Internet des objets: Proxy pour Caméra IP

Approche sur la sécurité et implémentation d'un projet pratique

Travail de Bachelor

Loïc Rosset

Décembre 2018

Supervisé par :

Prof. Dr. Jacques Pasquier-Rocha
et
Pascal Gremaud
Software Engineering Group

UNIVERSITÉ DE FRIBOURG
UNIVERSITÄT FREIBURG

Groupe Génie Logiciel
Département d'Informatique
Université de Fribourg (Suisse)

Software Engineering Group

Remerciements

Je tiens à remercier mes proches pour leur soutien, ainsi que l'équipe du groupe génie logiciel, principalement Pascal Gremaud pour sa disponibilité et son aide tout au long de ce travail. Je remercie aussi l'entreprise de sécurité qui m'a proposé ce projet.

Résumé

Ce travail de Bachelor s'insère dans l'air de l'Internet of Things et du Web of Things, révolutionnant continuellement le monde actuel. En effet, de nombreuses entreprises se sont créées autour de ces deux notions, en produisant ou en distribuant sur le marché des objets connectés de toute sorte.

Une partie de ces objets touche directement à la sécurité, c'est pourquoi, après une introduction sur l'impact des objets connectés sur celle-ci, différents protocoles tels que SSL et SSH sont étudiés.

Finalement, un projet proposé par une entreprise de sécurité locale a été réalisé de manière indépendante dans le cadre de ce travail. Celui-ci permet de comprendre les notions discutées précédemment dans un cas pratique. L'objectif de ce dernier est d'implémenter une architecture utilisant un Raspberry Pi comme boîtier installé chez le client et permettant de proxifier de manière sécurisée, par des tunnels SSH, le flux vidéo d'une caméra IP sur un serveur externe afin qu'un utilisateur puisse le visualiser peu importe sa localisation. Le serveur regroupe trois modules : une API RESTful implémentant le framework Koa et communiquant avec une base de données MongoDB au travers de Mongoose, un site web implémentant le framework Vue.js et un serveur Websocket. L'application sur le boîtier utilise le langage Java.

Keywords : Internet of Things, Sécurité, Camera IP, Proxy, REST, Koa, Node.JS, Javascript, MongoDB, Mongoose, Vue.JS, SSH, SSL, Reverse Tunneling, Java, Raspberry Pi 3, Webpack, Websocket

Table des matières

Liste des figures

Liste des tableaux

Liste des codes source

1
Introduction

1.1. Contexte, motivation et objectifs

Dans une ère où l'informatique s'impose de plus en plus dans nos vies, il est bon de se poser certaines questions et d'acquérir des connaissances permettant de comprendre les mécanismes qui résident derrière ces objets connectés qui prennent de l'ampleur dans notre quotidien. Comment se connectent-ils ? Quelles informations transmettent-ils ? Est-ce sécurisé ?

Ce travail va apporter des éléments de réponses à ces questions, en premier de manière théorique, puis de manière pratique au travers d'un projet allant d'un bout à l'autre : de l'objet connecté à une interface utilisateur sur un site web.

1.2. Organisation

Chapitre 1 : Introduction

L'introduction contient les motivations et objectifs de ce travail, une rapide récapitulation de la structure de chaque chapitre ainsi qu'une présentation des conventions de format.

Chapitre 2 : IoT et Sécurité

Ce chapitre décrit le lien et les impacts de l'Internet of Things sur la sécurité et regroupe divers protocoles et technologies utilisées pour la maintenir.

Chapitre 3 : Projet - Proxy pour Caméra IP

Dans un deuxième temps, un projet concret en rapport avec la sécurité, intégrant des caméras IP, est décrit et mis en œuvre. L'objectif de celui-ci est de proxifier le flux vidéo d'une caméra sur un serveur afin qu'un utilisateur puisse y accéder peu importe sa localisation.

Chapitre 4 : Conclusion

La conclusion synthétise le travail effectué, décrit la contribution que celui-ci peut apporter au domaine de recherche ainsi que les défis rencontrés personnellement.

Appendix

Contient une liste des acronymes employés ainsi que les références utilisées tout au long de ce travail.

1.3. Notations et Conventions

— Conventions de format :

  — Les abréviations et acronymes sont sous la forme Hypertext Transfer Protocol (HTTP) pour le premier usage et HTTP pour les suivants ;

  — `http://localhost:8888` est utilisé pour les adresses web ;

  — Le code est formaté de la façon suivante :

```
1 public double division(int _x, int _y) {
2     double result;
3     result = _x / _y;
4     return result;
5 }
```

— Le travail est séparé en 4 chapitres, eux-mêmes subdivisés en sections, sous-sections et sous-sous-sections. Ces dernières sont organisées en paragraphes, signifiant des coupures logiques.

— Les objets de type Figure, Table et Listing sont numérotés au sein des chapitres. Par exemple, une référence à Figure j du chapitre i aura la notation *Figure i.j.*

— En ce qui concerne le genre, la forme masculine est constamment employée par simplicité. Les deux genres sont considérés de façon égale.

2
IoT et Sécurité

2.1. IoT

L'Internet of Things (IoT) possède de nombreuses définitions. On peut résumer l'IoT a un réseau d'objets connectés capables, grâce à des logiciels, d'interagir entre eux. Parmi ces objets on retrouve des voitures, des montres, des lunettes, des téléphones, des drones mais aussi d'autres plus sensibles comme un pacemaker ou des caméras de sécurité.

Ceux-ci sont dits intelligents car ils sont dès lors capables de récolter des données, les analyser, s'y adapter ou les envoyer vers d'autres systèmes informatiques qui les enregistrent ou les distribuent encore plus loin.

Certaines limitations s'envolent alors. En effet, on peut par exemple contrôler une voiture à distance. Ceci est rendu possible par l'utilisation de technologies premièrement destinées au Web. On parle alors de Web of Things (WoT). [1]

2.2. Sécurité

La connexion à ces objets ainsi que la sensibilité des données qu'ils peuvent transmettre peut alors inquiéter bon nombre de personnes, et ce à juste titre.

Prenons le cas de Jeep. En 2015, Charlie Miller et Chris Valasek expliquaient en détails comment ils avaient pris le contrôle d'une Jeep Cherokee et l'avaient, à distance depuis leur maison, arrêtée au milieu de l'autoroute.[2]

Un autre cas très sensible et médiatisé est celui des pacemakers. Ceux-ci peuvent aussi être piratés et des hackers l'ont démontré lors de la conférence *Black Hat* de 2018. Il est alors possible de vider les piles ou d'infliger une décharge mortelle à un patient.[3]

Il n'est alors pas étonnant que les coûts liés à la cybercriminalité deviennent de plus en plus élevés. Ces coûts s'élèvent à 600 billions de dollars par année dans le monde.

[4] Le nombre croissant des objets connectés, estimés à 20 billions en 2020 [5], y joue certainement un rôle.

On se pose donc la question : comment sécuriser ces objets et les données qu'ils transmettent ? Le plus important est de sensibiliser les utilisateurs et les développeurs aux attaques potentielles et aux moyens de s'en prémunir. Gary Sims explique dans son article [6] qu'une attaque Distributed Denial of Service (DDoS) menée contre Dyn [1], un fournisseur DNS pour Twitter, SoundCloud, Spotify, Reddit et d'autres sites célèbres, avait été menée à l'aide de caméras de sécurité et d'autres objets de stockage reliés à des réseaux. Ceci a été rendu possible grâce à un logiciel malveillant qui essaie de se connecter aux objets connectés qu'il trouve en utilisant des combinaisons d'identifiants et de mots de passe basiques, pour s'infiltrer et prendre possession de leurs ressources. Gary propose alors une liste de points à vérifier :

- Authentification : ne jamais créer un produit avec un mot de passe par défaut qui est le même sur tous les objets. Chaque objet devrait avoir un mot de passe complexe.
- Debug : ne jamais laisser un accès pour débugger un objet.
- Chiffrement : toutes les communications entre les objets connectés et le *cloud* doivent être cryptées.
- Privatisation des données : ne pas laisser un accès direct à des données sensibles, toujours les stocker de manière chiffrée.
- Interface web : toute interface web devrait être protégée contre des techniques de piratage standard.
- Mise à jour des logiciels : tout objet connecté devrait être capable de recevoir des mises à jour à distance.

Cette liste fait intervenir l'importance du cryptage des communications et des données. Il existe plusieurs protocoles qui permettent de crypter les communications, Secure Shell (SSH) et Secure Socket Layer (SSL) en font partie.

2.2.1. SSH

Dans le Request For Comments (RFC) 4253 , SSH est décrit comme un protocole qui permet de se connecter à distance de manière sécurisée au travers d'un réseau non sécurisé. Le protocole peut être utilisé comme base pour de nombreux services de réseaux sécurisés. Il offre un fort cryptage, une authentification sur les serveurs et une protection de l'intégrité. Plusieurs méthodes sont réalisées pour se faire : un échange de clés, l'utilisation de l'algorithme de clé publique, de cryptage symétrique, d'authentification des messages et de hachage. [7]

C'est donc un protocole très utile pour créer des connexions sécurisées au travers de réseaux non sécurisés. Celui-ci est mis en application au point 3.3.4.

2.2.2. SSL/TLS, HTTPS et Let's Encrypt

En 1996, la version 3.0 du protocole SSL a été publiée. Les navigateurs courants ne supportent plus la version 2.0. Il est défini dans le RFC 6101 comme protocole en couche.

[1] https ://dyn.com/

A chaque couche des messages incluent des champs pour la longueur, la description et le contenu. Le protocole se charge de prendre les messages à transmettre, de les fragmenter en blocs (les compresse éventuellement) puis y ajoute un MAC, les crypte et les transmets. Les données reçues sont ensuite à l'inverse, décryptées, vérifiées, décompressées et rassemblées pour être transmises à des clients de plus haut niveau.[8]

Quand un client et un serveur commencent à communiquer avec le protocole SSL, ils font un *handshake*, pour définir plusieurs paramètres dont notamment le manière dont est cryptée la communication et aussi pour s'authentifier grâce à des certificats. Ceux-ci sont délivrés par des Certificate Authority (CA). Let's Encrypt [2] est un CA qui délivre gratuitement et de manière automatisée des certificats, ce qui permet à tout un chacun de sécuriser ses communications.

On fait aussi souvent mention de TLS. Ce protocole est simplement le successeur de SSL. De manière générale, ces deux protocoles sont utilisés dans le protocole HTTPS qui est la version sécurisée de HTTP, dont la définition est donnée au point 3.2.1.

Ces protocoles sont utilisés dans le projet réalisé et présenté dans le chapitre 3.

[2] `https://letsencrypt.org/`

3

Projet - Proxy pour Caméra IP

3.1. Description du projet

Une entreprise de sécurité locale, initiatrice de ce projet, propose une solution de vidéo hébergée dans le cloud. Elle travaille avec les caméras AXIS. Ces caméras intègrent un système "one click" qui permet aux caméras de se connecter de manière autonome sur un serveur de streaming de l'entreprise sans aucune configuration requise sur la caméra. L'entreprise aimerait développer un système similaire mais ouvert à tout type de caméra ou autre périphérique connecté.

Le but du projet est de développer un boîtier qui, une fois installé chez les clients, a comme tâche de se connecter à un serveur dans le cloud et de faire proxy pour les caméras en local. De ce fait, lorsqu'un client se connecte sur la plateforme de l'entreprise, peu importe sa localisation dans le monde, il devrait être capable de voir sa caméra en direct.

3.1.1. VSaaS

Un Video Surveillance as a Service (VSaaS) permet à des clients du service d'enregistrer, de visualiser, et de manager des flux de caméras de sécurité à distance et ce de manière sécurisée, entièrement dans le *cloud*. C'est un marché très concurrentiel qui est estimé valoir 48,95 billion de dollar en 2020. [10]

Ce projet est donc l'ébauche d'un VSaaS, qui permet d'étendre les produits de l'entreprise actuellement limitée à l'utilisation de caméras AXIS. Dans le cadre de ce travail, un prototype est réalisé.

3.1.2. Architecture initiale

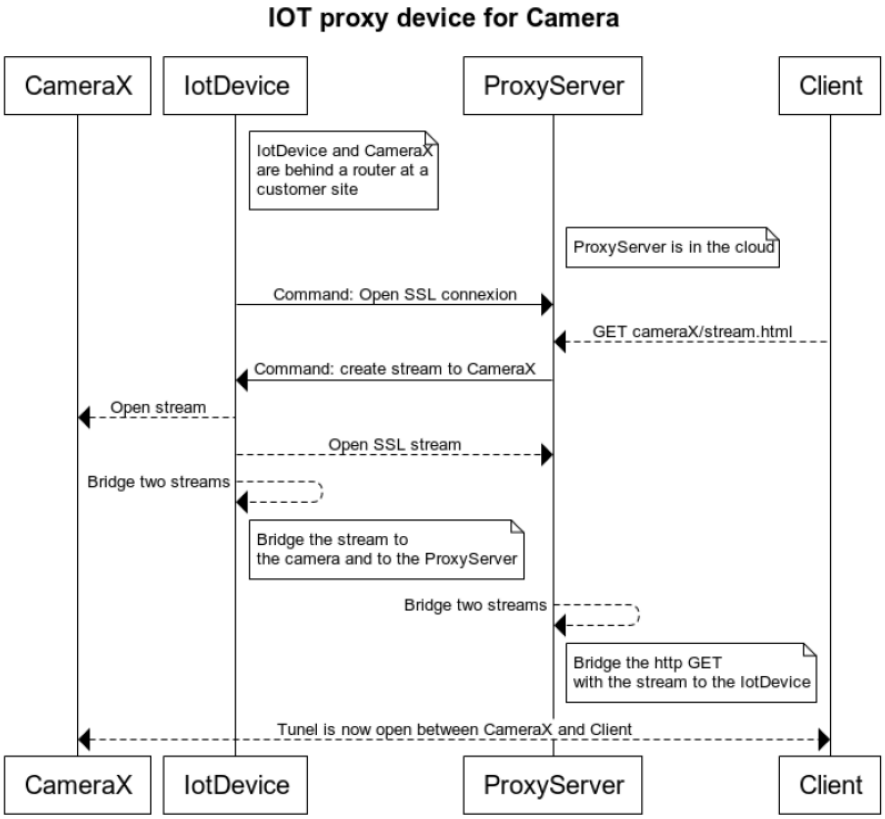

Figure 3.1. – Diagramme de séquence fourni par l'entreprise

Dans le mandat initial de l'entreprise, illustré dans la figure 3.1, l'idée est de permettre à un client de faire une requête sur un serveur dans le cloud pour avoir accès à sa caméra. Le client peut alors se trouver n'importe où et simplement interagir avec ce serveur. La caméra, elle, se trouve dans un réseau privé, au domicile du client, donc derrière un routeur. Afin d'éviter toute configuration et de permettre d'utiliser n'importe quel type de caméra, il est nécessaire d'utiliser un intermédiaire entre la caméra et le serveur, c'est ce qui est décrit par "IoT Device", qu'on va nommer boîtier IoT pour la suite. Celui-ci se charge d'écouter les requêtes provenant du serveur et de les exécuter. De plus, il permet de traverser le Network Address Translation (NAT) par le biais du protocole Websocket et des tunnels SSH, cette problématique est décrite plus en détail au point 3.3.4.

Une demande de connexion au flux d'une caméra provoque ainsi l'ouverture d'un tunnel entre la caméra et le serveur par l'intermédiaire du boîtier. On juge suffisant de ne pas pousser le tunnel jusqu'au client, et de créer une interface utilisateur sur le serveur, par le biais d'un site web. Pour information, le nom de domaine de ce serveur est `camera-stream.tk`. L'architecture du réseau est illustrée dans la figure 3.2.

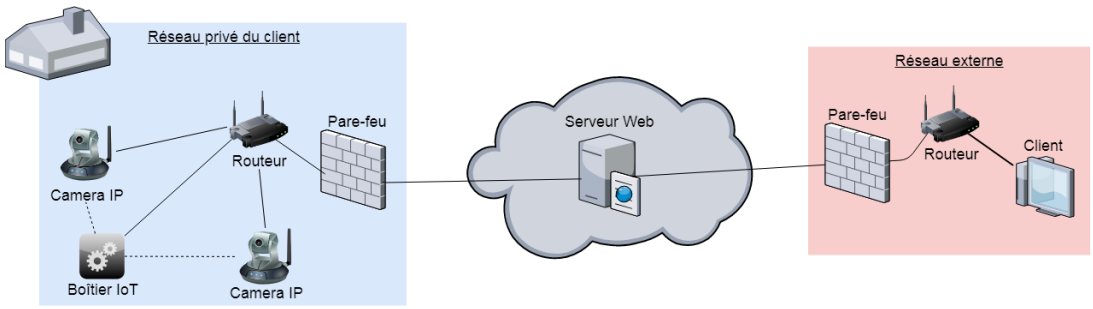

Figure 3.2. – Architecture du réseau

3.1.3. Les langages et les framework

Ce projet nécessite le développement de plusieurs parties, représentées dans la figure 3.3 :

— L'application sur le boitier IoT : cette application doit communiquer avec le serveur et gérer les tunnels avec les caméras IP du réseau local. Java est le langage qui a été choisi, par familiarité.

— Le *backend* du serveur : celui-ci doit offrir une API RESTful, c'est-à-dire qui utilise les contraintes Representational State Transfer (REST)[19]. L'avantage d'une API REST est que si l'on décide d'étendre le système avec une application mobile par exemple, ou tout autre système, on peut alors sans autre développer l'extension en respectant les contraintes REST et utiliser le même *backend*. Cette API permettra de communiquer avec une base de données MongoDB [1], en utilisant en parallèle Mongoose [2]. MongoDB est un système de gestion de base de données orientée NoSQL et Mongoose un outil permettant de gérer plus facilement les données destinées à y être stockées. Une description reprend en détails ces technologies au point 3.4.2.

  Pour tout cela, Node.JS [3] est parfaitement adapté. En effet, cet environnement de développement pour serveur est open source et propose une multitude de modules qui le rendent très extensible. Le code s'écrit alors en Javascript. De plus, le framework Koa [4], développé par l'équipe qui a travaillé sur le framework Express, offre un bon point de départ. Une autre partie du *backend* est aussi consacrée au serveur Websocket décrit en détail aux points 3.3.1 et 3.4.1.

— Le frontend du serveur : destiné au client, il doit permettre de se connecter à un compte et de visualiser les caméras possédés, puis de s'y connecter. Plusieurs frameworks existent pour le développement d'une application web de ce type, il y en a

[1] `https://www.mongodb.com/fr`
[2] `https://mongoosejs.com/`
[3] `https://nodejs.org/`
[4] `https://koajs.com/`

trois qui se démarquent particulièrement : Angular, React et Vue.js. Pour ce projet Vue.js [5] est utilisé et décrit au point 3.5.1.

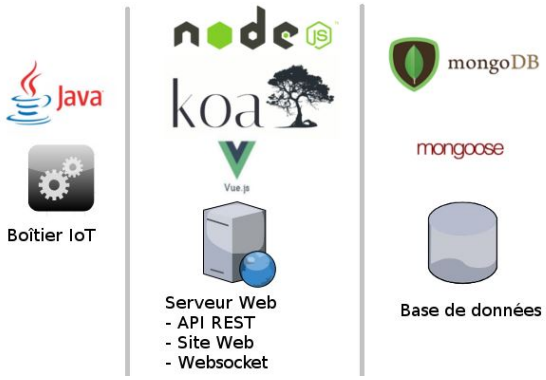

Figure 3.3. – Les différentes parties du projet

3.2. Choix du matériel

3.2.1. Les caméras IP

Il existe plusieurs types de caméras Internet Protocol (IP). Elles se différencient par leurs caractéristiques mais toutes ont en commun la capacité de se connecter à un réseau. Ce réseau peut être publique ou privé.

Les caméras IP sont généralement utilisées pour la surveillance et peuvent être centralisées avec un Network Video Recorder (NVR) ou décentralisées de manière à enregistrer les données directement sur un espace de stockage local ou sur le cloud.

Les caractéristiques des caméras IP sont multiples :

- Design et types : boitier fixe, modulaire, panoramiques, thermiques, embarquées
- Power over Ethernet (PoE) : permet de ne plus avoir besoin d'un câble d'alimentation externe, l'alimentation vient du câble Ethernet
- Pan-Tilt-Zoom (PTZ) : capacité de pilotage à distance, zoom
- Connexion : Wifi, 4G, Ethernet
- Distributed artificial intelligence : possibilité de la caméra à faire différentes opérations sur les images (détection de mouvement, reconnaissance faciale...)
- Alarme, microphone et haut-parleur intégrés
- Lampe

Avantages

Les avantages, comparés aux caméras analogues, sont donc évidents et nombreux. Par exemple, l'architecture classique pour un système de surveillance utilisant des caméras analogues utilisent des Digital Video Recorder (DVR), là où le flux vidéo est traité. Les caméras IP, quant à elles, traitent directement le flux vidéo et l'envoient éventuellement à

[5] `https://vuejs.org/`

un NVR. L'avantage dans le deuxième cas est que l'on peut avoir un seul NVR sur le réseau tandis que les caméras analogues doivent être directement connectées au DVR. Ainsi il est peut-être nécessaire d'avoir plusieurs réseaux de caméras suivant les emplacements, et donc plusieurs DVR.

Inconvénients

Les inconvénients liés à l'utilisation d'une caméra IP reposent principalement sur les problèmes de sécurité. En effet, une caméra IP est directement connectée à un réseau et est donc plus sujette au hacking. Mais d'autres paramètres sont aussi à prendre en compte et peuvent poser problème : avoir une adresse IP statique ou dynamique, par exemple, peut poser des problèmes de configuration et nécessitera peut-être un DynDNS, permettant de rediriger un Domain Name System (DNS) vers la caméra même si celle-ci change d'adresse. Les caméras IP sont aussi dépendantes du réseau : si le routeur s'avère avoir un problème, les caméras ne fonctionneraient plus tandis que des caméras analogues continueraient à filmer.

Pour ce projet

Il n'y a pas besoin de choisir une caméra spécifique pour ce projet comme le but est de permettre une connexion sur n'importe quel type de caméra IP. Une caméra AXIS prêtée par l'entreprise est donc utilisée, elle est illustrée à la figure 3.4.

Figure 3.4. – Caméra IP AXIS

L'idée est de créer un tunnel entre la caméra et le serveur et pour cela l'adresse IP de la caméra est nécessaire ainsi que le port auquel on désire accéder. Ce qui change selon les différents types de caméra, c'est l'accès au flux vidéo de celle-ci. En effet, il existe des Uniform Resource Locator (URL) d'accès différents pour chaque type ou marque de caméra. Certains d'entre eux sont relevés dans le tableau 3.1.

Camera	**URL**	**Protocole**	**Type**
AXIS	/axis-cgi/mjpg/video.cgi	HTTP	FFMPEG
AXIS	/axis-media/media.amp	RTSP	FFMPEG
SAMSUNG	/now.jpg	HTTP	JPEG
SAMSUNG	/onvif/profile1/media.smp	RTSP	FFMPEG
LOGITECH	/videofeed	HTTP	MJPEGE
LOGITECH	/ ?action=stream	HTTP	MJPEG

Table 3.1. – URL d'accès au flux vidéo de caméras IP

Il est à noter que ces URL peuvent encore différer selon les modèles de caméras au sein d'une même marque. De plus, on voit qu'il existe deux types de protocoles : Hypertext Transfer Protocol (HTTP) et Real Time Streaming Protocol (RTSP).

— HTTP : est un protocole de la couche application. C'est un protocole générique qui peut être utilisé, au travers d'extensions de son système de requête, d'entêtes, et de codes d'erreurs, pour des tâches qui dépassent la simple utilisation d'hypertexte. Une particularité de ce protocole est de pouvoir gérer et typer des représentations de données, ce qui permet à des systèmes d'être construits indépendamment du type de données transmises. [11] C'est le cas pour les caméras IP qui transmettent leurs images de cette manière.

— RTSP : est un protocole de la couche application qui permet de préparer et contrôler le transfert de données, tel que de l'audio ou de la vidéo, avec des propriétés en temps réel. Les sources peuvent être issues d'un transfert en direct ou de fichiers stockés. Le but du protocole est de permettre le contrôle de plusieurs sessions de transfert et de pouvoir en choisir le canal (UDP, TCP, ...) ainsi que la méthode (parmi un choix de méthodes basées sur le protocole RTP). [12]

3.2.2. Le boîtier IoT

Concernant le boîtier à placer chez le client, il faudra qu'il soit capable de se connecter au réseau et de lancer une application. Plusieurs petits ordinateurs à bas prix correspondent à ces critères, pour ce projet nous utilisons un Raspberry Pi 3 illustré dans la figure 3.5 : Quad Core 1.2GHz Broadcom BCM2837 64bit CPU, 1GB RAM, BCM43438 wireless LAN and Bluetooth Low Energy (BLE) on board. L'Operating System (OS) Raspbian est utilisée.

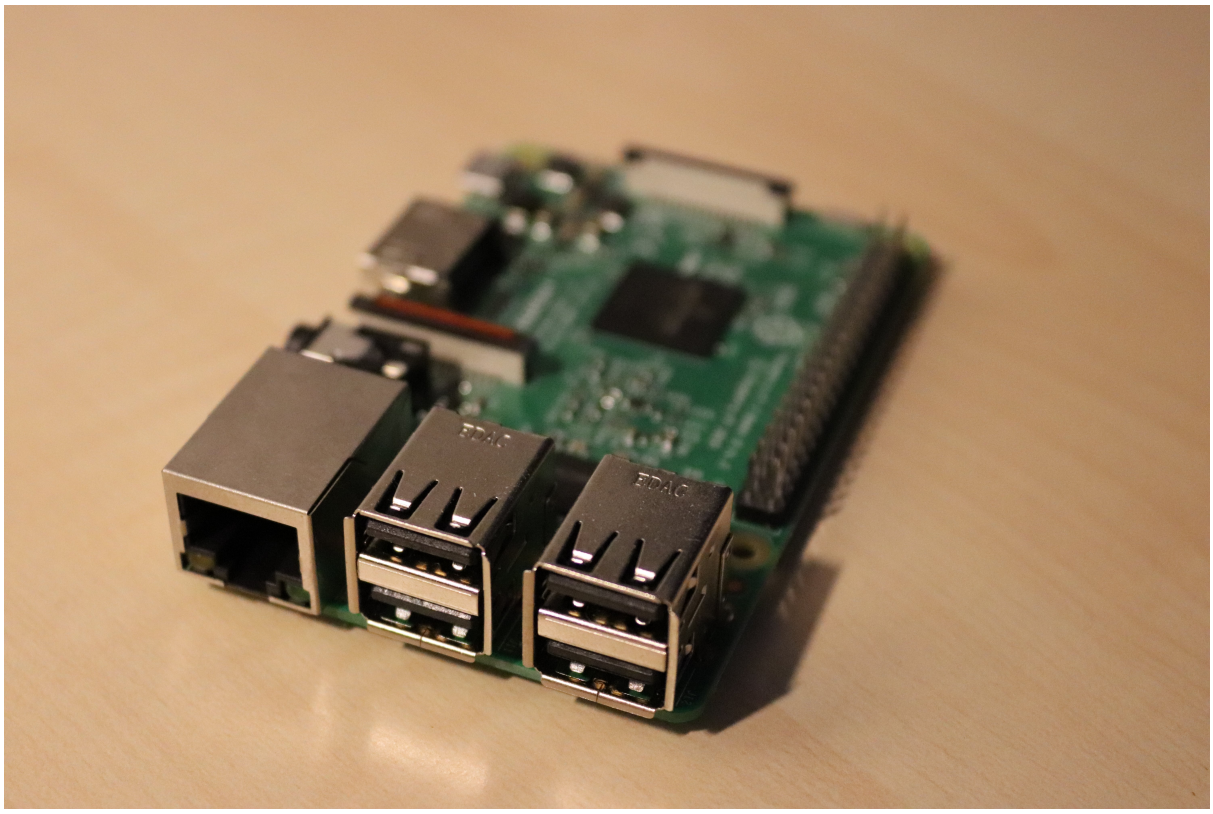

Figure 3.5. – Raspberry Pi 3

3.3. L'application IoT sur le boîtier

Afin de faire en sorte que l'application fonctionne de manière permanente sur le boîtier, on utilise Supervisor [6]. C'est un système pour client et serveur qui permet de contrôler des processus sur des OS "UNIX-like". Les prochaines sous-sections décrivent les différentes fonctionnalités du boîtier.

3.3.1. Connexion à un serveur distant en SSL

Premièrement, l'application doit se connecter au serveur. Cette connexion se fait par le biais du protocole Websocket. Celui-ci est de plus en plus utilisé et supporté par tous les navigateurs récents. Il a été normalisé par l'IETF [14] et son API définie par le World Wide Web Consortium (W3C) [15].

Ce protocole permet une communication bidirectionnelle, ou autrement dit "full-duplex", permettant ainsi l'échange de données en temps réel entre le client et le serveur mais aussi entre le serveur et le client. Ce qui crée donc des connexions de type Peer to Peer (P2P). Le protocole est construit par-dessus une connexion Transmission Control Protocol (TCP) et débute par un "handshake", permettant de faire évoluer la connexion à Websocket.

[6] `http://supervisord.org/`

Pour permettre cette communication, un module du serveur est aussi implémenté et décrit au point 3.4.1. Il a été par ailleurs sécurisé par le protocole SSL, offrant ainsi une communication sécurisée.

Pour l'application, la librairie javax.websocket est utilisée avec l'implémentation provenant de Tyrus [7], un projet open source en Java pour Websocket.

La connexion se fait alors comme indiquée dans le Listing 3.1.

```
1  BoxEndpoint boxEndpoint = new BoxEndpoint();
2  WebSocketContainer container = ContainerProvider.getWebSocketContainer();
3  Session session = container.connectToServer(boxEndpoint, new URI("wss://camera-stream.
       tk:8001"));
4  String macAddress = tools.GetNetworkAddress.GetAddress("mac");
5  JSONObject obj = new JSONObject();
6  obj.put("mac", macAddress);
7  boxEndpoint.sendMessage(obj.toString());
8  while(session.isOpen()){
9    if(boxEndpoint.hasRequest() && boxEndpoint.getManaging()==false){
10     boxEndpoint.manageRequest();
11   }
12   Thread.sleep(1000);
13 }
```

Listing 3.1 – Code source de connexion au serveur Websocket

La classe BoxEndpoint implémente la classe javax.websocket.Endpoint dont elle hérite. C'est dans cette classe qu'on gère les différents évènements et requêtes (voir 3.3.3). Ici, on se contente de créer une session de connexion avec le serveur. Pour que celui-ci accepte la connexion du boîtier, il est nécessaire d'envoyer son adresse MAC afin que le serveur contrôle qu'il s'agisse bien d'un des boîtiers mis en place chez les clients.

Si le boîtier est accepté, la session est gardée ouverte par le serveur et le boîtier se met alors en attente de requête. Chaque seconde, celui-ci contrôle s'il existe des requêtes en attente d'être traitées, et les exécute l'une après l'autre si c'est le cas.

Cette connexion, ainsi que le démarrage des serveurs REST et Websocket, est détaillée dans le diagramme de séquence de la figure 3.6. Les diagrammes de séquence de ce rapport ont été réalisé avec le site `https://sequencediagram.org/`.[13]

[7] `https://tyrus-project.github.io/`

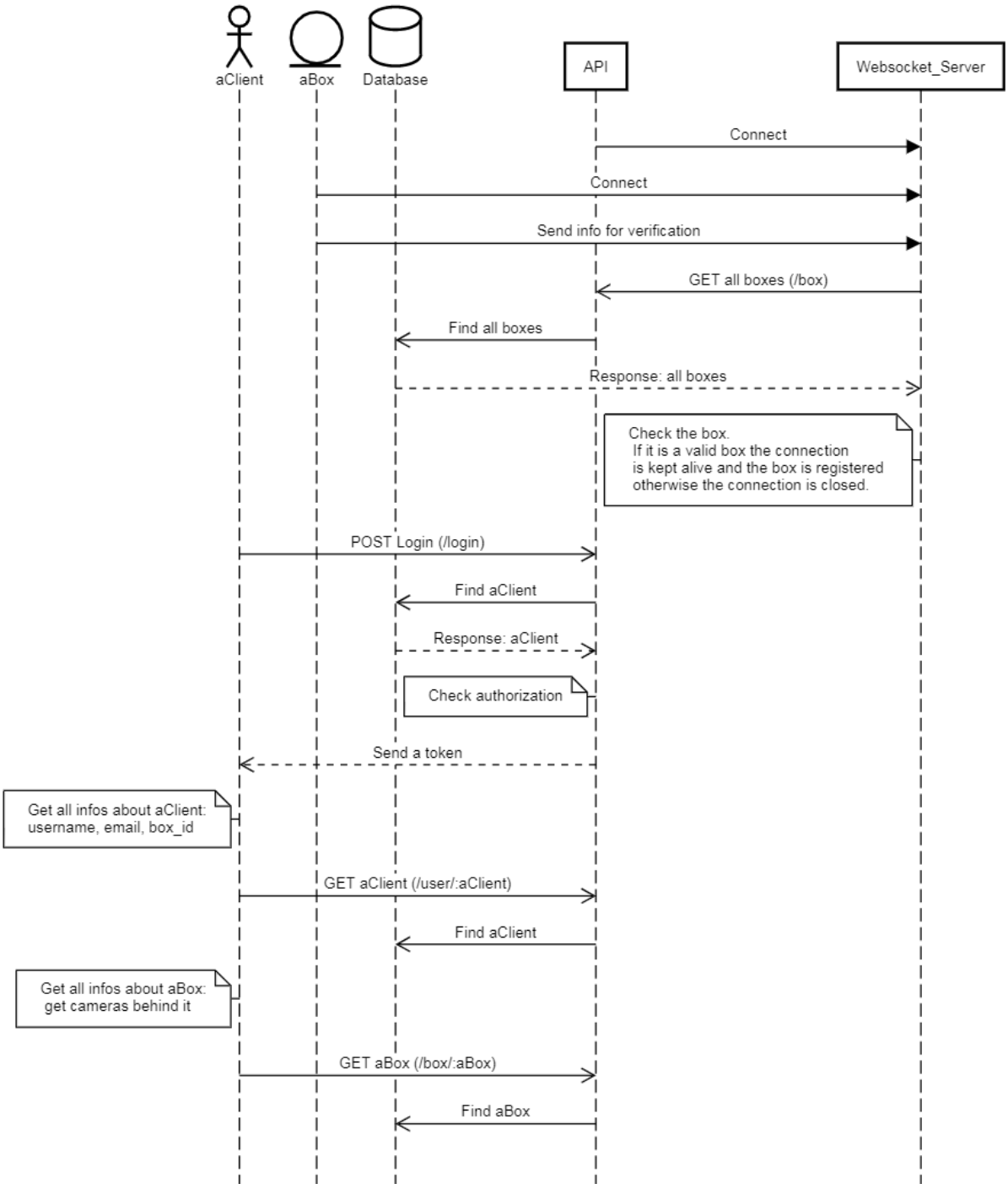

Figure 3.6. – Diagramme de séquence : connexion d'un boîtier IoT et d'un client.

3.3.2. Liste statique de caméras

Dans ce prototype, les caméras sont enregistrées manuellement dans un fichier. Sur chaque ligne du fichier, on écrit alors l'adresse IP d'une caméra, connectée au même réseau local que le boîtier.

L'idéal serait cependant d'avoir un système qui permette de découvrir les caméras existantes sur le réseau. Un tel système serait envisageable, par exemple, en envoyant des

"ping" à des adresses du réseau local, puis, pour contrôler qu'il s'agisse bien de caméras, on pourrait tenter de vérifier si des ports utilisés par défaut par le protocole RTSP, comme le port 8554, sont ouverts. Finalement, ce système serait encore plus performant s'il permettait de découvrir quelle est la marque et le modèle de la caméra, afin de faire correspondre à la caméra, l'URL qui permet d'accéder à son flux vidéo.

3.3.3. Gestion des requêtes

La gestion des requêtes provenant du serveur et des évènements liés au protocole Websocket est implémentée dans la classe BoxEndpoint. Celle-ci possède un constructeur qui lit le fichier contenant la liste des caméras, et enregistre cette liste dans une variable de type ArrayList.

Ensuite, une fonction onOpen (provenant de l'API pour le protocole Websocket) se charge, dès l'ouverture de la session de communication avec le serveur, d'implémenter une fonction onMessage qui enregistre tout message provenant du serveur dans une variable de type ArrayList qui est utilisée comme file d'attente. L'application, toutes les secondes, contrôle si cette file d'attente est vide ou non. Si elle possède des messages, le premier dans la file est traité par la fonction manageRequest(). Celle-ci fait passer une variable booléenne *managing* à vraie, ce qui empêche le traitement d'un autre message tant que celui en cours n'a pas été traité jusqu'au bout, puis elle identifie le type de requête relatif au message.

Il y a trois requêtes possibles :

- Requête d'envoi des caméras connectées sur le réseau : pour y répondre, on crée un tableau en format JSON, que l'on remplit avec chacune des caméras enregistrées précédemment. Ce tableau est ensuite renvoyé au serveur. Une fois le message traité, on l'enlève de la file d'attente et on fixe la variable *managing* à faux.
- Requête de connexion avec une caméra : pour y répondre, on extrait du message la caméra concernée et on crée une commande qui sera exécutée afin d'ouvrir un tunnel entre la caméra et le serveur. Cette commande est décrite au point 3.3.4.
- Requête de fin de connexion avec une caméra : pour y répondre, on extrait du message la caméra concernée et on crée une commande qui fermera le tunnel.

```
1  public void manageRequest() throws FileNotFoundException, IOException{
2      managing = true;
3      String message = store.get(0);
4      JSONObject msg = new JSONObject(message);
5      //Get cameras
6      if(msg.getString("msg").compareTo("camera") == 0){
7          System.out.println("Send cameras");
8              JSONArray arr = new JSONArray();
9              JSONObject camerasJSON = new JSONObject();
10             int i = 0;
11             for(String cameraIP : cameras){
12                 JSONObject camera = new JSONObject();
13                 camera.put("ip", cameraIP);
14                 camera.put("name", "camera"+i);
15                 arr.put(camera);
16                 i++;
17             }
18             camerasJSON.put("cameras", arr);
19             if(i!=0){
```

```
20                 sendMessage(camerasJSON.toString());
21             }
22         store.remove(0);
23         managing = false;
24     //Create connexion to camera
25     } else if(msg.getString("msg").compareTo("connexion") == 0){
26         String port = msg.getString("port");
27         String camera = msg.getString("camera");
28         camera = camera.replaceAll("[^0-9]", "");
29         int cameraId = Integer.parseInt(camera);
30         String ip = cameras.get(cameraId);
31         String connexionID = "conCam"+camera;
32         String command = "ssh -M -S "+connexionID+" -f -N -T -l loic -R"+port+":"+ip+"
               :80 camera-stream.tk";
33         Runtime rt = Runtime.getRuntime();
34         Process pr = rt.exec(command);
35         System.out.println("Connexion with camera is opened !");
36         //send answer to the server [...]
37         connexionsCam.put(camera, pr);
38         store.remove(0);
39         managing = false;
40     //Kill the connexion to a specific camera
41     } else if(msg.getString("msg").compareTo("kill") == 0){
42         System.out.println("Kill the connexion");
43         String camera = msg.getString("camera");
44         camera = camera.replaceAll("[^0-9]", "");
45         String connexionID = "conCam"+camera;
46         String command = "ssh -S "+connexionID+" -O exit loic@camera-stream.tk";
47         Runtime rt = Runtime.getRuntime();
48         Process pr = rt.exec(command);
49         //send answer to the server [...]
50         connexionsCam.remove(camera);
51         store.remove(0);
52         managing = false;
53     //Everything else
54     } else {
55         System.out.println(msg);
56         store.remove(0);
57         managing = false;
58     }
59 }
```

Listing 3.2 – Code source de la fonction manageRequest()

La communication de ces requêtes au sein du système est décrite dans le diagramme de séquence de la figure 3.9.

3.3.4. Ouverture d'un tunnel SSH inversé

Pour la connexion à une caméra, le problème réside sur le fait qu'elle se trouve derrière un routeur, dans un réseau privé. Le serveur ne peut donc pas y accéder directement. En effet, le NAT empêche de connaître les adresses IP se trouvant derrière le routeur. Ce problème est récurent, comment traverser le NAT ? Une solution serait de faire une redirection de port, mais cela nécessiterait de faire des configurations pour toutes les caméras dans le réseau. Une alternative est alors envisagée : l'utilisation du boîtier IoT au sein du

réseau permet l'accès aux caméras. Le boîtier se connecte au serveur automatiquement comme on l'a vu précédemment et fait l'intermédiaire entre les caméras et le serveur. L'Internet Engineering Task Force (IETF) propose ensuite certaines solutions standardisées pour passer à travers le NAT, mais comme discuté dans l'article de Gianni D'Angelo et Salvatore Rampone [18], certaines de ces solutions peuvent avoir un impact sur la performance. Leur article compare certaines de ces solutions et recommande le protocole Websocket pour passer le NAT et transmettre un flux d'une caméra, car celui-ci est le plus performant. Dans le prototype pour ce projet, nous utilisons donc le protocole Websocket pour la communication avec le serveur, mais on choisit aussi d'utiliser les tunnels SSH pour la transmission du flux de la caméra. La combinaison de ces deux solutions permet de les reconnaître toutes deux, cependant il est possible que ce ne soit pas la méthode la plus performante.

Une fois le boîtier ayant créé une communication au travers du protocole Websocket, celui-ci peut donc à la demande du serveur, connecter une caméra directement au serveur par le biais d'un tunnel SSH. Ce tunnel est dit inversé car ce n'est pas le serveur qui établit une connexion vers la destination mais la destination vers le serveur. Le boîtier est l'initiateur du tunnel, mais le début est la caméra et la fin le serveur. Dans la requête que le serveur fait, se trouve un numéro de port sur lequel le tunnel SSH est mis en place. Dès que la connexion est établie sur ce port, une connexion est établie à l'adresse IP de la caméra, sur le port spécifié (ici le port 80). La commande utilisée est celle de la ligne 32 du Listing 3.2.

Une fois ce tunnel établi, le serveur peut communiquer avec la caméra par le biais du port qu'il a transmis dans sa requête. Ainsi, pour accéder au flux de la caméra, il ne reste plus qu'à faire une requête à ce port. Par exemple, si le port 10112 a été spécifié, on accède à la caméra par le biais de l'url suivant : `https://camera-stream.tk:10112`. Comme le port 80 de la caméra est le début du tunnel, cet URL mènera à l'interface HTTP de la caméra.

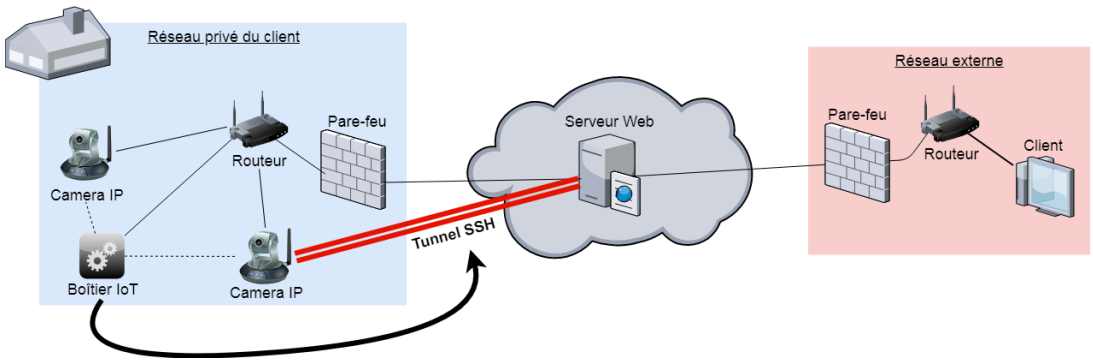

Figure 3.7. – Création d'un tunnel SSH

Les tunnels SSH sont gérés par le boîtier IoT. Lorsqu'une connexion à une caméra doit être terminée, c'est celui-ci qui s'en charge en arrêtant le processus à l'origine du tunnel. Pour cette gestion des connexions SSH, l'option ControlMaster est utilisée.

Un récapitulatif de la communication des différents éléments du système est trouvable sous la forme d'un diagramme de séquence dans la figure 3.9.

Sécurité

Un problème se pose néanmoins avec cette solution. L'accès au flux de la caméra est possible pour tout le monde. En effet, il suffit de connaître le bon numéro de port sur le serveur pour y accéder. Un hackeur peut sans autre, avec une attaque par force brute, trouver les ports qui sont destinés aux caméras. Il faut alors les sécuriser. Ceci, ainsi que le fonctionnement du serveur vous est présenté dans la section 3.4.

3.4. Le serveur - Back end

Le serveur web est composé de plusieurs modules distincts qui pourraient être placés sur des serveurs différents sans que cela n'affecte le fonctionnement du système. Ces modules sont :

— Le serveur Websocket : celui-ci s'occupe de gérer les requêtes faites par le client au travers du front-end et de les envoyer au boîtier IoT. De même, il renvoie les informations transmises par le boîtier au client.

— L'API Rest : le front-end et le serveur websocket communiquent avec l'API afin d'avoir accès à la base de données.

— L'interface utilisateur : sous forme de site web, elle utilise le framework Vue.JS.

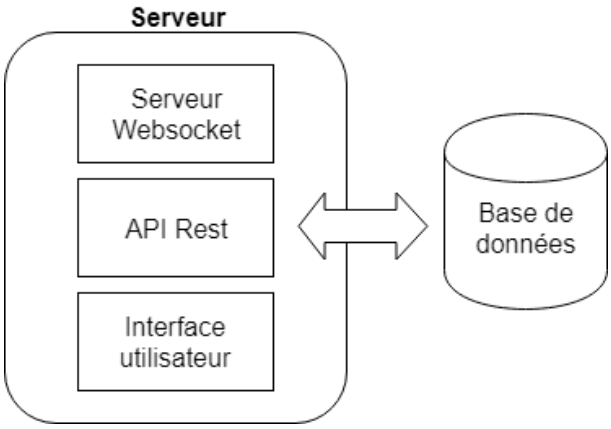

Figure 3.8. – Le serveur Web

3.4.1. Serveur Websocket

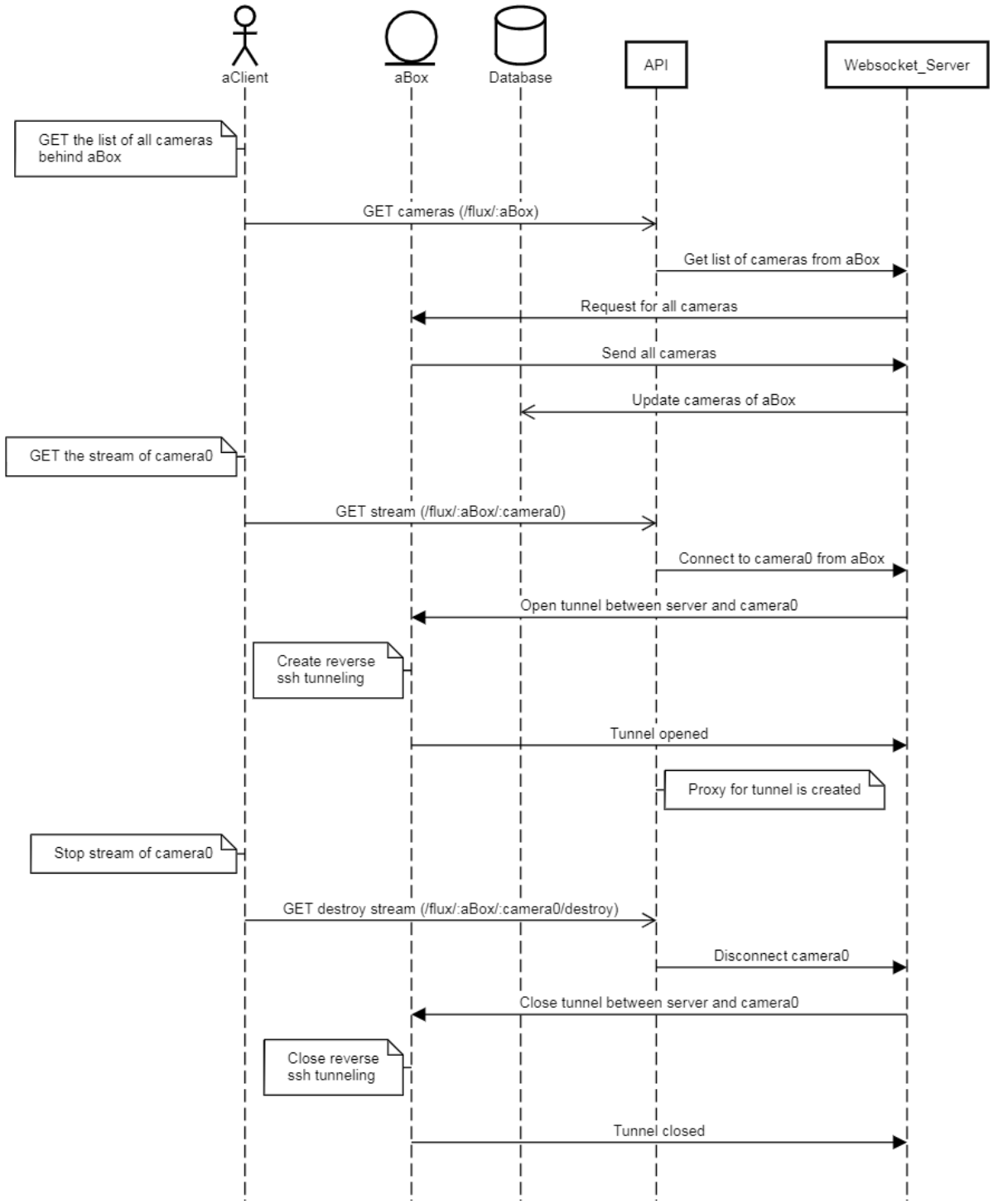

Figure 3.9. – Diagramme de séquence : les différentes requêtes.

Le fonctionnement du serveur Websocket est relativement simple. Celui-ci est tout d'abord créé avec des certificats SSL provenant de Let's Encrypt (pour plus d'informations revoir le point 2.2.2). La librairie ws [8] est utilisée comme implémentation du protocole Websocket.

[8] `https://www.npmjs.com/package/ws`

La gestion de l'évènement de connexion est ensuite implémentée, tout comme la gestion de l'évènement d'un message entrant.

Pour pouvoir être reconnu comme un boîtier IoT authentique de l'entreprise et non pas comme un potentiel hackeur, une vérification est tout d'abord faite lorsque le boîtier se connecte au serveur Websocket, comme expliquée à la section 3.3.1 et illustrée dans la figure 3.6. En effet, le boîtier IoT envoie son adresse MAC dans le premier message qu'il adresse au serveur Websocket. Cette adresse ainsi que l'adresse IP globale de la résidence du client sont déjà enregistrées au préalable dans la base de données par l'entreprise. Ainsi, le serveur Websocket récupère l'adresse MAC envoyée, ainsi que l'adresse IP provenant du message et compare ces deux adresses avec celles enregistrées dans la base de données. On peut aussi penser à ajouter un élément supplémentaire comme une sorte de mot de passe qui serait comparé, mais cela n'est pas implémenté dans le prototype.

Si la vérification se passe bien, le serveur Websocket associe l'identifiant du boîtier IoT contenu dans la base de données à la connexion, afin de pouvoir communiquer avec par la suite. Sinon, la connexion est interrompue.

De plus, le serveur Websocket gère cinq autres types de messages :

- La requête pour la liste des caméras associées à un boîtier IoT, qui est simplement retransmise au boîtier concerné en utilisant l'identifiant associé.
- La réponse du boîtier à la requête concernant la liste des caméras : les caméras associées au boîtier dans la base de données sont alors supprimées et remplacées par les nouvelles reçues dans le message.
- La demande de l'ouverture d'un tunnel entre le serveur et une caméra spécifique : à nouveau la requête est simplement retransmise en utilisant l'identifiant du boîtier concerné.
- La réponse du boîtier après l'ouverture ou la fermeture d'un tunnel : le message contient une variable booléenne indiquant si le tunnel est ouvert ou fermé.
- La demande de fermeture d'un tunnel entre le serveur et une caméra spécifique : comme pour les autres demandes, celle-ci est simplement retransmise au boîtier concerné.

Tout autre message non reconnu est affiché dans la console et n'est pas pris en compte.

3.4.2. API RESTful et base de données

Comme précisé précédemment cette Application Programming Interface (API) est implémentée avec le framework Koa. Un *boilerplate* est utilisé comme point de départ de l'implémentation : Koalerplate [9]. Celui-ci contient plusieurs librairies permettant le bon fonctionnement de l'API dont notamment koa-router [10] et cors [11] offrant des ressources pour construire une API RESTful.

Deux autres librairies sont aussi utilisées : mongoose et koa-jwt [12].

[9] `https://github.com/dbalas/koalerplate`
[10] `https://www.npmjs.com/package/koa-router`
[11] `https://github.com/koajs/cors`
[12] `https://www.npmjs.com/package/koa-jwt`

Mongoose et MongoDB

Mongoose permet d'interagir de manière plus aisée avec une base de données MongoDB. En effet, il permet de créer des schémas recensant tous les champs à attribuer aux entités qui seront enregistrées dans la base de données. Un tel schéma est ensuite repris par un modèle : un constructeur compilé pour respecter la définition du schéma.

Le schéma pour le boîtier IoT est donné comme exemple dans le Listing 3.3. On trouve d'ailleurs dans ses champs, des références aux schémas de l'utilisateur et de la caméra, dans le but de relier le boîtier IoT à un utilisateur précis et d'y associer des caméras. L'instance d'un modèle est appelé document. Ce sont ces documents qui sont enregistrés dans la base de données MongoDB.

```
const BoxSchema = new mongoose.Schema(
   {
       user: { type: mongoose.Schema.ObjectId, ref: 'User'},
       ip: { type: String},
       mac: { type: String},
       cameras: [{ type: mongoose.Schema.ObjectId, ref: 'Camera'}]
   },
   { timestamps: true }
);
```

Listing 3.3 – Schéma du boîtier IoT

Pourquoi utiliser MongoDB ?[20] Ce système de gestion de base de données est un des plus utilisés et s'insère dans l'air du Not only SQL (NoSQL), se différenciant donc des systèmes classiques comme MySQL avec le Structured Query Language (SQL). L'utilisation d'un tel type de système permet entre autre de pouvoir distribuer la base de données sur un nombre quelconque d'ordinateurs, chose qui devient très courante avec l'architecture en *cluster*. Les détails de ce qu'offre le NoSQL sortent du cadre de ce travail néanmoins de plus amples explications peuvent être trouvées dans le livre d'Andreas Meier et de Michael Kaufmann [21]. L'avantage d'utiliser MongoDB pour le prototype de ce projet est que beaucoup d'exemples en lien avec Koa utilisent ce système de gestion de base de données. De plus, c'est un système simple sans réel schéma prédéfini pour les données, qui sont structurées au format BSON, une extension du format JavaScript Object Notation (JSON).

La base de données est créée avec mLab [13], un Database-as-a-Service (DaaS).

Pour ce prototype, on crée trois schémas différents pour l'utilisateur, la caméra et le boîtier IoT.

JSON Web Token

Un JSON Web Token (JWT) va permettre de sécuriser les échanges entre l'API et ses utilisateurs. Celui-ci consiste simplement à s'assurer qu'une demande à une ressource de l'API que l'on voudrait garder privée, provienne d'un utilisateur authentifié. Pour ce faire, l'utilisateur peut obtenir un *token* (jeton), qui certifiera son authenticité.

[13] `https://mlab.com/`

Les routes et les contrôleurs

L'API est atteignable à l'URL suivante : `https://camera-stream.tk:3000/v1/app`. Ensuite, plusieurs routes ou Unified Resource Identifier (URI) différentes mènent à certaines ressources. Celles-ci sont indiquées dans le tableau 3.2.

Requête	URI	Description
POST	/login	Contrôle l'utilisateur et renvoie un JWT
GET	/user	Renvoie une liste de tous les utilisateurs
GET	/user/user_id	Renvoie un utilisateur
POST	/user/create	Crée un utilisateur selon le schéma
DELETE	/user/user_id	Supprime un utilisateur et ce qui s'y rapporte
GET	/box	Renvoie une liste de tous les boîtiers IoT
GET	/box/box_id	Renvoie un boîtier IoT
POST	/box/box_id	Mise à jour des caméras d'un boîtier IoT
POST	/box/create/user/user_id	Crée un boîtier IoT pour un utilisateur
DELETE	/box/box_id	Supprime un boîtier IoT
GET	/camera/camera_id	Renvoie une caméra
POST	/camera/create/box/box_ic	Crée une caméra pour un boîtier IoT
DELETE	/camera/camera_id	Supprime une caméra
GET	/flux/box_id/	Demande l'actualisation des caméras d'un boîtier
GET	/flux/box_id/camera_id	Ouvre une connexion à une caméra
DELETE	/flux/box_id/camera_id	Ferme une connexion à une caméra

Table 3.2. – Routes menant aux ressources de l'API

Chacune de ces routes est associée à une fonction précise d'un contrôleur. Il y a quatre contrôleurs : pour les boîtiers, les caméras, les utilisateurs et les connexions aux caméras.

Le Listing 3.4 présente la fonction qui permet de créer une connexion à une caméra. C'est ici que nous répondons à la question posée à la section 3.3.4. En effet, une fois le tunnel ouvert sur un port du serveur, celui-ci est accessible par n'importe qui si l'on connait le bon numéro. L'idée pour régler ce problème de sécurité est de fermer les ports au public et de créer un proxy par l'API pour y accéder. Celui-ci se charge alors d'acheminer le flux vers le client, tout en utilisant l'URL spécifique d'accès au flux de la caméra (voir le tableau 3.1).

```
1  async function createProxy (ctx) {
2     const box = ctx.params.box_id;
3     const camera = ctx.params.camera_id;
4     const port = await getPort();
5     const portToProxy = port.toString();
6     console.log('Available port : %s', portToProxy);
7     var request = {msg: 'connexion', boxId: box, cameraId: camera, port: portToProxy}
8     //send message to websocket server, to open connection at the box
9     ws.send(JSON.stringify(request));
10    //wait websocket answer
11    await sleep(2000);
12    //create proxy
13    const p = proxy('51.15.227.253', {
14     port: portToProxy,
15     proxyReqPathResolver: (ctx) => { return '/axis-cgi/mjpg/video.cgi';}
```

```
16      });
17      await p(ctx);
18  }
```

Listing 3.4 – Fonction pour la connexion à une caméra

Dans le prototype, cependant, des problèmes ont lieu avec l'utilisation de ce proxy. La librairie koa-better-http-proxy [14] est utilisée. Ces problèmes sont sûrement dus à une mauvaise compatibilité entre le proxy et le protocole HTTPS mis en place et n'ont malheureusement pas pu être réglés. En guise de solution provisoire mais non sécurisée, l'URL d'accès au flux de la caméra par le port ouvert au public est transmis au client.

On remarque dans le Listing 3.4 l'utilisation d'une programmation asynchrone. Celle-ci a un fonctionnement non bloquant qui permet au serveur de continuer d'avancer dans le programme pendant que la fonction s'exécute. Cette programmation est aussi utilisée pour les autres contrôleurs. Ici, l'utilisation d'*await* permet de faire appel à des fonctions asynchrones dans une syntaxe synchrone. La fonction sleep est construite comme une *Promise* (une promesse), c'est-à-dire qu'on attend un résultat de cette fonction. Le résultat attendu ici est que la promesse se résolve après 2 secondes.

La fonction sleep est utilisée afin d'attendre que le tunnel s'ouvre bien avant de pouvoir créer le proxy, cependant une programmation plus optimale serait de demander au serveur Websocket de retourner une réponse quand le tunnel est créé et d'en faire une promesse. Plusieurs librairies pour réaliser cette idée existent mais n'ont pas été testées.

3.5. Interface utilisateur - Front end

Pour le *front-end*, un site web a été créé avec le framework Vue.js. Celui-ci est disponible à l'adresse `https://camera-stream.tk/`. Comme on peut le constater, il possède des certificats SSL.

Pour l'implémentation de ce site, on démarre avec WebPack [15] qui permet d'organiser notre application Vue.JS en modules. Ensuite, le tutoriel de Pawel[17] permet de construire petit à petit notre site web.

Celui-ci possède uniquement quatre routes : une menant à la page d'accueil du site (/) une autre menant à la page de login (/login), puis à la page principale de l'application (/app) et enfin une route pour se déconnecter d'une session (/logout).

Vue.js a été choisi pour sa simplicité et sa petite taille, comme l'application à implémenter n'est pas spécialement grande.

Pour créer le design du site, on utilise Vuetify [16] qui met à disposition divers outils comme des menus, des boutons ainsi que certains styles à utiliser. Les figures 3.10, 3.11 et 3.12 montrent les différentes pages de l'application.

Sur le serveur, le site est déployé grâce au logiciel open source NGINX [17].

[14] `https://www.npmjs.com/package/koa-better-http-proxy`
[15] https ://webpack.js.org/
[16] `https://vuetifyjs.com/en/`
[17] https ://www.nginx.com/

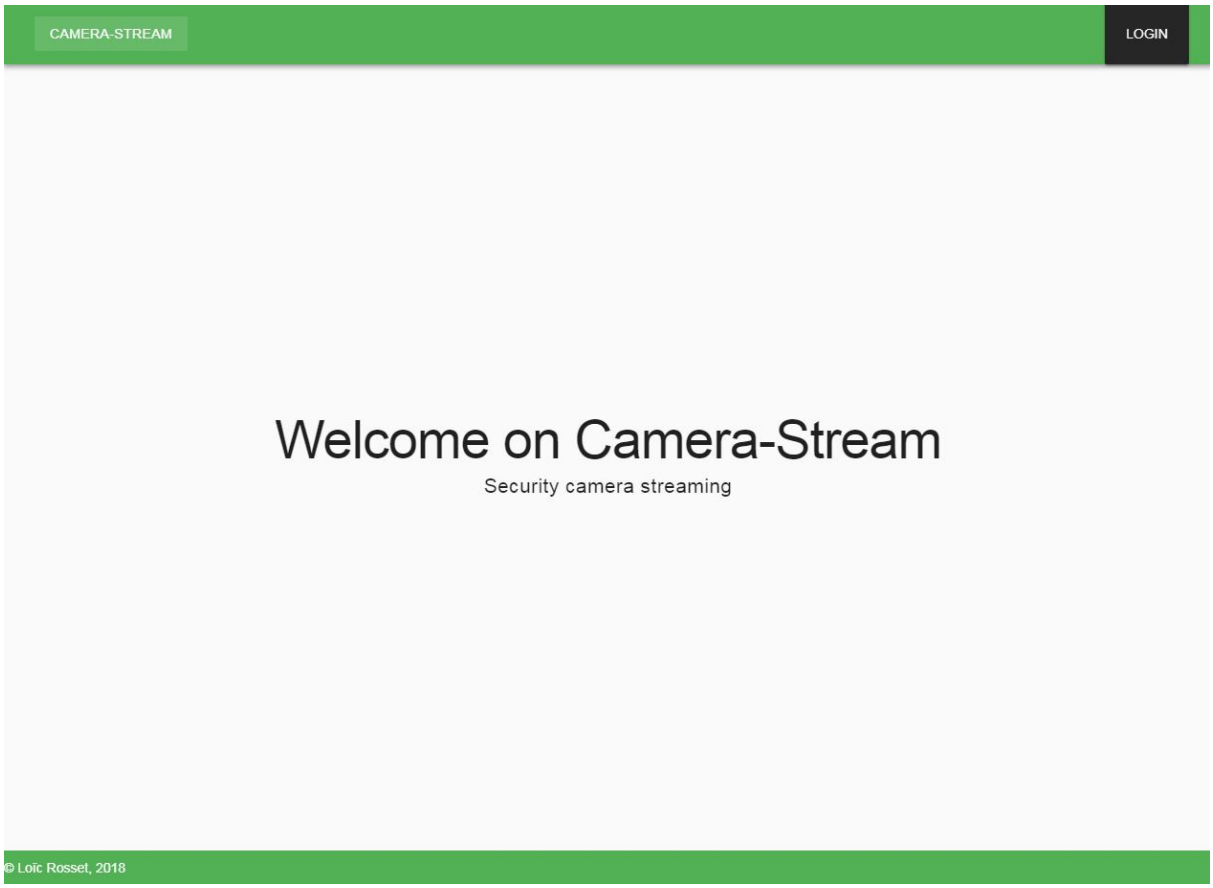

Figure 3.10. – Page d'accueil du site web.

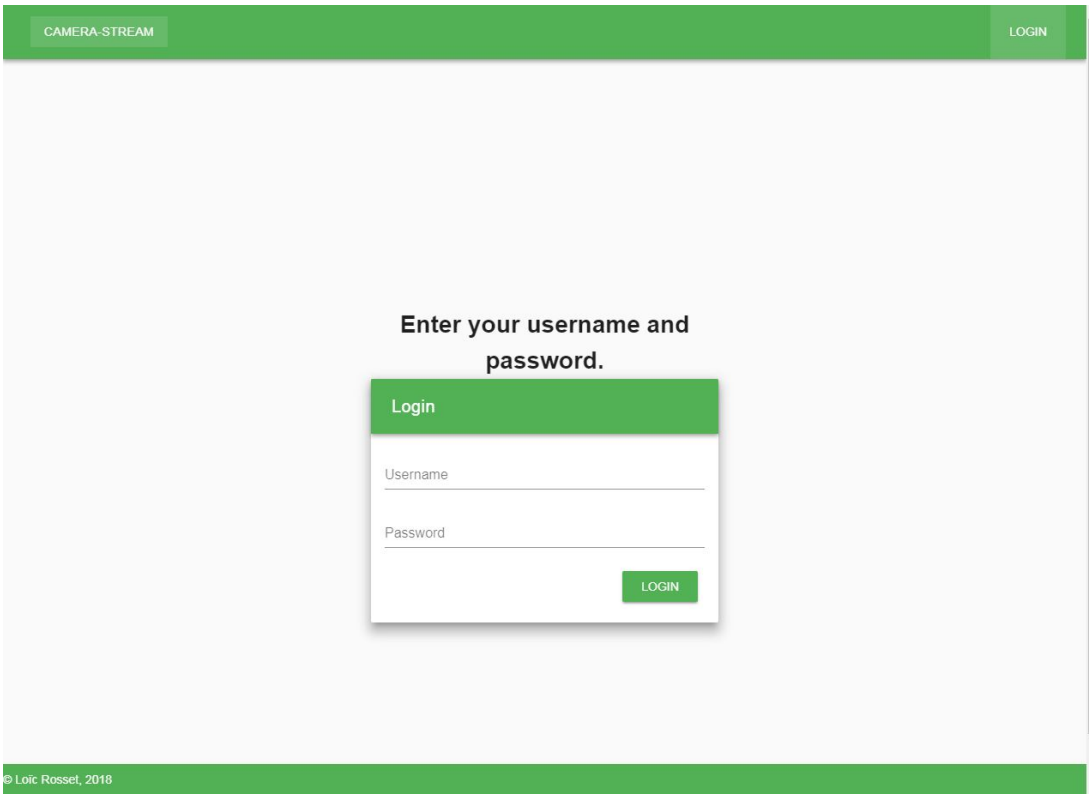

Figure 3.11. – Page de login du site web.

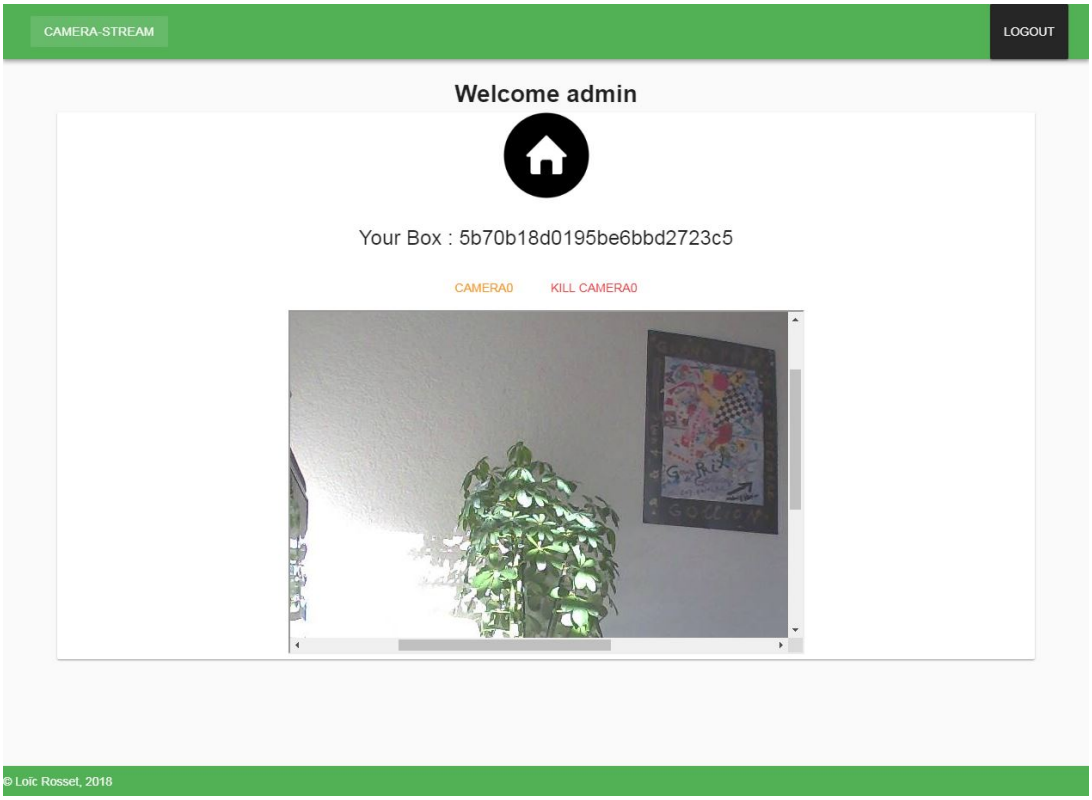

Figure 3.12. – Page principale de l'application avec une connexion ouverte sur une caméra.

3.5.1. Modules Vue.js

Le site web est divisé en plusieurs modules au format Vue.js (.vue). Chacun de ces modules est construit en trois champs : *template*, *script* et *style*.

Dans *template* on construit les éléments qui seront placés dans le module et c'est dans *script* que le comportement du module sera implémenté avec des méthodes et fonctions. Enfin dans *style*, on indique la manière dont les éléments doivent apparaitre, c'est donc dans ce champ que l'on peut changer la police ou la taille de certains éléments en usant du langage Cascading Style Sheet (CSS).

Le module pour la barre de menu est donné comme exemple dans le Listing 3.5.

```
<template>
  <v-toolbar color="green" app dark>
    <v-toolbar-title>
      <v-btn v-if="currentUser" flat :to="{name: 'MyApp'}">{{title}}</v-btn>
      <v-btn v-else flat :to="{name: 'HelloWorld'}">{{title}}</v-btn>
    </v-toolbar-title>
    <v-spacer></v-spacer>
    <v-toolbar-items>
      <v-btn v-if="currentUser" v-for="menu in menusUser" :key='menu.index' :to={name:
          menu.route}>
          {{menu.name}}
      </v-btn>
      <v-btn v-else v-for="menu in menus" :key='menu.index' :to={name:menu.route}>
          {{menu.name}}
      </v-btn>
    </v-toolbar-items>
  </v-toolbar>
</template>

<script>
import { mapGetters } from 'vuex'
export default {
  name: 'Navbar',
  data () {
    return {
      title: 'Camera-Stream',
      menusUser: [
        {name: 'Logout', route: 'Logout'}
      ],
      menus: [
        {name: 'Login', route: 'Login'}
      ]
    }
  },
  computed: {
    ...mapGetters({ currentUser: 'currentUser' })
  }
}
</script>

<style>
</style>
```

Listing 3.5 – Module Vue.js de la barre de menus

3.5.2. Extension possible

Une page d'administration pour l'entreprise pourrait être créée afin que celle-ci puisse instaurer de nouveaux utilisateurs et leurs associer des boîtiers IoT et les gérer.

4
Conclusion

4.1. Synthèse

Si on reprend la liste de Gary, à la section 2.2, le prototype respecte la plupart des points mais pas tous. En effet, la caméra IP utilisée n'a pas été configurée avec un identifiant et un mot de passe complexe. Ce point pourrait être réglé et les mots de passe et identifiants stockés de manière cryptée dans la base de données. D'ailleurs, les mots de passe et identifiants des utilisateurs sont stockés de manière non-cryptés dans celle-ci. Cela pourrait aussi être corrigé, avec une fonction de hachage par exemple. Un autre point qui devrait être réglé est celui des mises à jour. Celles-ci n'ont pas été prévues sur l'application du boîtier IoT décrite dans ce travail.

On remarque donc que pour sécuriser un projet intégrant ou non des objets connectés, il faut prendre du temps. L'idéal serait de tester son système et faire des simulations d'attaques potentielles. Mais tout cela coûte cher aux entreprises et le sujet de la sécurité est sans doute parfois étudié trop rapidement, engendrant des systèmes vulnérables. Ceux-ci peuvent être alors meilleur marché, mais le prix gagné en argent est payé en sécurité. Et souvent ce deuxième prix coûte bien plus cher que prévu. On se souvient du cas de Jeep évoqué dans le chapitre 2, la marque avait été avertie par Charlie Miller et Chris Valasek. Cependant les failles de sécurité n'ont pas réellement été fixées. Ainsi lorsque les détails de piratage ont été rendus publiques, la marque a été forcée de rappeler plus d'un million de véhicules pour corriger le logiciel ce qui a coûté des billions de dollars à la compagnie, selon Gary. [6]. Le côté positif est qu'il existe des solutions, des manières de crypter les communications et les données par des protocoles comme SSH et SSL, et ainsi protéger l'utilisation d'objets connectés. Certaines de ces solutions peuvent avoir des impacts sur la performance, comme étudié dans l'article de Gianni D'Angelo et Salvatore Rampone [18], mais de manière générale ceux-ci sont tout à fait acceptables.

Le prototype réalisé répond que partiellement aux attentes de l'entreprise mais les solutions aux problèmes, comme celui de la reconnaissance de la marque de la caméra pour connaître le bon URL d'accès au flux de celle-ci, décrites dans ce rapport, permettent de le rendre totalement opérationnel si implémentées.

4.1.1. Codes sources

Les codes sources de l'application sur le boîtier IoT et du serveur sont donnés avec ce rapport mais sont aussi disponibles sur des dépôts Git aux adresses suivantes : `https://github.com/LoRosset/boxSocket` et `https://github.com/LoRosset/camerastream`.

4.2. Évolution future

De nouvelles technologies se développent pour répondre aux problématiques liées à la sécurité, comme la Blockchain. Celle-ci révolutionne même le rapport à la confiance dans les réseaux. Des services comme le VSaaS pourraient alors simplement ne plus exister car ils ne seraient plus nécessaires.

Développée en premier lieu pour sécuriser les échanges de monnaie virtuelle, comme le BitCoin, cette technologie pourrait être appliquée aux objets connectés et son organisation décentralisée fournir de nouveaux usages de l'Internet des objets.

4.3. Défis personnels

Ce travail a été un défi sous plusieurs angles.

Tout d'abord, il a fallu comprendre les besoins de l'entreprise qui a proposé le projet et essayer de le mettre en œuvre, complètement, et ce de manière indépendante. Construire tout un système, seul, en partant de rien nécessite énormément d'implication, de concentration et de motivation.

Il m'a fallu recueillir beaucoup de connaissances dans plusieurs domaines différents qui m'étaient pour la plupart inconnus : les caméras, la sécurité, le réseau, certains langages et frameworks. Une fois la connaissance acquise, il faut encore la mettre en pratique et cela aussi est une connaissance en soit : c'est en faisant que l'on apprend. Mettre en place toutes les pièces de ce puzzle m'a pris beaucoup de temps et nécessita un grand nombre de réglages, qui n'ont d'ailleurs pas tous pu être terminés.

Cependant l'apport que m'a procuré ce travail de Bachelor est incontestable, tant au niveau personnel, dans ma vision de l'informatique, que professionnel, dans l'extension de mes connaissances.

A

Acronymes communs

API	Application Programming Interface
CA	Certificate Authority
CSS	Cascading Style Sheet
DaaS	Database-as-a-Service
DDoS	Distributed Denial of Service
DNS	Domain Name System
DVR	Digital Video Recorder
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IoT	Internet of Things
IETF	Internet Engineering Task Force
IP	Internet Protocol
JSON	JavaScript Object Notation
JWT	JSON Web Token
NAT	Network Address Translation
NoSQL	Not only SQL
NVR	Network Video Recorder
OS	Operating System
PoE	Power over Ethernet
PTZ	Pan-Tilt-Zoom
P2P	Peer to Peer
REST	Representational State Transfer
RFC	Request For Comments
RTSP	Real Time Streaming Protocol
SQL	Structured Query Language
SSH	Secure Shell
SSL	Secure Socket Layer
TCP	Transmission Control Protocol
TLS	Transport Layer Security
URI	Unified Resource Identifier
URL	Uniform Resource Locator
VSaaS	Video Surveillance as a Service
W3C	World Wide Web Consortium
WoT	Web of Things

Références

[1] Dominique D. Guinard et Vlad M. Trifa. *Building the Web of Things*. Manning, Juin 2016. 4

[2] Andy Greenberg. *Hackers remotely kill a jeep on the highway - with me in it.* `https://www.wired.com/2015/07/hackers-remotely-kill-jeep-highway/`. Wired, Juillet 2015. 4

[3] Katja Schaer. *Des hackers démontrent qu'un pacemaker peut être aisément piraté.* `https://www.rts.ch/info/sciences-tech/medecine/9772308-des-hackers-demontrent-qu-un-pacemaker-peut-etre-aisement-pirate.html`. RTS, 13 août 2018. 4

[4] Rob Lever. *Global cybercrime costs $600 bn annually : study.* `https://phys.org/news/2018-02-global-cybercrime-bn-annually.html`. Phys.org, 21 février 2018. 5

[5] Statista. *Number of IoT devices in use worldwide from 2009 to 2020 (in billion units).* `https://www.statista.com/statistics/764026/number-of-iot-devices-in-use-worldwide/`. Avril 2017. 5

[6] Gary Sims. *IoT security - what you need to know (Gary explains).* `https://www.androidauthority.com/iot-security-gary-explains-727977/`. 25 janvier 2018. 5, 29

[7] Ylonen T. and others. *The Secure Shell (SSH) Transport Layer Protocol.* (RFC 4253) Janvier 2006. 5

[8] A. Freier, P. Karlton and others. *The Secure Sockets Layer (SSL) Protocol Version 3.0.* (RFC 6101) Août 2011. 6

[9] Axis. *The Axis story.* `https://www.axis.com/about-axis/history`, dernière consultation : 02.12.2018.

[10] Seagate. *What is Video Surveillance as a Service & how does it work ?.* `https://www.seagate.com/de/de/tech-insights/what-is-video-surveillance-as-a-service-and-how-can-it-work-for-you-master-ti/`, dernière consultation : 09.12.2018. 8

[11] Fielding R. and others. *Hypertext Transfer Protocol - HTTP/1.1.* (RFC 2616) Juin 1999. 12

[12] Schulzrinne H. and others. *Real-Time Streaming Protocol Version 2.0.* (RFC 7826) Décembre 2016. 12

[13] SequenceDiagram.org. https://sequencediagram.org/, dernière consultation : 19.12.2018. 14

[14] Fette I., Google and others. *The WebSocket Protocol*. (RFC 6455) Décembre 2011. 13

[15] Wide Web Consortium. *The WebSocket API*. https://www.w3.org/TR/websockets/. 20 septembre 2012. 13

[16] Jack Rhysider. *Raspberry Pi : Phoning Home Using a Reverse Remote SSH tunnel*. https://www.tunnelsup.com/raspberry-pi-phoning-home-using-a-reverse-remote-ssh-tunnel/, dernière consultation : 02.12.2018.

[17] Pawel J. Wal. *Vue.js front end app*. https://paweljw.github.io/2017/09/vue.js-front-end-app-part-1-setting-up-the-app/, dernière consultation : 02.12.2018. 24

[18] D'Angelo, Gianni and Rampone, Salvatore. *A NAT traversal mechanism for cloud video surveillance applications using WebSocket*. Octobre 2018. 18, 29

[19] David Wettstein. *RESTful services and automation for comfort-oriented smart devices*. Master thesis, Department of Informatics, University of Fribourg, Switzerland, Décembre 2017. 9

[20] Prashanth Jayaram. *When to use (and not to use) MongoDB*. https://dzone.com/articles/why-mongodb, dernière consultation : 06.12.2018. 22

[21] Andreas Meier, Michael Kaufmann. *SQL- & NoSQL- Datenbanken*. 2016. 22

[22] BPIFrance. *Une infographie pour comprendre la blockchain*. https://www.bpifrance.fr/A-la-une/Actualites/Une-infographie-pour-comprendre-la-blockchain-38874. 29 janvier 2018.