionné dans la section *3.2.2 Le premier démo REST API*, nous avons besoin d'avoir une base de données distincte pour stocker les informations des utilisateurs. Étant donné que CP9000 est un encodeur autonome et qu'il a ses propres méthodes de sécurité intégrées, nous devons choisir une base de données noSQL compacte qui peut être facilement sauvegardée et ne nécessite pas une haute sécurité. Après des recherches, mon groupe a accepté de choisir nedb.

**Base de donnée NeDB** NeDB est un système de base de données léger intégrable et en mémoire qui utilisait un sous-ensemble de l'API MongoDB. Développé par Louis Chatriot, il a été achevé en 2017 et est maintenant disponible gratuitement sur GitHub.

Avec pour objectif principal de fonctionner comme une base de données noSQL comme MongoDB mais plus légère car moins de fonctionnalités, NeDB peut être utilisé comme une banque de données en mémoire uniquement ou comme une banque de données persistante. Si vous connaissez MongoDB, une banque de données dans NeDB est l'équivalent d'une collection MongoDB et en partageant l'api MongoDB, les commandes et le flux de travail NeDB sont quelque peu similaires à MongoDB et il est donc très facile de travailler dessus si vous avez déjà une connaissance décente de MongoDB.

Comme tous les clients de base de données, la première étape consiste à se connecter à la base de données principale. Cependant, dans ce cas, il n'y a pas d'application externe à laquelle se connecter, nous devons donc simplement lui indiquer l'emplacement de vos données.

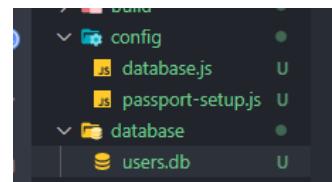


FIGURE 13 – Intergration Nedb dans le projet

```

1 // config/database.js
2
3 const Datastore = require('nedb');
4 const User = new Datastore({
5   filename: 'database/users.db',
6   autoload: true,
7 });
8
9 module.exports = User;

1 // users.db
2
3 {"username": "mockUser", "hash": "mockHash", "salt": "mockSalt", "role": "USER", "_id": "AdRY3WjPALPZsggrR"}
4 {"username": "mockAdmin", "hash": "mockHash", "salt": "mockSalt", "role": "ADMIN", "_id": "QLwBlnJYCHTRGTXG"}
```

Pour travailler avec notre base de données, nous appelons *Database.loadDatabase(<fonction de rappel>)* dans notre itinéraire.

```

1 // routes/auth/users.js
2
3 // Loading users database
4 const User = require('../config/database');
5 User.loadDatabase((err) => {
6   if (err != null) {
7     console.log('Error loading database');
8   }
});
```

**Passport.js** Passport est un middleware d'authentification pour Node.js. Comme il est extrêmement flexible et modulaire, Passport peut être déposé discrètement dans n'importe quelle application Web basée sur Express.

Le point fort du passeport est qu'il fonctionne comme un middleware séparé, ce qui n'affecte pas la structure globale d'une application. Passport reconnaît que chaque application a des exigences d'authentification uniques. Les mécanismes d'authentification, appelés "stratégies", sont regroupés sous forme de modules individuels. Les applications peuvent choisir les stratégies à utiliser, sans créer de dépendances inutiles. Donc malgré les complexités impliquées dans l'authentification, le code implémenté n'a pas besoin d'être compliqué.

```
1 // Middleware d'authentification passeport
2
3 app.post('/login', passport.authenticate('local', { successRedirect: '/data',
4 failureRedirect: '/login' }));

```

Dans cette section, avec passport.js je vais implémenter 3 façons différentes d'authentifier les utilisateurs, à savoir l'identifiant de OAuth2.0, de session et de JWT.

### 3.4.1 User session

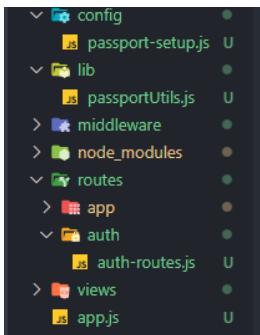


FIGURE 14 – Intégration stratégie passport-local

Chaque utilisateur se verra attribuer une session avec un identifiant correspondant à cette session chaque fois qu'il se connectera avec succès. Les informations de connexion seront donc enregistrées sur le navigateur en tant que cookie et ré-utilisées pour accéder aux routes protégées. La force de cette méthode est la simplicité, la réduction de la pression sur le serveur, mais c'est aussi sa faiblesse lorsque les informations sont stockées dans le navigateur, donc la session est très sécurisée.

**Stratégie passport-local** La stratégie d'authentification locale authentifie les utilisateurs à l'aide d'un "username" et d'un "password". La stratégie nécessite un rappel de vérification, qui accepte ces informations d'identification et les appels "done" fournissant un utilisateur. Le point important de cette méthode est que les informations de connexion de l'utilisateur seront enregistrées dans le navigateur en tant que cookie, chaque fois que l'utilisateur fait une demande, le navigateur attachera ces informations à cette demande. Pour commencer, la première chose que nous faisons est de configurer la fonction callback de vérification.

```
1 // passport-setup.js
2
3 const validPassword = require('../lib/passportUtils').validPassword;
4 passport.use(
5   new LocalStrategy(
6     {
7       usernameField: 'username',
8       passwordField: 'password',
9     },
10    (username, password, done) => {
11      for (i = 0; i < user.length; i++) {
12        const isValid = validPassword(password, user.hash, user.salt);
13        if (username === user[i].username && isValid) return done(null, user[i]);
14        else return done(null, false, {message: 'Incorrect username or password'});
15      })));

```

Puis pour travailler avec la session dans le navigateur, je dois créer un cookie unique et l'attribuer au navigateur. Ainsi, chaque fois que le navigateur envoie une demande au serveur avec ce cookie, le serveur s'en souvient et nous n'avons pas à refaire le processus de connexion pour chaque demande. La façon dont nous le faisons avec Passport, c'est en utilisant le sérialiseur de Passport dans lequel nous devons mettre l'identifiant de l'utilisateur. Et lorsque l'utilisateur sort de la session, je saisirai cet identifiant et le comparera avec l'identifiant dans la base de données par le déserialiseur puis passerai l'utilisateur dans l'étape suivante.

```

1 // passport-setup.js
2
3 passport.serializeUser((user, done) => {done(null, user.id)})
4
5 passport.deserializeUser((id, done) => {
6   for (let i = 0; i < user.length; i++) {
7     const _user = user[i].id;
8     if (_user === id) {
9       done(null, _user);
10    } else {
11      false;
12    }
13 });

```

Noté que dans mes fonctions, seul l'id utilisateur est sérialisé dans la session, ce qui réduit la quantité de données stockées dans la session.

Puis dans app.js, nous devons exiger l'intégralité de la configuration Passport pour que app.js le sache.

```

1 // app.js
2
3 require('./config/passport-setup');
4 app.use(passport.initialize());
5 app.use(passport.session());

```

Une fois que nous avons intégré avec succès la stratégie d'authentification, nous devons maintenant trouver un moyen de protéger les informations du compte utilisateur lorsqu'elles sont enregistrées dans la base de données.

### Fonction crypto username/password

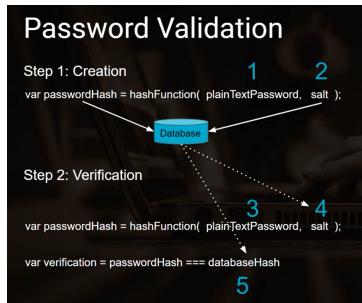


FIGURE 15 – Intégration du fonction crypto username/password

Il y a deux fonctions pour cela, une est de générer un mot de passe crypté et 2 est de vérifier le mot de passe. La fonction de génération prendra le mot de passe d'origine entré du côté de l'utilisateur et un "salt" est généré au hasard dans la bibliothèque de cryptographie de Node.js. En utilisant la méthode de cryptage, pbkdf2 je génère donc le mot de passe crypté "genHash".

```

1 // lib/passportUtils.js
2
3 const crypto = require('crypto');
4
5 function genPassword(password) {
6   // Generate random salt using crypto lib
7   var salt = crypto.randomBytes(32).toString('hex');
8   // Crypto method : pbkdf2, num of iterations: 1000, hash size : 64, hash func using : sha512
9   var genHash = crypto.pbkdf2Sync(password, salt, 10000, 64, 'sha512').toString('hex');
10  return {salt: salt, hash: genHash};
11 }

```

La fonction de vérification, lorsque l'utilisateur se connecte à son compte, la fonction obtiendra le mot de passe entré, puis elle récupère le sel de l'enregistrement de l'utilisateur dans la base de données (je peux le faire en vérifiant le nom d'utilisateur dans la fonction d'authentification de la route POST/login). Ensuite, en utilisant la même méthode de chiffrement pbkdf2, je crée un nouveau mot de passe de hachage et le compare avec le mot de passe dans la base de données.

```

1 // lib/passportUtils.js
2
3 function validPassword(password, hash, salt) {
4   var hashVerify = crypto.pbkdf2Sync(password, salt, 10000, 64, 'sha512').toString('hex');
5   return hash === hashVerify;
6 }
```

Enfin, j'appelle `passport.authenticate()` dans le rappel de la route pour effectuer l'authentification de la route respective.

```

1 // auth/auth-routes.js
2
3 router.post('/login', (req, res, next) => {
4   passport.authenticate(
5     'local',
6     { successRedirect: '/api',
7      failureRedirect: '/login',
8    },(err, user) => {})(req, res, next);
9 });
```

### 3.4.2 OAuth2.0

**OAuth2.0 (Open Authorization)** OAuth est un protocole d'autorisation standard ouvert qui offre aux applications la possibilité d'un accès désigné sécurisé. En fait, Oauth fonctionne en liant votre propre application à une application d'authentification tierce comme google, facebook, twitter, etc. que dans ce projet j'utiliserais google. Pour accéder à un itinéraire protégé dans votre application, l'utilisateur doit se connecter avec son compte google et une fois son compte vérifié par google, il peut accéder à cet itinéraire protégé.

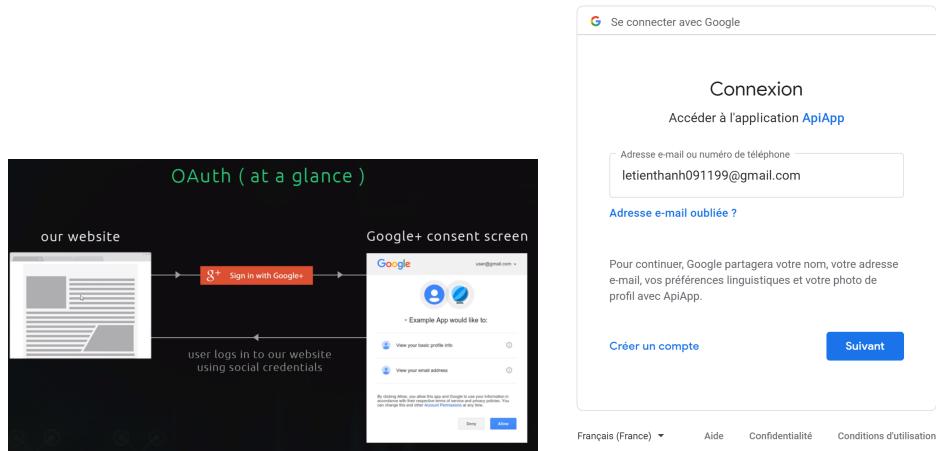


FIGURE 16 – Page de connexion redirigée fournie par Google lors de la tentative d'accès à un itinéraire protégé

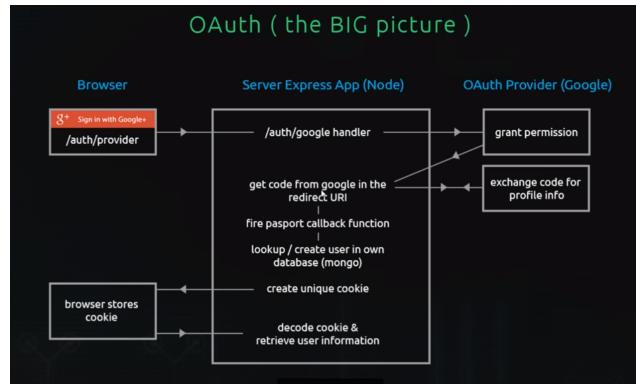


FIGURE 17 – Flux de travail OAuth2.0 avec Google

### Workflow d’Oauth2.0 avec l’implémentation stratégie passport-oauth2

La figure 11 vous décrit le workflow d’Oauth. Tout d’abord, pour accéder à une route protégée, l’utilisateur doit accéder à la route auth/google. Passport.js gérera ensuite le processus de redirection vers la page d’autorisation de Google (*la figure 12*).

Après avoir correctement connecté votre compte Google, Google vous renverra un code en l’ajoutant à l’URI redirigé. Ensuite, le passeport gérera ce code et le renverra à Google pour obtenir les informations de profil. Après avoir obtenu ces informations de profil, il déclenchera la fonction callback avec les informations réelles sur le profil. Ayant cette information, nous pouvons rechercher l’utilisateur dans la base de données en utilisant *User.findOne()*.

Dans la mise en œuvre, vous pouvez trouver que j’ai des «clés» comme les options pour Passport. Ces «clés» sont les informations d’identification fournies par Google <https://console.cloud.google.com/apis/> et sont associées à mon compte Google pour aider le passeport à générer le jeton d’accès associé à mon compte.

```

1 // config/passport-setup.js
2
3 const keys = require('./keys');
4 passport.use(
5     new OAuth2Strategy(
6         {
7             authorizationURL: keys.google_keys.authorizationURL,
8             tokenURL: keys.google_keys.tokenURL,
9             clientID: keys.google_keys.clientID,
10            clientSecret: keys.google_keys.clientSecret,
11            callbackURL: http://127.0.0.2:8001/auth/google/callback,
12        },
13        (accessToken, refreshToken, profile, cb) => {
14            //passport callback function
15            User.findOne({ googleid: profile.id }, (err, user) => {
16                return cb(err, user);
17            });
18        }
19    )
20 );

```

```

1 // auth/auth-routes.js
2
3 router.get('/google', passport.authenticate('oauth2', { scope: ['profile'] }));
4
5 router.get(
6     '/google/callback',
7     passport.authenticate('oauth2', {
8         successRedirect: '/api',
9         failureRedirect: '/login',
10    })
11 );

```

Après l’autorisation de Google avec succès, je dois stocker une partie du profil dans le navigateur sous forme de cookie afin de ne pas avoir à répéter ce processus. Ce processus est similaire à celui déjà expliqué dans l’identifiant de session mais a quelques petits changements pour s’adapter à la stratégie.

```

1 // config/passport-setup.js
2
3 passport.serializeUser((user, cb) => {cb(null, user.id)});
4 passport.deserializeUser((user, cb) => {
5   User.findById(id).then((user)=>{
6     cb(null,user);
7   })
8 });

```

Et je dois également définir manuellement la session de cookie dans app.js comme un middleware car cette stratégie n'a pas des modules de session cookie intégrés.

```

1 // app.js
2
3 app.use(require('cookieSession')({
4   maxAge: 24*60*60*1000,
5   keys:[ 'aVeryRandomCookieKey' ]
6 }));

```

### 3.4.3 JSON Web Token (JWT)

Toutes les options ci-dessus méritent d'être envisagées, mais sur la base des conditions du CP9000, l'option la plus réalisable est JWT. Pour comprendre comment fonctionne jwt, nous avons besoin d'une brève compréhension de la cryptographie asymétrique.

**Cryptographie asymétrique** L'idée est que lorsque nous transférons des informations entre différents ordinateurs, nous pouvons crypter les données avec une clé et décrypter avec une autre clé. Et donc la cryptographie asymétrique (alias Public-key cryptography) a été créé.

Elle est un système cryptographique qui utilise des paires de clés : des clés publiques (qui peuvent être connues d'autres personnes) et des clés privées (qui peuvent ne jamais être connues de personne, sauf du propriétaire). La génération de telles paires de clés dépend d'algorithmes cryptographiques basés sur des algorithmes mathématiques RSA. En ayant ces 2 clés, on peut chiffrer une information avec la clé privée mais on ne peut pas la déchiffrer avec cette clé, on ne peut utiliser que la clé publique pour la déchiffrer.

La particularité de ces 2 clés est que leur cryptage est une fonction de "trap-door", qui prend normalement des millions d'itérations. Cela signifie que lorsque nous cryptons avec une clé, il faut une éternité pour essayer d'inverser le processus ou de décrypter sans l'autre appairée clé même en connaissant l'algorithme et la clé publique. De plus, il crée toujours des données chiffrées suffisamment petites pour être envoyées via le header HTTP.

Avec node.js, il nous fournit une bibliothèque cryptographique "crypto" pour effectuer cette tâche

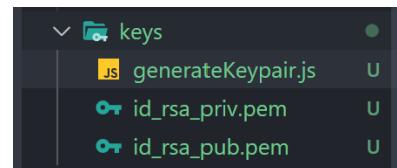


FIGURE 18 – Génération de paires de clés dans un projet node.js

```

1 // keys/generateKeypair.js
2
3 function genKeyPair() {
4   // Generates an object where the keys are stored in properties 'privateKey' and 'publicKey'
5
6   const keyPair = crypto.generateKeyPairSync('rsa', {
7     modulusLength: 4096, // bits - standard for RSA keys
8     publicKeyEncoding: {
9       type: 'pkcs1', // "Public Key Cryptography Standards 1"
10      format: 'pem' // Most common formatting choice
11    },
12    privateKeyEncoding: {
13      type: 'pkcs1', // "Public Key Cryptography Standards 1"
14      format: 'pem' // Most common formatting choice
15    }
16  });

```

```

15    });
16    // Create the public key file
17    fs.writeFileSync(_dirname + '/id_rsa_pub.pem', keyPair.publicKey);
18    // Create the private key file
19    fs.writeFileSync(_dirname + '/id_rsa_priv.pem', keyPair.privateKey);
20 }

```

Il peut donc faire une chose très utile pour notre projet : vérifier les identités avec la signature numérique.

```

1 \\ Digital signature
2 hashMessage = generateHash(message);
3 encrypt(hashMessage, private_key);
4 decrypt(hashMessage, public_key);

```

**JSon Web Token (JWT)** Basé sur le principe de la cryptographie asymétrique, JSON Web Token (JWT) permet l'échange sécurisé de jetons (tokens) entre plusieurs parties.

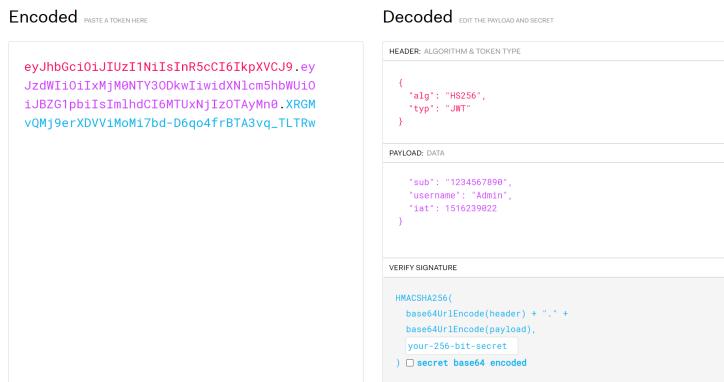


FIGURE 19 – Exemple un jeton JWT

Sous sa forme compacte, les jetons Web JSON se composent de trois parties séparées par des points ".", qui sont :

- En-tête (header) se compose généralement de deux parties : le type de jeton, qui est JWT, et l'algorithme de signature utilisé.
- Charge utile (payload) représente les informations embarquées. Dans notre cas, cette charge utile contiendra 3 informations : sub est un identifiant, le nom d'utilisateur et iat signifie "issue à".
- Signature numérique (Signature) est utilisée pour vérifier que le message n'a pas été modifié en cours de route et, dans le cas de jetons signés avec une clé privée, elle peut également vérifier que l'expéditeur du JWT est bien celui qu'il prétend être.

## Workflow d'authentification JWT

Avec toutes les informations ci-dessus, voici comment jwt s'intègre dans notre processus d'authentification : L'utilisateur se connecte d'abord avec son nom d'utilisateur et son mot de passe, le serveur vérifiera ensuite si ces informations d'identification sont valides. Si vrai, le serveur créera un JWT, le signera avec la clé privée du serveur et enverra ce JWT à l'utilisateur. L'utilisateur conservera ce JWT dans le stockage local du navigateur, puis attachera ce JWT à toute demande de route protégée. Après avoir reçu ce JWT de l'utilisateur, le serveur le vérifie ensuite avec la clé publique. Si le JWT réussit ce processus, l'utilisateur sera autorisé à accéder à cette route.

**Stratégie passport-jwt** Comme pour toute stratégie que j'ai présentée auparavant, nous devons d'abord mettre en place la stratégie *JwtStrategy(opt, <fonction callback>)*. Dans les options, je passe le jeton de l'en-

tête lorsque l'utilisateur l'envoie dans sa demande d'authentification, puis je le vérifierai avec la clé publique avec l'algorithme RS256.

```
1 //config/passport-setup.js - Options
2
3 const opts = {
4   jwtFromRequest: ExtractJwt.fromAuthHeaderAsBearerToken(),
5   secretOrKey: PUB_KEY,
6   algorithms: ['RS256'],
7 };
```

Et dans la fonction de rappel, lorsque je récupère l'identifiant de la charge utile dans le JWT, je vais rechercher dans la base de données utilisateur en fonction de cet identifiant.

```
1 //config/passport-setup.js - callback function
2
3 (jwt_payload, done) => {
4   User.findOne({ _id: jwt_payload.sub }, (err, user) => {
5     if (err != null) return done(err, false);
6     if (user != null) return done(null, user);
7     else return done(null, false);
8   });
}
```

Voilà pour la partie de vérification jwt envoyée par l'utilisateur, puis j'implémente la fonction issuer JWT lorsqu'un utilisateur se connecte. Le JWT émis aura la charge utile contenant l'identifiant de l'utilisateur, le moment où il est émis et sera signé par la clé privée en utilisant le même algorithme RS256.

```
1 //Dans lib/passportUtils.js
2
3 function issueJWT(user) {
4   const expiresIn = '1d';
5   const payload = {
6     sub: user._id,
7     iat: Date.now(),
8   };
9   const signedToken = jsonwebtoken.sign(payload, PRIV_KEY, {
10     expiresIn: expiresIn,
11     algorithm: 'RS256',
12   });
13   return {
14     token: 'Bearer ' + signedToken,
15     expires: expiresIn,
16   };
}
```

L'essentiel du processus de vérification du nom d'utilisateur, le déchiffrement du mot de passe de hachage ne varie pas beaucoup avec la stratégie passeport-local, mais cette fois un JWT est émis lorsque les informations d'identification sont corrigées.

```
1 //Dans auth/users.js
2 router.post('/login', (req, res) => {
3   User.findOne({ username: req.body.username }, (err, user) => {
4     if (user == null) { ... }
5     const isValidPassword = utils.validPassword(
6       req.body.password,
7       user.hash,
8       user.salt);
9     if (isValidPassword) {
10       const tokenObject = utils.issueJWT(user);
11       return res.status(200).json({
12         success: true,
13         user: user,
14         token: tokenObject.token,
15         expiresIn: tokenObject.expires,
16       });
17     } else {
18       return res.status(401).json({ success: false, msg: 'Wrong password' });
19     });
});});
```

Ensuite, j'implémente le middleware de passeport dans mon app.js

```

1 //app.js
2
3 //API Route
4 app.use(
5   '/REST_API/api',
6   passport.authenticate('jwt', { session: false }),
7   require('./routes/app/api')
8 );

```

### Exemple avec le client ARC REST

Pour se connecter à notre application, un client fournit un json dans le corps de la requête. Une fois connecté avec succès, le client recevra un jeton qui expire dans 1 jour. Le client attachera ensuite ce jeton dans la partie autorisation de l'en-tête pour la prochaine demande d'accès aux routes protégées.

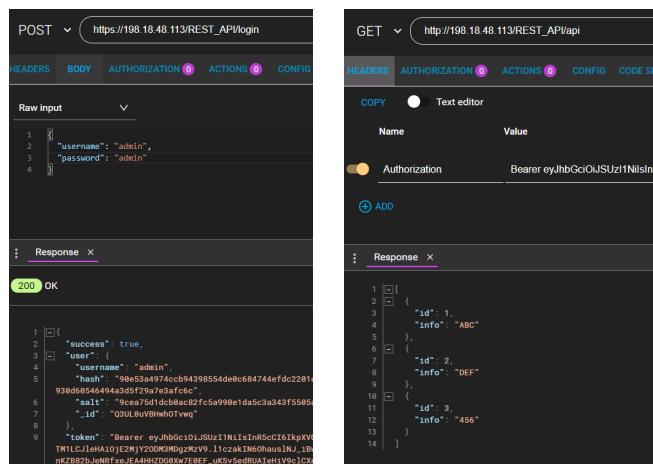


FIGURE 20 – Exemple de connexion en tant qu'administrateur pour obtenir toutes les données

Ne pas fournir le nom d'utilisateur/mot de passe lors de la connexion ou attacher un en-tête d'autorisation à la demande lors de l'accès aux routes protégées entraînera une erreur.

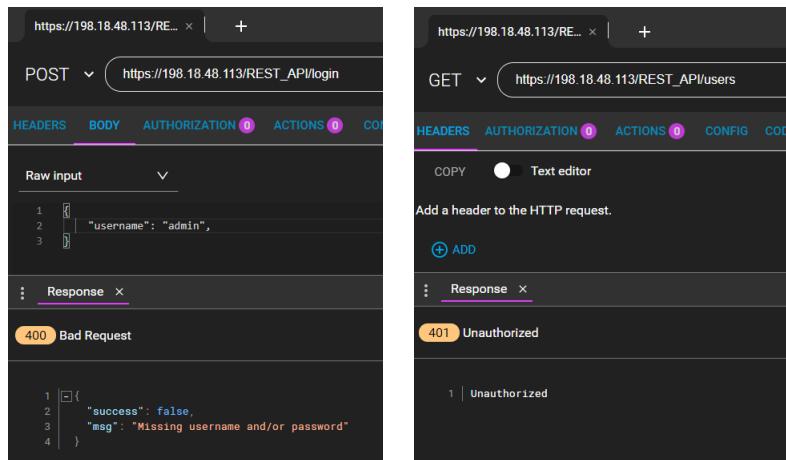


FIGURE 21 – Faire des erreurs lors de la connexion/de l'utilisation de l'API

**Conclusion** A partir de maintenant, les sections suivantes utiliseront la méthode JWT car c'est la meilleure option dans le contexte de CP9000.

### 3.5 La gestion des erreurs

Dans les sections précédentes, chaque fois que nous rencontrions une erreur, nous utilisions souvent deux manières de gérer cette erreur. Par exemple :

```

1 if (!err) return res.status(<HTTPcode>).json({ success:
2   false, msg: 'Failed' });
3 //ou
4 try {
5 } catch (error) return res.status(<HTTPcode>).json({
6   success: false, msg: 'Failed' });

```

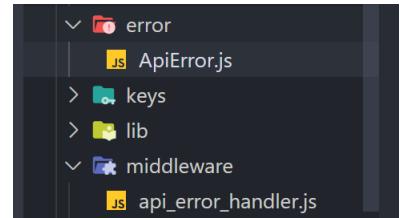


FIGURE 22 – Implémenter le gestionnaire d'erreurs

Cette façon de gérer est ok, ce n'est pas très mal mais quand notre application devient de plus en plus compliquée, elle devient difficile à maintenir, à gérer ainsi qu'à mettre à jour. C'est pourquoi nous avons besoin d'un middleware de gestion d'erreurs séparé.

<pre> 1 // error/ApiError.js 2 3 class ApiError { 4   constructor(code, message) { 5     this.code = code; 6     this.message = message; 7   } 8   static badRequest(msg) { 9     return new ApiError(400, msg); 10 } 11 static notFound(msg) { 12   return new ApiError(404, msg); 13 } 14 static internal(msg) { 15   return new ApiError(500, msg); 16 } 17 static unauthorized(msg) { 18   return new ApiError(401, msg); 19 } 20 } 21 module.exports = ApiError; </pre>	<pre> 1 // middleware/api_error_handler.js 2 3 const ApiError = require('../error/ 4   ApiError'); 5 const apiErrorHandler = (err, req, res, 6   next) =&gt; { 7   // Known errors 8   if (err instanceof ApiError) { 9     return res.status(err.code).json({ 10       success: false, 11       msg: err.message, 12     }); 13   } else { 14     // Unknown errors 15     res.status(500).json({ 16       success: false, 17       msg: 'Something went wrong', 18     }); 19   } 20   module.exports = apiErrorHandler; </pre>
--	---

J'ai créé une classe ApiError pour initialiser diverses erreurs rencontrées lors de l'exécution du programme. Alors que le middleware *api-error-handler* est un middleware de gestion des erreurs d'Express qui est chargé de vérifier s'il y a une erreur lors de la demande, il sera donc appelé dans app.js. Chaque fois que notre programme doit gérer une erreur, par exemple, lorsque nous nous connectons et que nous manquons le nom d'utilisateur et/ou le mot de passe, nous pouvons simplement transmettre ce middleware de gestion des erreurs avec *next()*.

```

1 // route/auth/users.js
2
3 router.post('/login', ...
4   if (!req.body.username || !req.body.password) {
5     next(ApiError.badRequest('Missing username and/or password'));
6     return;

```

### 3.6 Gestion des droits d'utilisateur

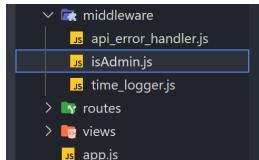


FIGURE 23 – Implément des rôles différents

Pour faciliter la gestion des applications, il est essentiel de définir les autorisations des utilisateurs, nous pouvons le faire en affectant à chaque utilisateur un rôle spécifique.

Dans ce projet, il y a 2 rôles différents : USER est un utilisateur normal qui peut se connecter à l'application, accès à la route protégée mais a des droits d'accès limités, ADMIN a tous les droits systèmes en accédant à la route users-management. ADMIN peut accéder à la base de données des utilisateurs du côté client ainsi que modifier les informations de n'importe quel compte.

URI	Méthode HTTP	POST body	Résultat
/users/	GET	NULL	Obtenir toutes les informations utilisateur
/users/<id>	GET	NULL	Obtenir des informations utilisateur spécifiques
/users/<id>	PUT	"password" :"nouveau mdp"	Changer un mot de passe utilisateur
/users/<id>	DELETE	NULL	Supprimer un compte utilisateur

La façon dont nous procérons est en fait assez simple, nous attribuons d'abord un rôle à chaque utilisateur et l'enregistrons dans la base de données.

```
1 // database/users.db
2 {"username": "user", "hash": "HashedPassword", "salt": "Salt", "role": "USER", "_id": "AdRY3WjPALgrR"}
3 {"username": "admin", "hash": "HashedPassword", "salt": "Salt", "role": "ADMIN", "_id": "QLwBlnJYGTXG"}
```

Lorsqu'un utilisateur se connecte avec succès, l'application passe à un middleware isAdmin pour vérifier le rôle de l'utilisateur. Ainsi, toutes les routes restreintes à l'administrateur devront passer par ce middleware, tout utilisateur n'étant pas l'administrateur sera bloqué par ce middleware.

```
1 // middleware/isAdmin.js
2 const isAdmin = (req, res, next) => {
3   if (req.user.role == 'ADMIN') next();
4   else return res.status(403).json({ msg: 'Only admins can access this route' });
5 };
6 module.exports = isAdmin;
```

Maintenant, je peux le vérifier en me connectant à l'aide d'un compte administrateur et d'un compte utilisateur. Chaque compte me donne un jeton, je vais ensuite utiliser ce jeton pour accéder à la route pour l'administrateur uniquement.

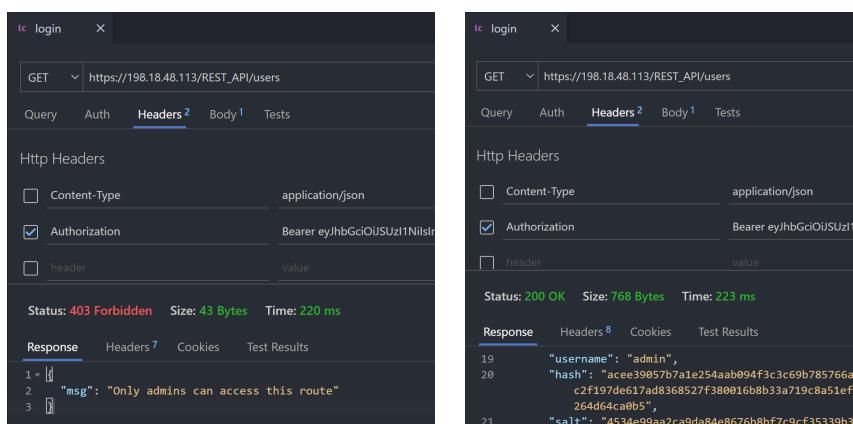


FIGURE 24 – Utilisation d'un jeton pour l'utilisateur et pour l'administrateur afin d'accéder à la route restreinte

## 3.7 Intégration de API REST au framework héritage CP9000

### 3.7.1 La structure du CP9000

Globalement, la structure de ce produit est un système de supervision. La machine combine une partie "coeur" et les autres composants installés pour fonctionner avec ce noyau. Le cœur de l'application fait tout le travail et diadoques avec les autres parties comme la carte de codage vidéo, la porte de sortie vidéo. En termes de supervision, il est intégré avec une petite interface LCD qui peut faire les communication basique et connecté via USB. Le cœur est aussi capable de générer plusieurs interfaces de management comme l'interface de web, de SNMP. Il y a aussi une partie de test, de debug, de maintenance et de domotique qui travaille avec un protocole TCP particulier pour installer, faire les configurations de bas.

Ce qui nous importe le plus sera la partie de l'équipement de management : Le web browser qui communique avec le cœur par HTTP et SNMP client qui communique par SNMP (Simple Network Management Protocol), une structure hiérarchique qui fonctionne comme une API.

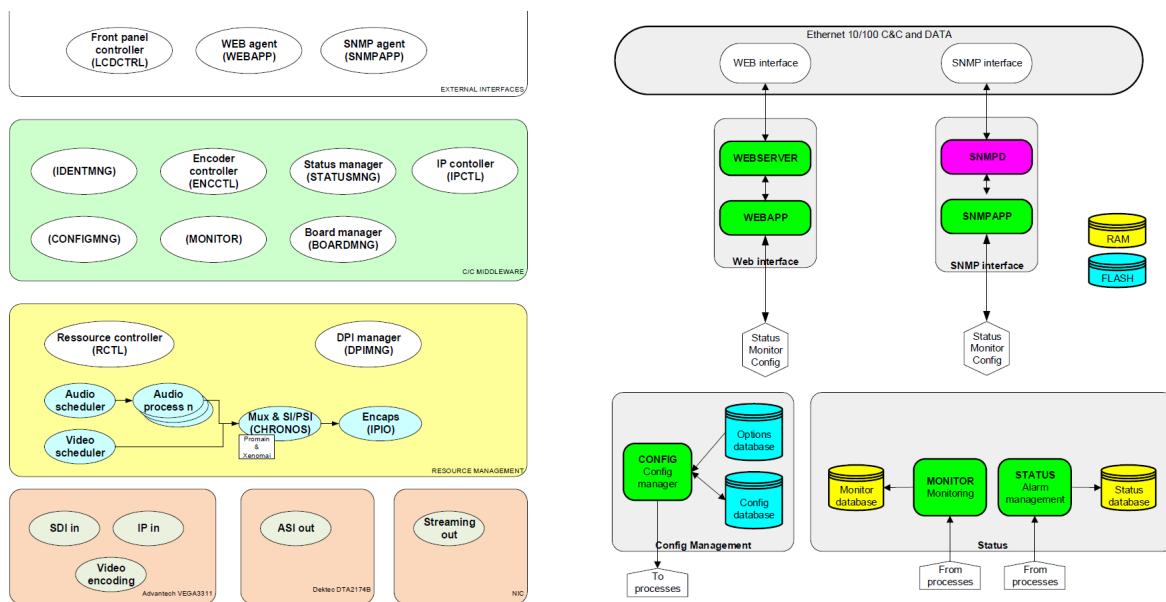


FIGURE 25 – Architecture de supervision de CP9000 détaillée

On a une partie middleware qui génère la configuration. Cette partie contient plusieurs modules différents, chacun d'eux a un rôle spécifique.

- CONFIGMNG (Config Manager) qui gère tous les accès à la base de données en termes de configuration NCCP. CONFIGMNG gère donc tous la partie de configuration et les autre montrent plus des informations sur l'état de la machine. Il va falloir créer la possibilité d'accéder cette partie depuis l'agent Web (WEBAPP).
- IDENTMNG (Ident Manager) qui fait le structuration les codes. Il lire toute la numérotation de hardware comme le numéro de série, le type de components, etc. On doit mettre aussi le web agent pour cette partie pour savoir qui passe.
- STATUSMNG (Status Manager) qui joue le même rôle avec IDENTMNG mais pour les alarmes. Il gère tous les alarmes de l'équipement et les synthétique. Il va aussi permet de voir les alarmes sur WEB agent.
- BOARDMNG (Board Manager) qui gère les version des logiciels
- MONITOR qui fait du moniteur et gérer tous les compteurs de la performance de la machine.
- ENCCTL (Encoder Controller) qui décortique entre les parties différentes de la machine . A chaque fois, quand il y a un configuration qui bouge, il va le décortique et suivre à quel operation qu'il bouge, puis s'addresser aux autres parties pour reconfigurer l'équipment.

Dans la partie Resource Management, il y a quelques composants comme :

- DPIMNG (DPI Management) qui contrôle de pub. Il exploite directement des informations relatives du système d'automation, de commutation, etc
- IPCTL (IP controller) qui contrôle tous les aspects réseaux d'interface d'entrée/sortie pour transfert les packages.
- RCTL (Ressource controller) qui contrôle tout le reste comme le décodage, encodage, transcoding vidéo et les signaux qui arrivent

A RCTL, le planificateur audio est passé par un processus audio, puis sera synchronisé (CHRONOUS) avec le planificateur vidéo. Il y a aussi une partie SI/PSI qui aider dans télévision les chaînes différentes. Tout cela va passer par l'encapsulation (IPIO), qui a les IPs sortent de IP Controller. Globalement, quand on fait une configuration par le web, cela passe par le WEB agent, arrive vers config manager où on fait toutes les étapes cohérentes et il met la base de données internes NCCP. Ensuite, cela revient sur Encoder Controller pour vérifier. Dans le cas que c'est une configuration réseau, il va passer IP Controller pour reconfigurer et puis passer à IPIO. Si c'est de type encodage, il passe de ENCCTL à Ressource Controller qui va programmer le processeur audio/vidéo. Statut manager va récupérer toutes les alarmes pendant cette période.

### 3.7.2 Manipulation des structures de donnée NCCP.

Tout d'abord, pour pouvoir récupérer chaque paramètre séparément par add-ons dans Node.js, nous devons créer une classe abstraite qui inclut des méthodes virtuelles et fonction comme une interface pour accéder aux autres classes. Dans chaque méthode de cette classe, nous passerons une clé, cette clé correspond au ce que nous devons mapper par envoyer un pointeur sur l'object qui nous intéresse et dans l'argument restant, nous passerons les données d'entrée.

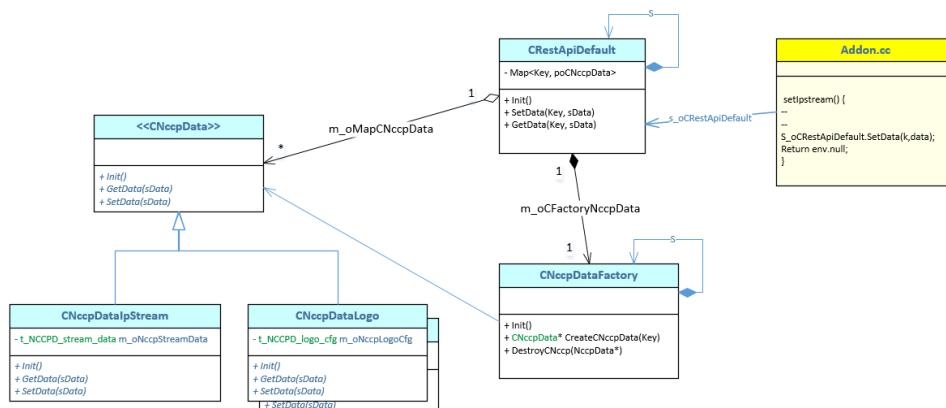


FIGURE 26 – Génération de paires de clés dans un projet node.js

De la part de la classe CNccpData, il y aura 2 méthodes SetData et GetData et il va rappeler automatiquement les méthodes polymorphes correspondantes.

```

1 class CNccpData
2 {
3     public:
4         void init();
5         virtual int getdata(const unsigned int, std::string[]);
6         virtual int setdata(const unsigned int, const std::string[]);
7     };
  
```

### 3.7.3 GET/SET adresse IP et port UDP source en SMPTE2022-6

**Implémentation de classe CNCCP\_Data** Dans le système hérité du CP9000, les informations sont stockées sous forme de tableau. Et avec ces deux tableaux *t\_NCCP\_stream* et *t\_NCCP\_logo* que nous devons faire GET et SET, nous pouvons observer qu'ils partagent une structure similaire. Pour cette raison, je vais décrire comment je le fais avec *t\_NCCP\_stream*, avec *t\_NCCP\_logo* l'application est la même.

```
1 typedef struct
2 {
3     unsigned char stream_enable;
4     uint32_t IP_subscribe_address;
5     unsigned short UDP_subscribe_port;
6     uint32_t source_IP_address;
7     unsigned char FEC_enable;
8     unsigned char physical_port;
9 } t_NCCPD_stream;
10
11 typedef struct
12 {
13     unsigned char logo_number;
14     unsigned char FTP_upload_enable;
15     char logo_pathname[  
16         MAX_SIZE_LOGO_PATHNAME + 1];
17     char logo_filename[  
18         MAX_SIZE_LOGO_FILENAME + 1];
19 } t_NCCPD_logo;
```

Nous créons donc une classe `Ipstream` et implémentons les méthodes nécessaires. On les déclare dans `CIpstream.h` et les définir dans `CIpstream.c`.

```
1 class CIpstream
2 {
3 private:
4     unsigned int ipstream_id;
5     t_NCCPD_stream m_oNCCPDstream;
6 public:
7     CIpstream();
8     ~CIpstream();
9     const unsigned char get_stream_enable() const { return m_oNCCPDstream.stream_enable; };
10    void set_stream_enable(const unsigned char str_en) { m_oNCCPDstream.stream_enable = str_en
11        ; };
12    ...
13    const unsigned int get_ipstream_id() const { return ipstream_id; };
14    void set_ipstream_id(const unsigned int id) { ipstream_id = id; };
15};
```

Puis dans addons de NodeJs, on définit le méthode GET pour retourner les données sur le type table en JavaScript avec le premier élément est l'id de l'*Ipstream* que nous voulons et pour SET on veut modifier les données un *Ipstream*. Ici, je montre par exemple le SET pour modifier un *Ipstream* dans *addons.cc*.

```
1 // Set_ipstream()
2
3     string data[6];
4     unsigned int id = (unsigned int)info[0].ToNumber();
5     for (int i = 0; i < 6; i++) data[i] = (string)info[i + 1].ToString();
6     s_oCRestapiDefault.setdata(id, data);
7     return env.Null();
```

Enfin, simplement on peut rappeler cette fonction dans notre route en JavaScript.

```
// routes/app/api.js
2
3 router.put('/controller/ipstream/:id', ....
4   addon.setIpstream(
5     parseInt(req.params.id),
6     newData.stream_enable,
7     newData.IP_subscribe_address,
8     ...
9  ));

```

Nous pouvons le tester en envoyant la demande à notre serveur.

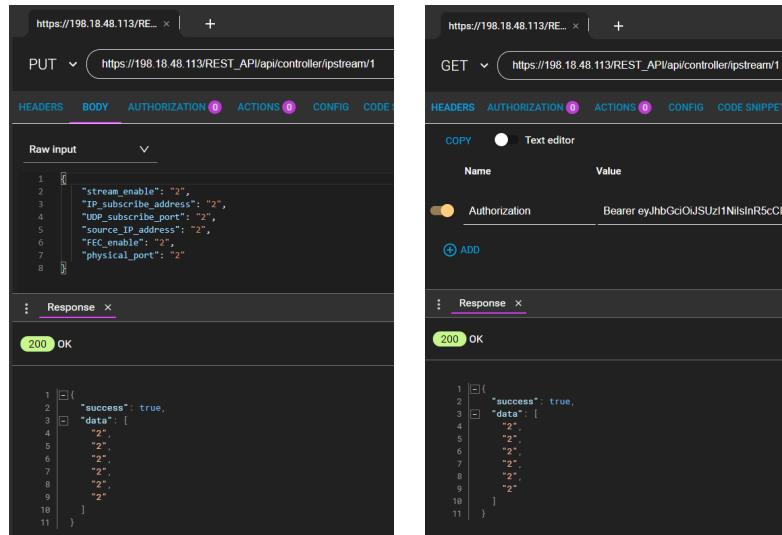


FIGURE 27 – Test de SET et GET de l'ipstream ID 1

### 3.8 Documentation Api

Swagger est un langage de description d’interface pour décrire les API RESTful exprimées à l'aide de JSON. Swagger inclut une documentation automatisée, la génération de code et la génération de cas de test. Pour documenter notre API, j’ai créé une interface web pour interagir directement avec l’API via l’UI Swagger. Cela permet une connexion à l’API en direct via une interface utilisateur interactive basée sur HTML. Les requêtes peuvent être effectuées directement depuis l’interface utilisateur et les options explorées par l’utilisateur de l’interface.

```
1 // Extended: https://swagger.io/specification/
2 const swaggerDocs = require('./swagger.json');
3 app.use('/REST_API/api-docs', swaggerUi.serve, swaggerUi.setup(swaggerDocs));
```

FIGURE 28 – Documentation API avec Swagger

## 4 Conclusion

Après le stage de 6 mois, j'ai globalement atteint la plupart des objectifs initialement proposés, cependant, en raison des difficultés de la période Covid, je n'ai pas pu terminer la mise en place de toutes les structures du cadre hérité. .

Les principales tâches réalisées sont le développement d'une API REST adaptée sur la machine CP9000 avec des méthodes de sécurité ainsi que la gestion des exceptions, l'hébergement de l'API à la société Nginx reverse-proxy, l'intégration de l'API pour travailler avec les modules C/C++ existants.

Grâce à ce stage, j'ai acquis beaucoup d'expérience sur les techniques de programmation sous Linux avec JavaScript et C/C++, le processus d'intégration, la manière de modularisation et la construction sur un standard de l'industrie, et aussi la compilation croisée avec srv0359. Ce stage m'a permis de préparer les connaissances, les compétences indispensables pour mes projets professionnels dans l'avenir.

## Références

- [1] Node.js in Practice (2014) *Book*, Alex Young and Marc Harter
- [2] Web API Design : The Missing Link - Crafting Interfaces that Developers Love (2018) *Apigee. eBook*, Mulloy, Brian.
- [3] Node.js v16.6.1 documentation *Documentation*, OpenJS Foundation, Joyent, Inc.
- [4] V8 9.0.257(node16.0.0) documentation *Documentation*, Google's open source JavaScript engine
- [5] NeDb documentation <https://dbdb.io/db/nedb> *Documentation*, Louis Chatriot
- [6] Passport.js documentation <http://www.passportjs.org/docs/> *Documentation*, Jared Hanson
- [7] Swagger documentation <https://swagger.io/specification/> *Documentation*, © 2021 SmartBear Software