Rapport technique
LOG795 - Projet de fin d'études
Département de génie logiciel et des TI

Environnement applicatif IoT dynamique

Auteurs
Guy Martial Ngowa Nzali - NGOG18039008
Maxime Lanthier - LANM04119308
Francis Groleau - GROF15069207
Vincent Labrie - LABV18039208

Professeur superviseur
Alain April

Date
12 août 2019

Remerciements

L’équipe tient à remercier l’École de Technologie Supérieure pour les différentes opportunités d'apprentissage tel que le projet de fin d’études ainsi que pour tous les enseignements offerts.

L’équipe tient à remercier le professeur Alain April pour son encadrement lors du projet.

L’équipe tient à remercier, pour leur expertise, leurs conseils et leur encadrement : Alex Tremblay, Cédric Melançon et Fadi Kaseir.

Environnement applicatif IoT dynamique

Guy Martial Ngowa Nzali - NGOG18039008
Maxime Lanthier - LANM04119308
Francis Groleau - GROF15069207
Vincent Labrie - LABV18039208

RÉSUMÉ

Ce projet a pour but de créer une plateforme permettant de générer des interfaces utilisateurs dynamiques, en utilisant du metadata reçu d'équipements IoT, afin de faire des statistiques, de l'analyse de données ou même différents rapports à l'aide de la télémétrie. Un portail Web permet aussi la configuration de messages envoyés par des équipements IoT, ainsi qu'une gestion d'utilisateurs et de groupes. Cette plateforme est donc composée de trois composants logiciels, soit une application Web, un Web API et une Azure function qui s'occupe de l'ingestion des messages.

TABLE DES MATIÈRES

TABLE DES MATIÈRES 4

LISTE DES TABLEAUX 6

LISTE DES FIGURES 7

LISTE DES ABRÉVIATIONS, SIGLES ET ACRONYMES 8

CHAPITRE 1 - INTRODUCTION 9

CHAPITRE 2 - PROBLÉMATIQUE ET OBJECTIFS 10

2.1 Présentation de l'entreprise 10

2.2 Problématique 10

2.3 Objectifs 11

2.4 Composition de l'équipe 13

2.5 Risques 14

2.6 Livrables 15

CHAPITRE 3 - SOLUTION DÉVELOPPÉE 16

3.1 Outils et librairies logiciels 16

3.1.1 Visual Studio Code 16

3.1.2 Visual Studio 16

3.1.3 SQL Server Management Studio 16

3.1.4 Azure Portal 16

3.1.5 Azure DevOps 16

3.1.6 Azure IoT Hub 17

3.1.7 Azure Blob Storage 17

3.1.8 Redis 17

3.1.9 Entity Framework Core 17

3.1.10 Identity Framework Core 17

3.1.11 Swagger 17

3.1.12 AutoMapper 18

3.1.13 Angular 18

3.1.11 Kendo UI 18

3.2 Architecture et conception 19

3.3 Implémentation 21

3.3.1 Web App 21

3.3.2 Web API 23

3.3.3 Ingestion 25

3.3.3 Protocol Buffer 26

CHAPITRE 4 - SUIVI DE PROJET **27**
4.1 Méthodologie **27**
4.2 Azure DevOps **27**
4.3 Rencontre de suivi de projet **28**
4.4 Rétrospective de sprint **28**
4.5 Communication **29**
CHAPITRE 5 - AMÉLIORATIONS ET TRAVAUX FUTURS **30**
CHAPITRE 6 - ENJEUX ENVIRONNEMENTAUX **32**
CHAPITRE 7 - CONCLUSION **33**

LISTE DES TABLEAUX

Tableau	Description	Page
Tableau 1	Liste des abréviations, sigles et acronymes	8
Tableau 2	Liste des objectifs	12
Tableau 3	Membres du projet	13
Tableau 4	Liste des risques	14
Tableau 5	Tableau des livrables reliés à ce projet	15
Tableau 6	Liste des fonctionnalitées non implémentées	30

LISTE DES FIGURES

Figure	Description	Page
Figure 1	Architecture de l'application	19
Figure 2	Interface de gestion pour les groupes	21
Figure 3	Interface de gestion pour les utilisateurs	22
Figure 4	Interface de gestion pour les utilisateurs	22
Figure 5	Architecture du Web Api	23
Figure 6	Diagramme de séquence de l'ingestion	25
Figure 7	Azure Board	27
Figure 8	Git Flow	28

LISTE DES ABRÉVIATIONS, SIGLES ET ACRONYMES

Acronyme	Définition
API / Web API	Interface de Programmation Applicative (*Application Programming Interface*)
ÉTS	École de Technologie Supérieure
IoT	Internet des objets (*Internet of Things*)
IDE	Environnement de développement intégré (*Integrated development environment*)
SDK	Kit de développement logiciel (Software Development Kit)
PFE	Projet de fin d'études
ESB	Enterprise Service Bus
BPMS	Logiciel de gestion de processus métiers (*Business Process Management Software*)
Azure	Plateforme cloud de Microsoft
Azure DevOps	Plateforme DevOps de Microsoft
DevOps	Philosophie visant à lier étroitement le développement et la gestion des opérations / infrastructures informatiques
CRUD	Acronyme signifiant les opérations de base pouvant être faites sur une entité (*Create-Read-Update-Delete*)
Hot storage	Contient les données critiques, qui doivent être accessibles le plus rapidement possible et qui sont souvent demandées
Cold storage	Contient les données moins critiques, ou qui sont moins souvent demandées
.cs	Type d'extension pour des fichier C#
.dll	Type d'extension pour *Dynamic Link Library.* C'est une bibliothèque logicielle dont les fonctions sont chargées en mémoire
Repository	Sert à avoir un stockage de données organisé
ORM	*Object-Relational Mapping*. Sert d'interface entre une application et une base de données

Tableau 1. Liste des abréviations, sigles et acronymes

CHAPITRE 1 - INTRODUCTION

Depuis quelques années, l'internet des objets est de plus en plus en expansion. L'IoT est en fait l'interconnexion entre l'internet et des objets. Par objets, on entend ici des dispositifs (devices) ayant des fonctionnalités qui requièrent la connexion à l'internet. Par exemple un réfrigérateur intelligent, une cafetière intelligente, un odomètre intelligent, etc. La configuration de ces différents dispositifs est souvent complexe, puisque différents capteurs entrent en jeu selon les différents objets.

Dans ce contexte, l'entreprise Matricis, qui oeuvre dans le domaine du IoT, cherchait à créer une interface simple d'utilisation pouvant permettre la configuration des différents objets connectés.

Ce projet confidentiel « Environnement applicatif IoT dynamique », fait dans le cadre de notre projet de fin d'études, a été réalisé avec la supervision de Matricis. En effet, il est question de concevoir et d'implémenter une application Web qui génère dynamiquement les pages Web en fonction de la structure de données reçue par les devices ou équipements IoT. Cette application Web va également offrir un portail qui permet la configuration des messages envoyés par les devices et la gestion des utilisateurs, des groupes et des devices. Pour réaliser ce projet, les différentes étapes qui ont été traversées vont être présentées dans ce rapport.

CHAPITRE 2 - PROBLÉMATIQUE ET OBJECTIFS

Dans ce chapitre, nous présenterons la problématique rencontrée par l'entreprise pour qui le projet a été effectué. Une mise en contexte faisant la présentation de l'entreprise sera aussi effectuée. Les objectifs du projet seront par la suite discutés.

2.1 Présentation de l'entreprise

Depuis 20 ans, Matricis accompagne ses clients afin d'optimiser et de mettre en valeur leurs données et processus d'affaires. Ses experts mettent en œuvre des solutions et technologies novatrices par des méthodologies éprouvées, dans le but d'opérer un virage numérique structurant. Que vous souhaitiez intégrer vos actifs informationnels par une architecture infonuagique orientée services avec un ESB, automatiser vos processus d'affaires par l'implantation d'un système BPMS, réduire vos coûts d'exploitation grâce à l'intelligence artificielle ou encore générer de nouveaux revenus via une offre de produits « connectés »; Matricis a l'agilité pour vous conseiller et vous épauler dans votre transformation.

2.2 Problématique

Le projet consiste à créer un outil qui génère dynamiquement des interfaces utilisateur pour un contexte de projet IoT. L'outil utilise du metadata afin de déterminer l'information qui doit être affichée avec quels composants en fonction de la télémétrie provenant d'un équipement IoT. Le tout doit pouvoir se faire sans avoir à redéployer l'application.

Le système qui doit être développé consiste donc à créer un site Web permettant la gestion d'utilisateurs appartenant à différents groupes afin d'avoir accès à des dispositifs disponibles sur Azure. Afin de récolter les données de ces dispositifs, un utilisateur peut configurer l'envoi de messages des dispositifs, et lorsqu'un message sera envoyé par l'équipement, les données seront accessibles pour que l'utilisateur puisse les analyser ou consulter différentes statistiques.

2.3 Objectifs

Matricis avait au préalable effectué une analyse de leurs besoins par rapport à la solution désirée. Les objectifs étaient donc déjà énumérés et comprenaient 3 modules importants, soit le Web API, le Web App et la fonction d'ingestion. Pour chaque module demandé, une liste de fonctionnalités sous la forme de User Stories avait été rédigée et était disponible sur la plateforme DevOps d'Azure.

Modules	**Objectifs**
Interface	En tant qu'administrateur, je veux pouvoir ajouter un message.
	En tant qu'administrateur, je veux être en mesure d'effacer un message.
	En tant qu'utilisateur, je veux pouvoir m'inscrire à l'application.
	En tant qu'administrateur, je veux être en mesure d'effacer un groupe.
	En tant qu'utilisateur, je veux pouvoir réinitialiser mon mot de passe.
	En tant qu'administrateur, je veux être en mesure d'ajouter un device.
	En tant qu'administrateur, je veux être en mesure d'ajouter un groupe et d'y associer des utilisateurs.
	En tant qu'administrateur, je veux être en mesure d'effacer un utilisateur.
	En tant qu'administrateur, je veux être en mesure d'effacer un device.
	En tant qu'administrateur, je veux être en mesure de modifier un device.
	En tant qu'administrateur, je veux être en mesure de modifier un groupe.
	En tant qu'administrateur, je veux être en mesure de modifier un message.
	En tant qu'administrateur, je veux être en mesure de modifier un utilisateur.
	En tant qu'administrateur, je veux être en mesure de voir les groupes.
	En tant qu'administrateur, je veux pouvoir voir les devices.
	En tant qu'administrateur, je veux pouvoir voir les messages.
	En tant qu'administrateur, je veux être en mesure de voir les utilisateurs du système.

	En tant qu'administrateur, je veux qu'un utilisateur ait accès seulement à ce que son groupe a accès.
	En tant qu'utilisateur, je veux pouvoir voir les statistiques de mes devices.
	En tant qu'utilisateur, je veux pouvoir voir le contenu des messages envoyés par mes devices.
API	En tant que développeur, je veux un CRUD API pour les devices
	En tant que développeur, je veux un CRUD API pour les groupes
	En tant que développeur, je veux un CRUD API pour les utilisateurs.
	En tant que développeur, je veux un CRUD API pour les messages.
Ingestion	En tant que développeur, je veux avoir une cache pour la définition des messages.
	En tant qu'utilisateur, je veux que les messages envoyés par mon device soient sauvegardés dans le blob storage.
	En tant qu'administrateur, je veux que les messages non reconnus soient sauvegardés dans le blob storage.
	En tant qu'administrateur, je veux recevoir une notification par email lorsqu'un device envoie plusieurs messages non reconnus.
	En tant qu'utilisateur, je veux que les messages envoyés par mon device soient sauvegardé dans Cosmos DB.
	En tant que développeur, je veux qu'un message reçu du IoT Hub soit désérialisé dynamiquement.
	En tant qu'administrateur, je veux qu'un utilisateur ait accès seulement à ce que son groupe a accès.

Tableau 2. Liste des objectifs

2.4 Composition de l'équipe

L'équipe du projet fut déterminée à l'avance par l'ÉTS dans le cadre du cours LOG795 : Projet de fin d'études. Elle était composée de 5 étudiants dans le baccalauréat en génie logiciel soit : Vincent Labrie, Maxime Lanthier, Thiérry Cheutin-Girard, Guy-Martial Ngowa-Nzali et Francis Groleau. Cependant, lors de la 3e semaine du projet, Thiérry a dû quitter l'équipe et le projet pour des raisons personnelles. De plus, l'équipe fut supervisée par certains employés de Matricis. Le tableau suivant représente les rôles des différents membres de l'équipe.

Nom	**Rôle**
Cédric Melançon	*Lead programmer* chez Matricis. Dans le cadre du projet, Cédric était le Product Owner et la personne ressource pour les questions technique de programmation.
Alex Tremblay	Développeur chez Matricis. Dans le cadre du projet, Alex prenait le rôle de Scrum Master.
Fadi Kaseir	Architecte de solution chez Matricis. Dans le cadre du projet, Fadi était l'expert Angular.
Maxime Lanthier	Développeur, responsable du module d'ingestion.
Francis Groleau	Développeur, responsable du module Web Api.
Guy-Martial Ngowa-Nzali	Développeur, responsable du module de Web App.
Vincent Labrie	Développeur, responsable du Web App.

Tableau 3. Membres du projet

2.5 Risques

Plusieurs risques ont été identifiés au début du projet. Le tableau ci-dessous les énumère.

Risque	Impact	Probabilité	Mitigation / atténuation
Manque d'expérience en développement Angular	Moyen	Moyenne	Des formations et tutoriels ont été suivi par les membres de l'équipe
Manque d'expérience avec la technologie Azure	Moyen	Moyenne	Des formations et tutoriels ont été suivi par les membres de l'équipe
Difficulté de l'implémentation	Haut	Faible	Certains composants semblent difficiles à concevoir dû à notre manque d'expérience, principalement au niveau de la technologie Azure et de l'utilisation de Protobuf, mais des formations et prototypes seront réalisés par les membres de l'équipe.

Tableau 4. Liste des risques

2.6 Livrables

Nom de l'artefact	Description
Code source	Code source de l'application livré à Matricis.
Diagrammes de composants	Diagrammes illustrant l'architecture de l'application (voir Figure 1 et 5).
Diagramme de séquence	Diagramme de séquence de haut niveau illustrant la dynamique de l'ingestion (voir Figure 6).
Présentation du projet	Présentation finale du projet à l'ÉTS avec comme audience d'autres étudiants, des professeurs et deux membres de Matricis (Alex tremblay et Fadi Kaseir).
Rapport technique	Présent rapport. Vise à documenter la gestion, l'analyse, la conception et le développement encadrant ce projet.

Tableau 5. Tableau des livrables reliés à ce projet

CHAPITRE 3 - SOLUTION DÉVELOPPÉE

3.1 Outils et librairies logiciels

Les différents modules comprenaient différentes technologies et des dépendances à des librairies externes. La liste suivante représente les différents outils et librairies logiciels utilisés lors du projet.

3.1.1 Visual Studio Code

Visual Studio Code est un IDE que nous avons utilisé pour le développement du module Web App. Il s'agit d'un éditeur de code supportant les langages de programmation Web, comme l'HTML, le CSS, le Javascript et le Typescript.

3.1.2 Visual Studio

Visual Studio est l'IDE que nous avons utilisé pour le développement du module Web Api. Il s'agit d'un éditeur de code contenant des fonctionnalités de déverminage et de déploiement. Visual Studio comporte aussi différentes extensions permettant de déployer facilement l'API sur Azure.

3.1.3 SQL Server Management Studio

SQL Server Management Studio (SSMS) est un environnement intégré de gestion des infrastructures SQL. SSMS permet de configurer, gérer, administrer et développer tous les composants de SQL Server, Azure SQL Database et SQL Data Warehouse. SSMS fournit un utilitaire unique et complet qui associe un vaste ensemble d'outils graphiques à différents éditeurs de script performants, pour permettre aux développeurs et aux administrateurs de base de données de tous niveaux d'avoir accès à SQL Server.[1] C'est l'outil que nous avons utilisé pour interagir avec la base de données SQL Server.

3.1.4 Azure Portal

Azure Portal est une plateforme Web infonuagique où toutes les composantes logicielles étaient hébergées (Web Api, base de données, Redis, etc).

3.1.5 Azure DevOps

Azure Devops est une plateforme de gestion de projets en ligne. Nous avons utilisé Azure Devops pour faire la gestion et le suivi de projet. Nous avons utilisé le Azure Board afin de créer des tickets représentant les différentes user stories à développer.

---

1 https://docs.microsoft.com/fr-ca/sql/ssms/sql-server-management-studio-ssms?view=sql-server-2017

De plus, nous avons utilisé Azure DevOps pour la gestion de contrôle du code source. Trois repositories ont été créés pour les 3 modules différents.

3.1.6 Azure IoT Hub

IoT Hub est un service de gestion de messages bidirectionnel entre les appareils qui y sont enregistrés. Nous avons utilisé la Azure IoT Hub SDK afin de communiquer avec le IoT Hub depuis la Web Api pour y ajouter des appareils, les modifier et les supprimer.

3.1.7 Azure Blob Storage

Azure Blob Storage est un service de stockage de données nuagique qui permet de stocker différentes sources de données ainsi que des données non structurées. Nous avons utilisé le Blob Storage afin d'y stocker les fichiers de type .cs et de type .dll.

3.1.8 Redis

Redis est un cache de données *In-memory*. Redis fut utilisé afin d'accélérer l'accès aux fichiers de type .dll lors de l'exécution de la fonction d'ingestion.

3.1.9 .Net Core

.Net Core est une libraire C# *open-source* développée et maintenue par Microsoft. Elle permet la création de divers types de projet de programmation. Dans le projet, elle fut utilisée pour le développement du module Web Api.

3.1.9 Entity Framework Core

Entity Framework Core est un ORM permettant le mappage d'objet et de classe C# en tables de base de données relationnelle. Dans le projet, EF Core fut utilisée pour persister les données en lien avec les utilisateurs, groupes et messages.

3.1.10 Identity Framework Core

Identity Framework Core est une sous-librairie contenue dans le cadriciel .Net Core. Elle fut utilisée pour la gestion des utilisateurs, des rôles et des permissions.

3.1.11 Swagger

La librairie Swagger permet de créer dynamiquement une interface Web décrivant la structure d'un API. Elle permet de lister les différents *EndPoints* présents dans le Web Api en énonçant pour chacun : le verbe HTTP à employer, les paramètres d'entrée du *EndPoint* ainsi que son type de retour. De plus, Swagger permet de tester les différents *EndPoints*.

3.1.12 AutoMapper

AutoMapper est une librairie permettant le mappage d'entité en *DTO*.

3.1.13 Angular

Angular est cadriciel côté client, permettant le développement d'interfaces utilisateur séparées en modules et composants. Angular est un cadriciel *open-source* développé et maintenu par Google. C'est le cadriciel que nous avons utilisé pour développer le module de la Web App.

3.1.11 Kendo UI

Kendo UI est une librairie de composantes Javascript facilement intégrable à Angular. Cette librairie a été principalement utilisée pour l'implémentation de grilles dans les interfaces, donc elle a été utilisée dans la majorité des interfaces créées.

3.2 Architecture et conception

La majorité de la conception de l'application avait été réalisée au préalable par Matricis afin de déterminer les différents modules requis pour la réalisation de ce projet. Il ne nous restait donc qu'à concevoir l'implémentation de ces différents modules. L'application sera entièrement hébergée sur la plateforme Azure de Microsoft. Voici la structure finale de l'application ainsi que la description de chacun des éléments :

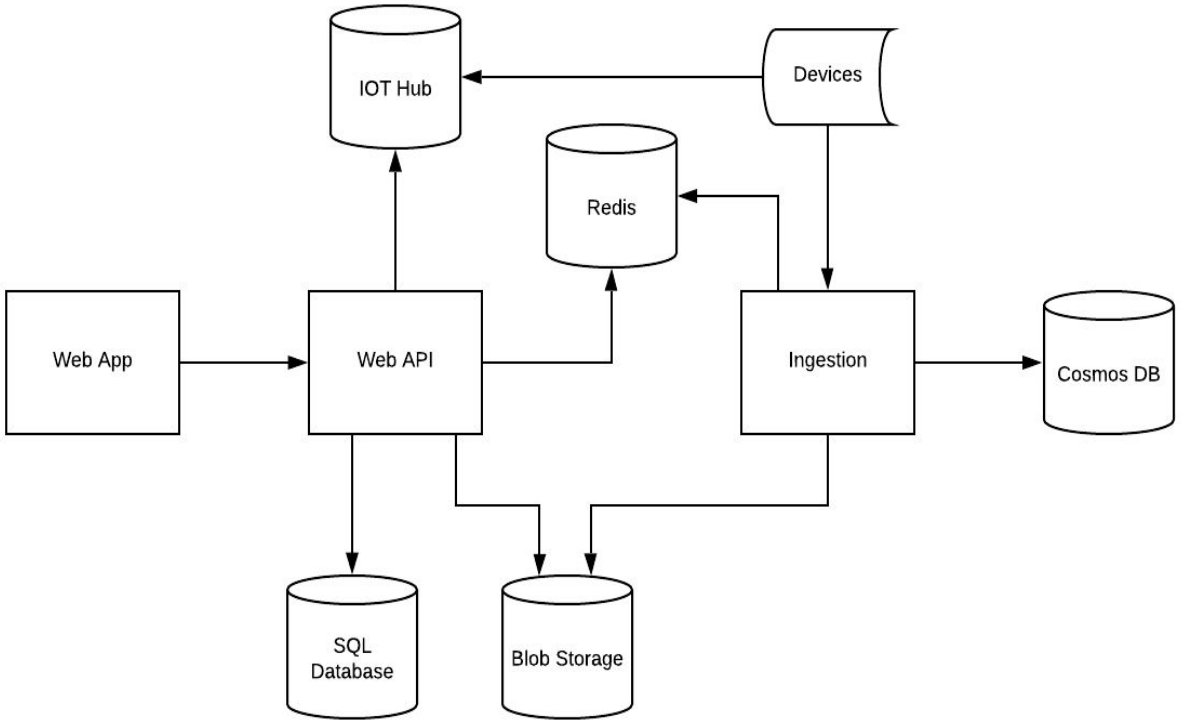

Figure 1. Architecture de l'application

1. **Web App :** La Web App est le point d'interaction entre l'utilisateur et l'application. Elle est constituée d'ensemble d'interfaces servant à gérer les différents aspects de l'application comme les utilisateurs, les groupes et les devices. Son rôle est de permettre aux utilisateurs d'observer et gérer les différents éléments de l'application ainsi que d'ajouter les définitions des messages pour les devices.

2. **Web Api :** L'API connecte l'application aux différentes bases de données. Elle offre les actions nécessaires à la Web App et assure l'authentification des utilisateurs. L'API contient aussi la logique d'affaires pour la conversion des

définitions des messages (fichiers .proto) en un format utilisable par l'ingestion.

3. **SQL Database:** Base de données relationnelle contenant les diverses entités de l'application comme les utilisateurs, les rôles, les groupes, etc.

4. **Devices:** Objets IoT enregistrés dans le IoT Hub qui envoie des données de télémétrie en format Proto. Ces messages sont contrôlés par le IoT Hub et validés par l'ingestion.

5. **IOT Hub :** Service hébergé dans le *Cloud* agissant en tant que concentrateur de messages pour maintenir une communication entre les devices et l'application développée.

6. **Blob Storage:** Stockage de blob intégré à Azure servant à stocker les fichiers .proto sous forme de dll pour pouvoir les utiliser dans le module d'ingestion. Utilisé comme « cold storage ».

7. **Redis :** Cache contenant les fichiers .proto sous forme de dll pouvant être utilisés par l'ingestion. Est utilisé comme « hot storage ».

8. **Cosmos DB :** Base de données intégrée à Azure. Contient les messages provenant des devices ayant été validés par l'ingestion.

3.3 Implémentation

3.3.1 Web App

La Web App a été développée avec Angular. Étant donné les connaissances limitées de l'équipe en ce qui concerne le développement front-end et la complexité peu élevée des interfaces à développer, ce cadriciel nous semblait être la meilleure option. De plus, Angular est une librairie open source et très répandue, ce qui fait en sorte que beaucoup de documentation est disponible en ligne. Notre conception de la Web App s'appuie sur les standards établis d'Angular. Les interfaces sont constituées de composantes indépendantes pouvant être réutilisées dans de multiples interfaces. Prenons par exemple l'interface suivante :

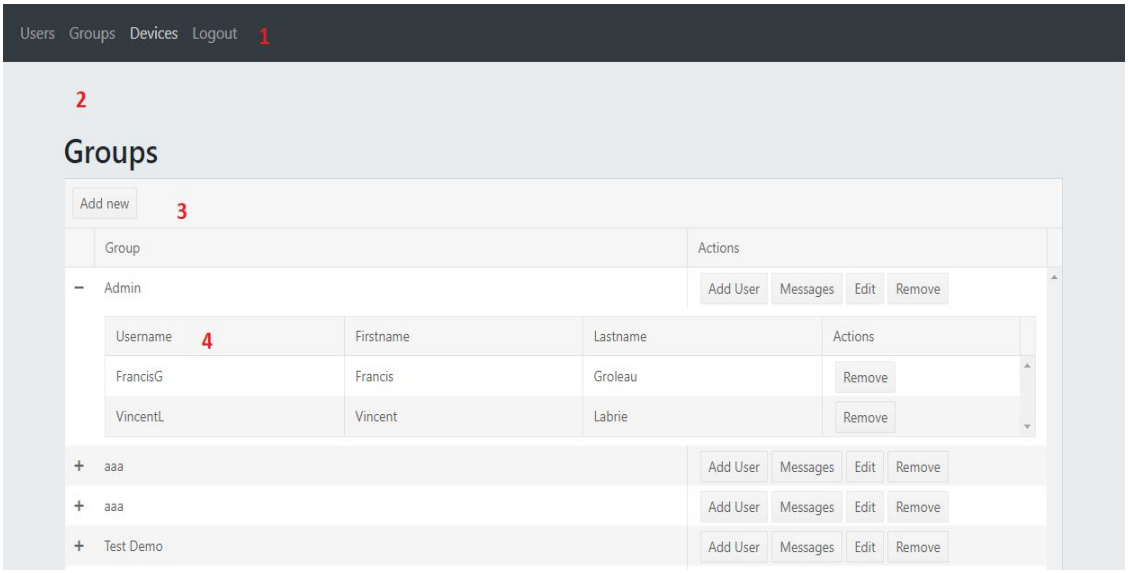

Figure 2. Interface de gestion pour les groupes

Les différentes sections numérotées, soit la barre de navigation, le contenant de la page, la grille et la sous-grille, représentent toutes une composante différente. Les composantes sont aussi regroupées par module. Cette modularité de la Web App nous offre une meilleure modifiabilité et permet à plusieurs développeurs de travailler sur la même interface sans se piler sur les pieds. Pour communiquer avec l'API afin d'accéder aux données, les composantes interagissent avec une couche de service responsable de gérer les différentes requêtes et de souscrire aux observateurs générés par celle-ci afin de réagir aux réponses.

La Figure 2 représente l'interface de gestion des groupes, où il est possible de créer, supprimer ou renommer un groupe, en plus de pouvoir ajouter ou enlever un utilisateur d'un groupe et de lui associer un message.

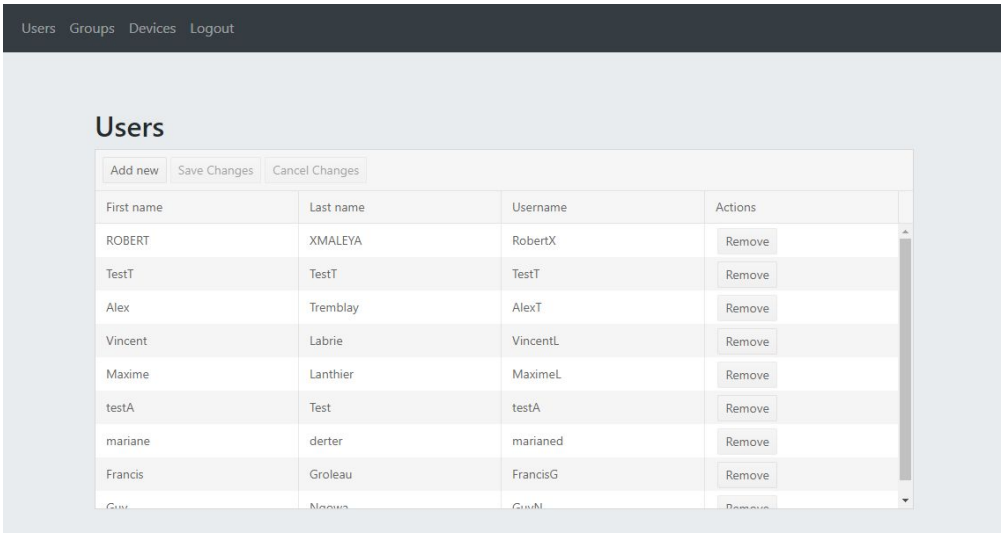

Figure 3. Interface de gestion pour les utilisateurs

La Figure 3 représente l'interface de gestion des utilisateurs, où l'on peut soit en créer, en supprimer ou en modifier.

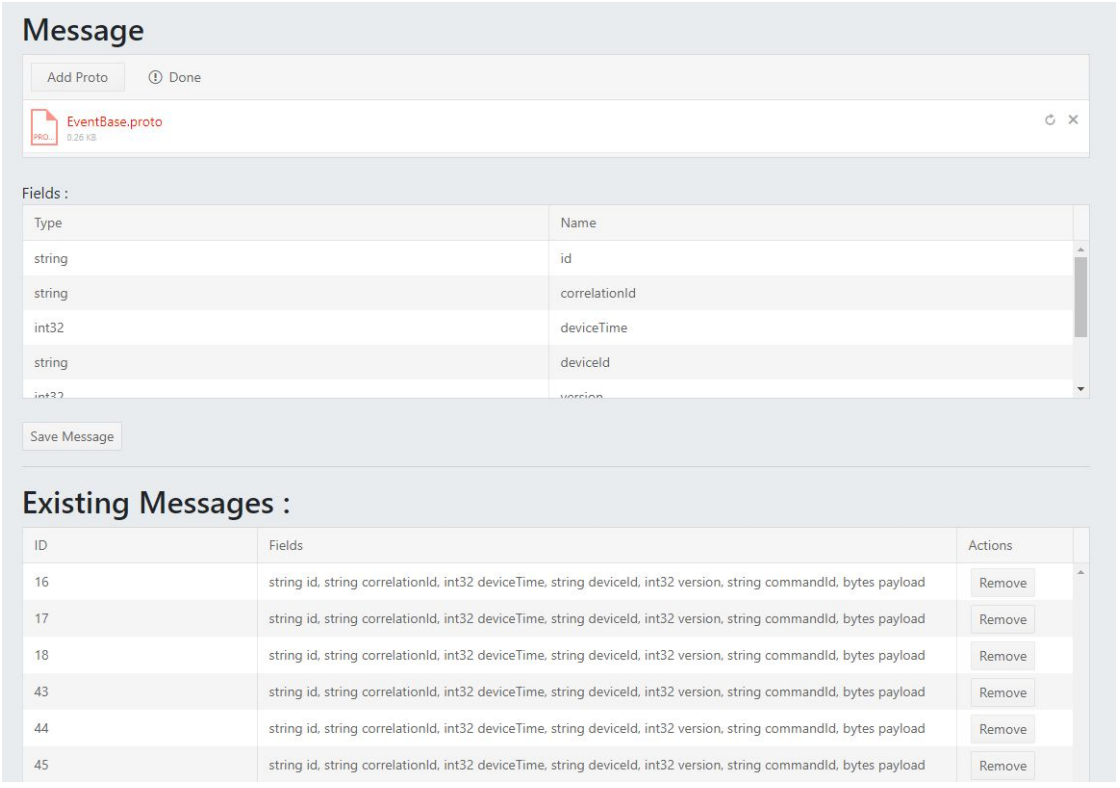

Figure 4. Interface de configuration des messages

La Figure 4 représente l'interface de configuration des messages, où l'on peut configurer le format des messages d'un groupe facilement en téléversant un fichier

.proto. Il est également possible de visualiser tous les messages déjà définis pour un groupe.

Pour ce qui est de la gestion des rôles, un utilisateur connecté au système peut soit être un utilisateur, un admin, ou un *super* admin. Un utilisateur normal ne peut voir que les groupes auxquels il est associé. Il ne peut rien créer, modifier ou supprimer d'autre. Les deux autres rôles permettent de tout voir et de tout faire, la seule différence est que le *super* admin possède les droits pour créer les autres administrateurs.

3.3.2 Web API

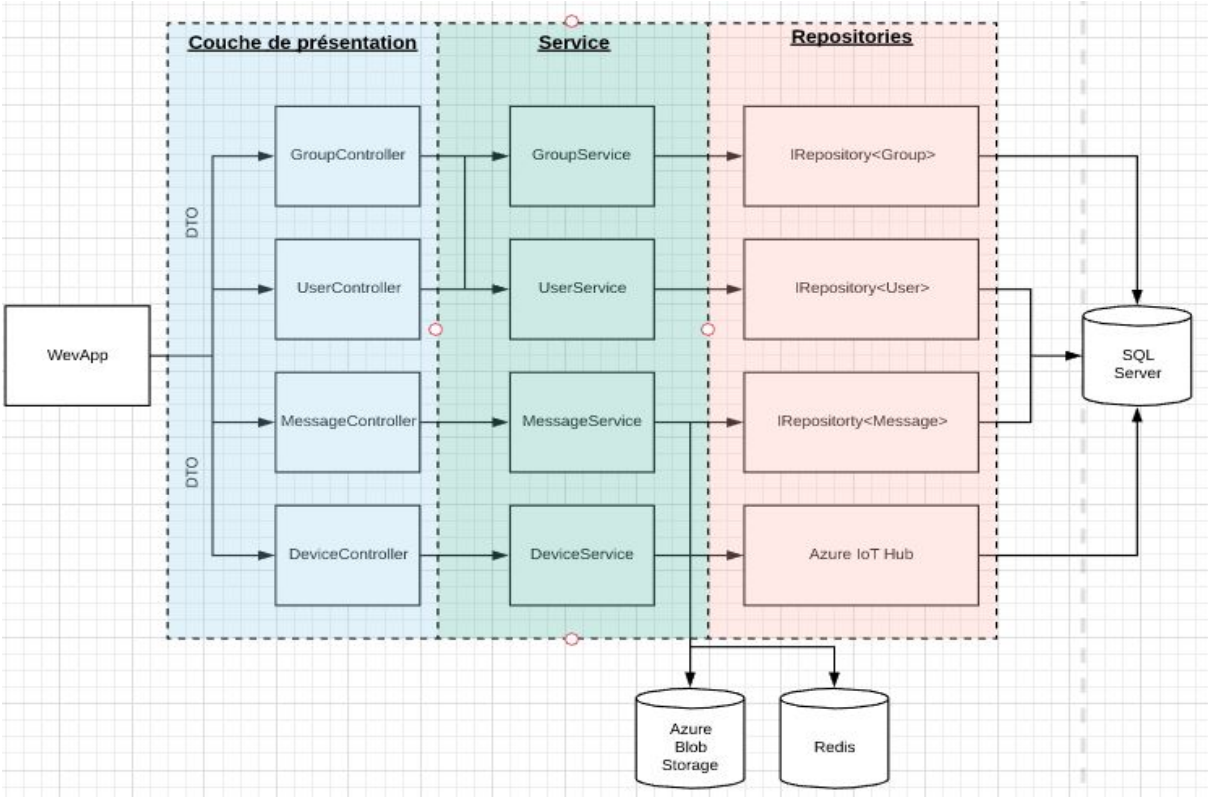

Figure 5. Architecture du Web Api

La Web Api a été développée à l'aide du cadriciel .Net Core. Ce choix technologique a été fait puisque la plupart des membres de l'équipe avaient déjà de l'expérience pour développer en C#, mais ce choix permettait aussi de faciliter l'intégration de notre application avec Azure, car ces deux plateformes ont été développées et sont maintenues par Microsoft. De plus, bien que ce n'était pas vraiment un besoin de notre projet, .Net Core permet la compilation et l'exécution de programmes sur différentes plateformes (Windows ou Linux par exemple). Ce choix est donc également robuste face aux changements possibles dans le futur du projet, offrant une flexibilité non négligeable.

La Figure 5 offre un bon visuel de l'architecture du Web Api. Afin d'avoir une bonne séparation des préoccupations, ce module a été séparé en 3.

La couche de présentation agit comme porte d'entrée au Web Api, on y retrouve les différents *Controllers* de l'application, ceux-ci étant appelés par la Web App. La notion d'injection de dépendances, utilisée un peu partout lors de notre projet, s'y retrouve ici afin de simplifier le code et de garder les *Controller* légers, car ceux-ci ne contiennent aucune logique d'affaires, ils ne font que le lien entre la Web App et la couche de services. Le concept de *DTO* (Data Transfer Object) est également utilisé. Ce patron permet de simplifier le transfert de données et d'isoler les différentes entités de l'application, car un *DTO* est uniquement un conteneur de données, afin d'y avoir accès et de pouvoir les modifier. Afin de facilement faire la création des *DTO* à partir des entités, le cadriciel AutoMapper a été utilisé, ce qui a aidé à sauver du temps de développement.

La couche de services contient principalement la logique métier de l'application. C'est dans celle-ci que les classes sont générées par Protobuf et à partir des messages (les fichiers .proto) envoyés par un utilisateur à l'aide de la Web App. C'est également dans cette couche que nous compilons ces classes à l'aide de Roslyn, un compilateur .Net *open-source*. Cette couche sert d'interface entre la couche de présentation et de données (*Repositories*), respectant ainsi une bonne séparation des préoccupations.

La dernière couche, celle des *Repositories*, permet de lire ou écrire les données. C'est dans celle-ci que les véritables entités se retrouvent. L'ORM *EF Core* a été utilisée afin de faire la transformation d'entités SQL en objet C#. L'approche *code-first* a été utilisée, c'est-à-dire que les objets C# ont d'abord été créés, puis avec l'aide de l'ORM et du principe de migrations, les tables SQL ont été créées dans SQL Server. Cette couche est appelée par la couche des services.

Pour faciliter l'authentification et le droit d'échanges d'information entre la Web App et le Web Api, .Net Core Identity et la notion de *JSON Web Token* (JWT) ont été utilisés. Donc quand un appel à l'API est fait, .Net Core Identity valide les droits de l'utilisateur et procède à l'appel à l'API si l'autorisation a été un succès. Sinon, un message d'erreur est envoyé au client.

De plus, afin de faciliter la consultation, la documentation et la testabilité du Web Api, le cadriciel Swagger a été utilisé.

3.3.3 Ingestion

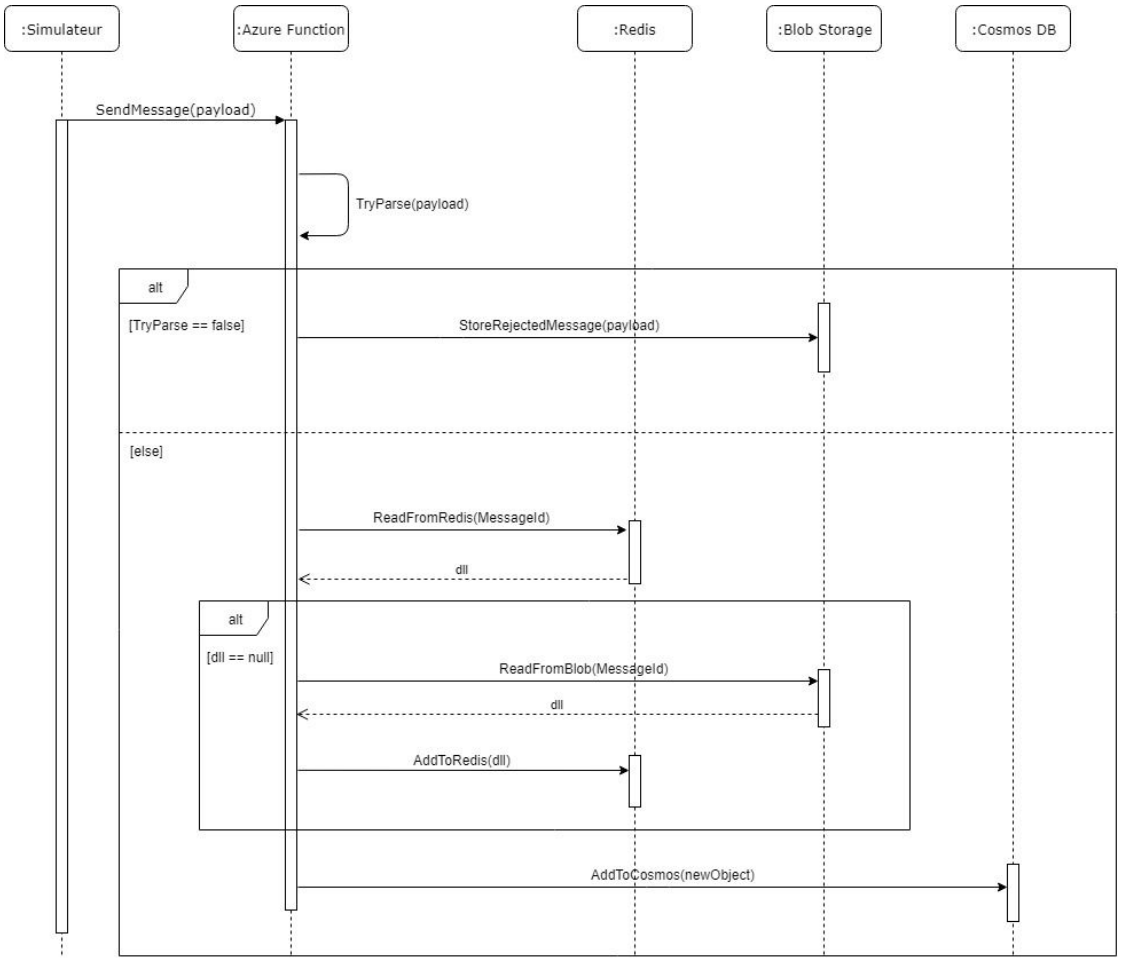

Figure 6. Diagramme de séquence de l'ingestion

La Figure 6 illustre un diagramme de séquence de haut niveau montrant l'interaction entre les différents composants du module de l'ingestion.

L'ingestion des messages se déclenche quand l'Azure function qui écoute l'IoT Hub reçoit un nouveau message envoyé par un équipement quelconque. Sa première tâche est d'essayer de désérialiser le message reçu. Si cela échoue, l'Azure function doit sauvegarder le message reçu dans le Blob Storage.

Une fois la désérialisation réussie, une recherche est effectuée dans Redis afin de trouver la dll associée au message reçu. Si la dll n'est pas trouvée dans Redis, l'Azure function doit aller la chercher dans le Blob Storage afin de l'utiliser et de la sauvegarder dans Redis. Par la suite, il est maintenant possible d'aller lire la *payload* reçue et de la convertir en objet afin d'enregistrer les données envoyées par l'équipement IoT dans Cosmos DB.

Afin de simuler l'envoi de messages d'un dispositif IoT et de tester la logique métier de l'Azure function, un simulateur C# a été fait. Il est possible d'envoyer des messages vers le IoT Hub à l'aide du *SDK* des devices Azure fournit par Microsoft.

3.3.3 Protocol Buffer

Protocol Buffer (*Protobuf*) a été utilisé lors de ce projet afin d'assurer la sérialisation et la désérialisation de données entre les différents composants logiciels. Ce format d'échange de données a été préféré à d'autres comme le *JSON* en raison de sa performance. En effet, au lieu d'envoyer des données lisibles pour des êtres humains, *Protobuf* envoie tout simplement des bytes, ce qui prend beaucoup moins de temps lors d'échanges de données que de traiter des caractères humains sous format *JSON* ou *XML*.

CHAPITRE 4 - SUIVI DE PROJET

4.1 Méthodologie

Pour maximiser la productivité et la compréhension du projet par tous, l'approche *Kanban* a été adoptée. C'est-à-dire qu'ils n'y avaient pas de fonctions ou de rôles (Designer, développeur backend, développeur frontend, testeur, etc) alloués à un individu. Tous les membres de l'équipe ont participé aux différents modules. Cependant, cette approche a trouvé un peu ses limites dans notre contexte. En effet, chaque membre choisissait à la fin d'une tâche une nouvelle tâche sans se fier à un plan de travail prédéfini.

4.2 Azure DevOps

Comme mentionné précédemment, nous avons utilisé Azure DevOps pour effectuer la gestion et le suivi de projet. Lors de ce projet, nous avons suivi les outils ou procédés suivants.

- Le Azure Board. Il fut utilisé afin de faire le suivi, l'assignation et la documentation des différentes fonctionnalités développées dans le projet. Au début de chaque semaine, chaque membre de l'équipe choisissait un ticket correspondant à une *User Story* et le mettait à l'état en développement. Une fois le ticket terminé, celui-ci était mis en revue et devait être révisé par minimalement 2 paires avant d'être approuvé. Lors de la revue de code, différents commentaires et suggestions étaient généralement apportés. Une fois les problèmes résolus, le ticket était mis à terminer.

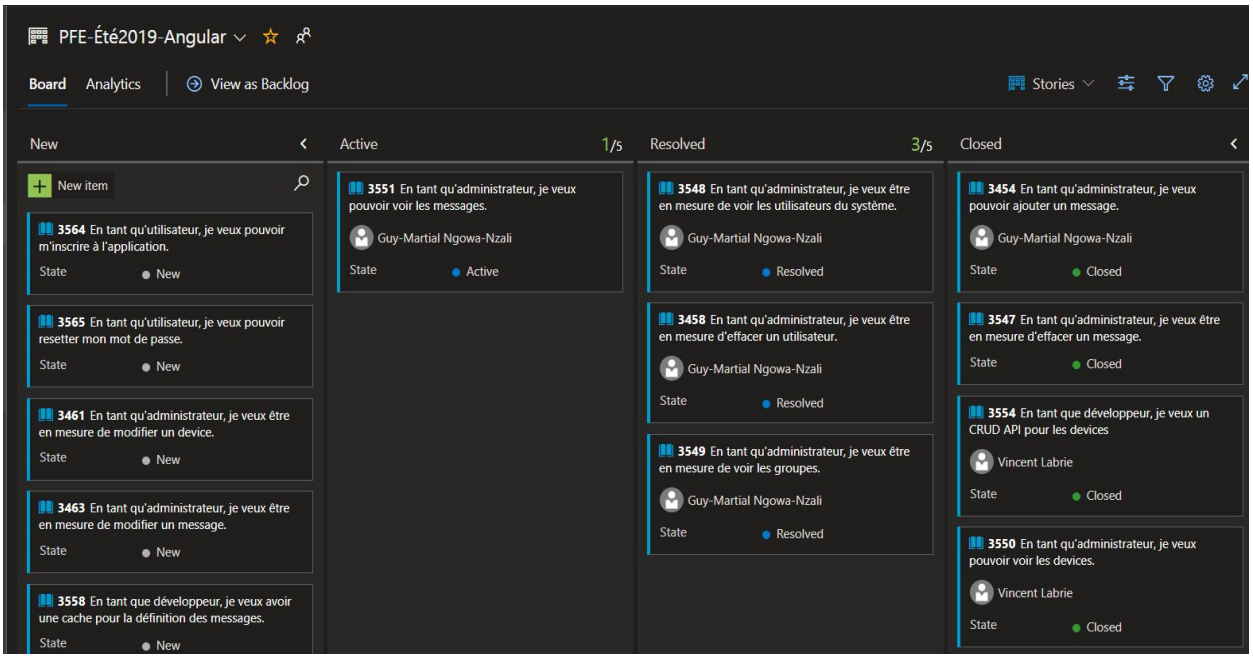

Figure 7. Azure Board

- Gestion de code source sous forme de plusieurs *Repository Git*. Afin de faire le suivi du code développé, nous avons employé le principe développé par la compagnie Atlassian du *Git Flow*. Chaque développeur créait une nouvelle branche afin de développer la fonctionnalité qu'il s'était associée via le Azure Board. Une fois que le développement de la fonctionnalité était terminé, le développeur créait une *pull-request* afin de merger le code dans la branche Develop.

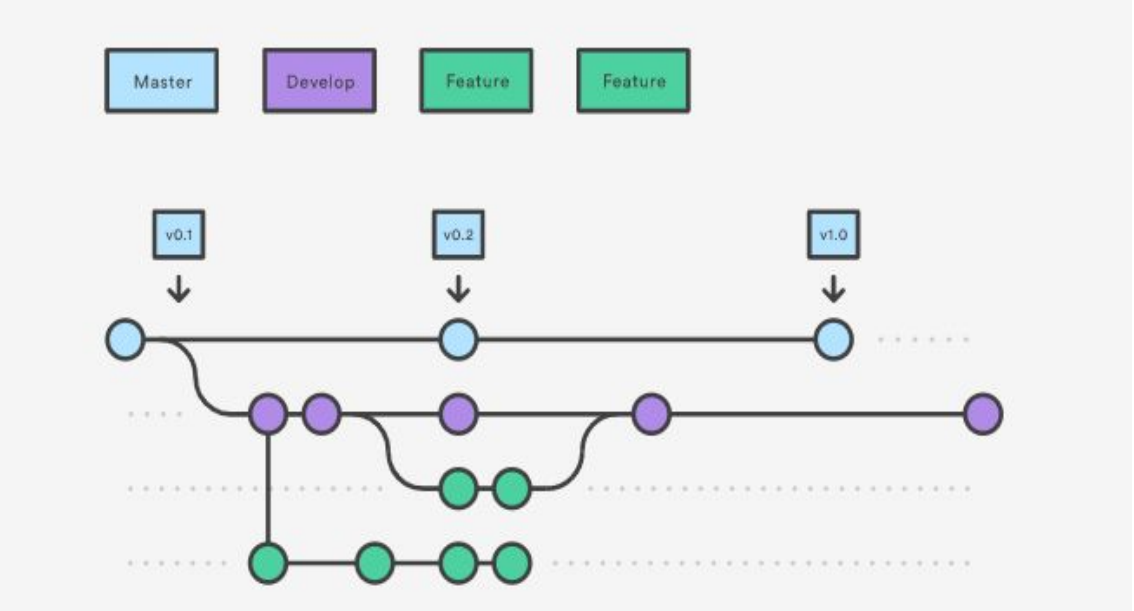

Figure 8. Git Flow
https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow

4.3 Rencontre de suivi de projet

Toutes les semaines étaient marquées par une rencontre dans les locaux de Matricis qui évaluait l'avancement du travail. C'était aussi l'occasion pour nous de poser nos questions, de mettre la lumière sur les difficultés rencontrées lors du développement de la semaine et de s'allouer de nouvelles tâches et/ou de prolonger les tâches qui n'avaient pas été finies.

4.4 Rétrospective de sprint

Après nos rencontres de suivi, nous faisions parfois des rétrospectives de *sprint* afin de suivre les standards énoncés dans la méthodologie Agile. Lors de ces rétrospectives, nous parlions de nos points forts, des points à améliorer et des pistes de solutions possibles pour ceux-ci.

4.5 Communication

Lors du projet, il était important de garder une bonne communication entre les membres de l'équipe de développement et les personnes ressources chez Matricis. Afin de communiquer avec Matricis, un groupe de conversation sur la plateforme de messagerie Slack a été créé. Dans ce groupe, nous pouvions poser nos questions directement et donc, par le fait même, garder un historique des réponses et solutions trouvées. Afin de communiquer entre les membres de l'équipe, nous avons de notre côté créé un groupe Messenger Facebook. Dernièrement, nous avons utilisé le service de messagerie courriel de l'ÉTS afin de communiquer avec Mr. April.

CHAPITRE 5 - AMÉLIORATIONS ET TRAVAUX FUTURS

Maintenant que ce projet de fin d'études arrive à son terme, il est évident qu'une bonne documentation doit être laissée à Matricis pour que l'entreprise puisse faire un bon suivi, comprends ce qui a été fait et ce qu'il reste à faire, afin de pouvoir bien poursuivre le développement de nouveaux besoins en lien avec l'application prototype que nous avons développée.

Pour répondre à ce besoin de documentation et de transfert de connaissances, le présent rapport sera disponible pour les employés de Matricis et le backlog de User Stories sera mis à jour afin de bien cibler les cas d'utilisation qui n'ont pas été couverts. Le tableau ci-dessous résume ce qu'il reste à développer en terme de fonctionnalités.

Modules	**Objectifs**
Interface	En tant qu'administrateur, je veux être en mesure d'effacer un message.
	En tant qu'utilisateur, je veux pouvoir m'inscrire à l'application.
	En tant qu'utilisateur, je veux pouvoir réinitialiser mon mot de passe.
	En tant qu'administrateur, je veux être en mesure de modifier un message.
	En tant qu'administrateur, je veux qu'un utilisateur ait accès seulement à ce que son groupe a accès.
	En tant qu'utilisateur, je veux pouvoir voir les statistiques de mes devices.
	En tant qu'utilisateur, je veux pouvoir voir le contenu des messages envoyés par mes devices.
Ingestion	En tant qu'administrateur, je veux recevoir une notification par email lorsqu'un device envoie plusieurs messages non reconnus.
	En tant que développeur, je veux qu'un message reçu du IoT Hub soit désérialisé dynamiquement.

Tableau 6. Liste des fonctionnalitées non implémentées

Donc, les principales fonctionnalités manquantes par rapport aux objectifs de base sont en lien avec des actions qui ne sont pas permises par l'interface à un utilisateur et à la gestion des rôles qui n'a pas complètement été implémentée. Pour ce qui est du module d'ingestion, celui-ci n'est pas complètement terminé non plus. La désérialisation dynamique et l'interaction avec Redis n'ont pas complètement été implémentées en raison du manque de temps et de compréhension.

CHAPITRE 6 - ENJEUX ENVIRONNEMENTAUX

Nos rencontres se faisaient une fois par semaine. Tout le reste de notre communication se faisait à travers Internet. Ceci nous a permis de diminuer nos moyens de transport physiques (autobus, métro, automobiles, etc), par conséquent, de diminuer nos émissions en CO2 dans la nature. De plus, nous n'avons pas fait de modélisations sur papiers, ce qui nous a permis de diminuer le nombre de papiers utilisés. Lors des réunions chez Matricis, nous séparions nos déchets (biodégradables / non biodégradables). Suite à ces explications, nous pouvons donc dire que nous avons diminué notre emprunt écologique.

CHAPITRE 7 - CONCLUSION

Pour conclure, la réalisation de ce projet a permis aux membres de l'équipe de travailler avec de nouvelles technologies, comme Azure ou Protobuf. Certaines difficultés ont été rencontrées, comme la gestion du temps, la courbe d'apprentissage de nouveaux concepts ou nouvelles technologies et la perte d'un membre de l'équipe en cours de projet. La grande majorité des objectifs ont cependant été atteints. Nous sommes également satisfaits de l'encadrement fourni par Matricis tout le long du projet.

C'était une première expérience de travail avec une plateforme *Cloud* pour chacun de nous, alors nous tenons à remercier Matricis de nous avoir donné la chance de travailler sur un projet de ce type et de cette envergure, en plus de nous avoir laissé une grande liberté par rapport à nos choix technologiques.