Nom : CLOVIS
Prénom : Cyril
Spécialité : IIM

Rapport de stage
4ème année du Cycle Ingénieur à Polytech Paris-Saclay
08/04/2024 – 08/07/2024

Tutrice école : Emmanuelle FRENOUX
Mail: emmanuelle.frenoux@universite-paris-saclay.fr

Développement d'une plateforme d'ingestion de données IoT médicales

Société : UMONS - Faculté Polytechnique de Mons
Adresse : 9 rue du Houdain, 7000, Mons
Responsable organisme d'accueil : Saïd Mahmoudi
Mail : said.mahmoudi@umons.ac.be
Tél : +32 65 37 40 56

Remerciements

Je tiens à remercier et à témoigner toute ma reconnaissance à l'ensemble des membres du service ILIA. Ils m'ont permis de vivre une expérience enrichissante au cours de cette période de stage. Celle-ci fut pleine d'intérêt et m'a permis de m'épanouir pleinement dans cette mission. Pour les connaissances et la maturité que j'ai pu développer à leurs côtés, je tiens à les remercier chaleureusement.

Tout d'abord, je remercie M. Saïd Mahmoudi, mon tuteur, pour m'avoir offert l'opportunité et la chance de réaliser mon stage au sein du service ILIA. Je le remercie aussi, pour son accueil et la confiance qu'il m'a accordée dans la construction de ce projet.

Je remercie Dounia Messaoudi, avec qui j'ai pu échanger lors de mon stage ET3 à Thales LAS France. A l'époque en ET5, le retour de son expérience au sein de ce service m'a donné envie d'en d'en apprendre davantage !

Je remercie aussi Tanguy et Islam pour leurs conseils qui ont su guider mon travail depuis le début de mon stage. Je les remercie également pour leurs remarques pertinentes, qui m'ont très souvent permis de voir les choses autrement. Je remercie aussi Matthis pour la grande aide qu'il m'a apportée en électronique. Cela m'a permis de me sentir plus confiant dans cette matière.

Mon sujet de stage étant la poursuite d'un projet auquel plusieurs personnes ont déjà contribué, je tiens à les remercier. Ainsi, je remercie Martin et Timothé pour l'héritage qu'ils ont laissé sur ce projet. En plus de m'avoir fourni une base solide, ils m'ont permis de mieux comprendre l'utilisation des capteurs.

Je tiens également à remercier Lilian, étudiant à Polytech Grenoble et ancien développeur de RAMI, pour avoir pris le temps de m'expliquer la structure choisie pour la mise en place de RAMI. Je remercie également Thomas Pons, qui est à l'origine du début de l'implémentation de RAMI.

Enfin, je tiens à exprimer à nouveau ma gratitude à l'ensemble de l'équipe pour m'avoir accueilli, intégré et conseillé durant mon stage.

Résumé

Dans le cadre de ma mobilité à l'étranger, j'ai décidé de réaliser mon stage dans le service ILIA de la Faculté Polytechnique de Mons, en Belgique. Ce service, spécialisé principalement dans l'IA, le logiciel et le traitement des données, était l'endroit idéal pour bénéficier d'un enseignement transversal, d'autant plus qu'il applique son expertise aux enjeux contemporains.

Concerné par l'augmentation et le vieillissement de la population mondiale, le service a remis en question la pertinence du modèle médical actuel. En effet, les rendez-vous sont de plus en plus difficiles à obtenir, alors qu'en parallèle ces mêmes consultations coutent de plus en plus chères et sont parfois écourtées.

En 2022, le service a donc décidé de conceptualiser une plateforme nommée RAMI. Cette plateforme utiliserait l'Internet des objets de la santé (HIoT) afin de promouvoir la télémédecine, ainsi que l'IA pour offrir un suivi personnalisé des malades chroniques [1]. Les contraintes sont nombreuses et variées, telles que la sécurité ou la contrainte de temps réel. Malgré tout, le service a décidé de relever ce défi avec une approche utilisant uniquement des logiciels open source.

Ma mission au sein du service consiste à reprendre la structure initiale proposée par les précédents développeurs de RAMI. J'interviens à la fois sur la partie électronique du projet via l'utilisation de capteurs cardiaques en tirant parti de l'Internet des Objets Médicaux et sur la partie web de la plateforme en développement full stack. L'objectif est de proposer une plateforme robuste permettant de visualiser et d'analyser toutes les données reçues. C'est avec l'approche progressive proposée par mon tuteur que je me suis attelé à faire évoluer RAMI un peu plus vers le modèle conceptualisé par le service.

Abstract

As part of my mobility abroad, I decided to complete my internship within the ILIA department of the Faculty of Engineering at Mons, Belgium. This department, primarily specializing in AI, software, and data processing, was the ideal place to receive interdisciplinary training, especially given its application of expertise to contemporary challenges.

Concerned about the increasing and aging global population, the department questioned the relevance of the current medical model. Indeed, appointments are becoming increasingly difficult to obtain, while these same consultations are becoming more expensive and sometimes shortened.

In 2022, the department decided to conceptualize a platform called RAMI. This platform would use the Health Internet of Things (HIoT) to promote telemedicine, as well as AI to provide personalized monitoring for chronic patients [1]. The challenges are numerous and varied, such as security and real-time constraints. Nonetheless, the department has decided to take on this challenge with an approach that relies exclusively on open-source software.

My mission within the department is to build upon the initial structure proposed by the previous RAMI developers. I am involved both in the electronic part of the project by utilizing cardiac sensors through the Medical Internet of Things and in the web part of the platform through full-stack development. The goal is to propose a robust platform that allows for the visualization and analysis of all received data. With the progressive approach suggested by my supervisor, I have been working to bring RAMI closer to the conceptual model envisioned by the department.

Introduction

Actuellement, le vieillissement de la population s'accompagne d'une volonté accrue des personnes âgées de conserver leur indépendance. En revanche, en raison de leur âge, il est essentiel de pouvoir vérifier régulièrement leur état de santé. Afin d'éviter des déplacements et/ou une prise en charge dans un système de santé onéreux, il est crucial de pouvoir suivre ces patients depuis leur domicile. L'objectif est de pouvoir détecter au plus tôt leurs potentiels problèmes de santé.

Ainsi, depuis ces dernières années, on observe un fort développement de l'IoMT (Internet of Medical Things). Il s'agit d'une sous-branche de l'IoT (Internet of Things) spécialisée dans le domaine de la santé. Plus précisément, l'IoMT regroupe tous les dispositifs médicaux connectés à Internet, tels que les capteurs, les moniteurs de santé, les applications mobiles, les implants, les objets connectés portables, etc. L'IoMT vise à offrir une surveillance précise et en temps réel de la santé des patients, à faciliter le diagnostic, le traitement des maladies et à améliorer la qualité des soins de santé en général.

Ces dispositifs présentent également certaines contraintes. En effet, étant donné que ces derniers collectent des **données sensibles sur la santé des individus**, il est crucial de mettre en place des mesures pour garantir la sécurité et la confidentialité de ces informations. De plus, il est impératif que le **mode de transmission des données soit fiable** et que leur traitement soit effectué en **temps réel** afin de ne pas mettre en péril la vie des patients.

Pour répondre à ces problématiques, l'architecture RAMi a été proposée en septembre 2022 dans un article scientifique. Cette architecture temps-réel est destinée au suivi des patients âgés via l'Internet des Objets Médicaux. À ce moment, il s'agissait d'une conceptualisation. Elle intègre un système d'alerte basé sur l'analyse des données en temps réel avec des algorithmes de Machine Learning, permettant d'alerter en cas de problème chez un patient. Par ailleurs, elle répond à toutes les contraintes concernant le respect de la vie privée et la confidentialité des données, notamment en sécurisant les données grâce à la blockchain [1].

Depuis cette conceptualisation, en 2023 une première version appelée RAMi0 a été développée. Cependant, cette version initiale n'était pas entièrement basée sur le concept original. Mon travail consiste à faire évoluer RAMi0 en l'alignant davantage avec le concept de RAMi, tout en préservant les éléments fondamentaux de la première version. Cette transition vise à intégrer pleinement les capacités de l'IoMT tout en respectant les contraintes de sécurité, de confidentialité et de traitement en temps réel des données récupérées.

Dans ce projet, un microcontrôleur connecté à un capteur est chargé de récupérer les données et de les envoyer. À partir de cela, mes contributions et décisions ont porté sur plusieurs aspects essentiels :

- Choix de la technologie de communication des données.
- Définition du format des messages envoyés par les microcontrôleurs.
- Évaluation de l'impact du choix de la technologie de communication sur le backend.
- Transformation et mise à jour du schéma de la base de données.
- Amélioration du frontend avec l'intégration d'un graphique en temps réel, tout en adaptant l'interface aux évolutions du schéma de la base de données.

Table des matières

Tables des figures

1. Présentation de l'organisme d'accueil

1.2. L'université de Mons

1.2.1. La place de l'UMONS au sein du monde universitaire francophone belge

Pour comprendre la forme actuelle de l'université de Mons (UMONS), ainsi que sa position au sein du monde universitaire belge, il convient de commencer par décrire la répartition des langues parlées en fonction des régions en Belgique [2]. Chaque région est elle-même divisée en provinces.

Premièrement, la Belgique est séparée en trois institutions régionales :

- La région flamande
- La région de Bruxelles-Capitale
- La région wallonne

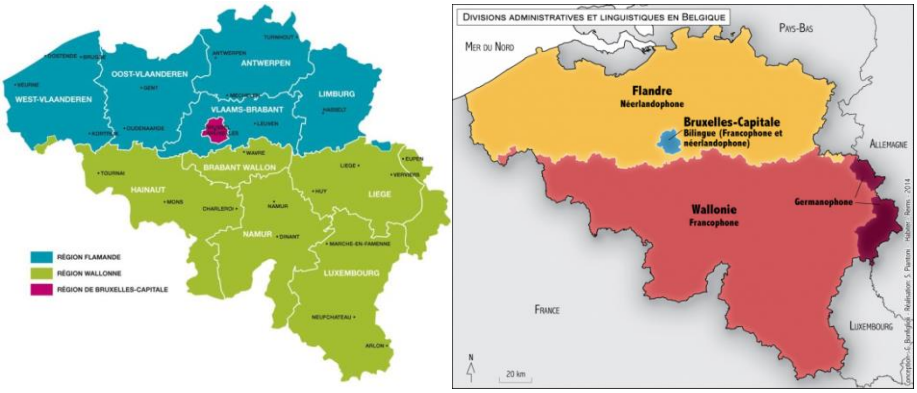


Figure 1 - Répartition des institutions régionales, des provinces et des communautés linguistiques belges (source : bruxelles-j.be et journals.openedition)

Ainsi, les communautés linguistiques francophone, néerlandophone et germanophone se répartissent sur le territoire selon ce découpage. Dans la région flamande, on parle le néerlandais. En région wallonne, la communauté francophone prédomine, avec une présence moindre de la communauté allemande. Enfin, la région de Bruxelles-Capitale est considérée comme bilingue, on y parle alors le français et le néerlandais [2].

Ces régions linguistiques illustrent la diversité linguistique de la Belgique et influencent fortement l'organisation administrative et éducative. Cette organisation permet de garantir la représentation et l'inclusion de chaque communauté dans le système éducatif belge.

C'est donc en Wallonie que l'on retrouve les 6 universités francophones. Voici ces universités, classées par ordre décroissant d'effectifs étudiant: l'Université catholique de Louvain (40 000), l'Université libre de Bruxelles (24 000), l'Université de Liège (20 000), l'Université de Mons dans le Hainaut (10 000), l'Université de Namur (7 000) et l'Université Saint-Louis – Bruxelles (3 000) [3].

Dans ce contexte, l'Université de Mons occupe une position centrale en tant qu'institution d'enseignement supérieur dans la Région wallonne francophone. Elle apporte une contribution significative à l'éducation et à la recherche, enrichissant ainsi la diversité et la richesse du paysage universitaire en Belgique francophone.

1.2.2. Histoire et activités d'enseignement

Pour comprendre l'identité de l'université de Mons, commençons par dresser un aperçu de son historique.

Un des premiers événements marquants dans l'histoire de l'université fut la création de l'École des Mines du Hainaut en 1837. Elle fut fondée par deux jeunes ingénieurs français diplômés de l'École Centrale Paris pour répondre au besoin d'ingénieurs de l'époque. En effet, la ville se développait rapidement. Cette école est ensuite devenue la **Faculté Polytechnique de Mons** et propose des formations touchant à l'énergie, à l'architecture, aux matériaux, à la mécanique et à l'informatique [4]. À l'époque, la faculté ne faisait pas partie de l'université, qui s'appelait l'Université de Mons-Hainaut (UMH).

Par la suite, d'autres instituts et écoles sont devenus des composantes de l'université en devenant à leur tour des facultés. On peut citer par exemple, la faculté des sciences de gestion, de psychologie, de traduction et interprétation, de médecine/pharmacie, d'architecture. L'université peut aussi compter sur ses trois écoles : droit, interprètes internationaux et sciences humaines et sociales [5].

Enfin, la forme actuelle de l'université est le résultat de la fusion entre l'UMH et la Faculté Polytechnique de Mons. C'est la création de l'UMONS [6].

Figure 2 - Carte de la Belgique avec les différents sites de l'UMons et Campus Plaine de Nimy (source : Google Maps et web.umons.ac.be)

Grâce à ces transformations, l'UMONS est en mesure de délivrer les grades de Bachelier, de Master et de Doctorat dans une grande variété de domaines, sur ses campus de Mons et Charleroi [6].

1.2.3. Activités de recherche

En matière de recherche, l'UMONS met en avant et développe son expertise pour contribuer à l'amélioration de la société. Elle peut compter sur l'appui de ses 1 000 chercheurs pour y parvenir. Cependant, son objectif ne se limite pas uniquement à sa région.

Forte de son influence à l'international, l'UMONS est intégrée dans des réseaux de chercheurs internationaux. Par ailleurs, elle est également titulaire du label européen « Research in Excellence ». Enfin, une grande partie de ses activités de recherche est structurée autour de ses dix instituts (Biosciences, Complexys, Énergie, **InforTech**, Langage, Matériaux, Numediart, Risques, Santé, Soci&ter). Généralement, ces instituts sont constitués de 50 à 100 chercheurs [6].

1.2.4. Développement durable

L'UMONS porte une attention particulière aux enjeux environnementaux et sociétaux qui marquent notre époque. C'est le Conseil du Développement Durable (CDD) qui définit la politique à l'échelle de l'université.

Instauré en 2012, il était anciennement connu en tant que Cercle du Développement Durable Institutionnel. Afin de lui accorder davantage d'importance, l'UMONS a fait évoluer cette instance, qui est devenue par la suite le CDD. Ce conseil s'appuie sur une quarantaine de membres permanents et/ou membres de la communauté universitaire pour aborder divers enjeux tels que la gestion des ressources comme l'énergie, des espaces verts et des déchets. Pour atteindre ses objectifs, le conseil s'appuie sur les unités compétentes de chaque composante de l'université, telles que la direction des infrastructures pour le recueil des relevés [7].

- Les missions du CDD :

Les missions du CDD s'organisent autour de quatre points principaux. Premièrement, il s'agit de contribuer à l'intégration du développement durable dans les programmes d'enseignement et de recherche. Deuxièmement, il s'agit de réduire l'empreinte environnementale de l'UMONS. De plus, le conseil sensibilise les membres de la communauté universitaire aux enjeux environnementaux qui sont les nôtres. Enfin, cette sensibilisation se prolonge à travers l'intégration de la dimension environnementale dans les services que l'UMONS propose à la société [7].

- Les objectifs/actions pour y parvenir :

Tout d'abord, en 2023, le conseil d'administration de l'université a adopté **une charte du développement durable** afin de déterminer une ligne directrice claire. Cette dernière tient compte des ODD (Objectifs de développement durable) proposés par les Nations Unies et définit les différents objectifs que l'UMONS devra atteindre.

Le CDD se rend également sur le terrain puisqu'il applique sa politique à travers des groupes de travail. Ces derniers sont organisés afin de pouvoir intervenir sur les thématiques mentionnées précédemment. Parfois même, ces groupes de travail interviennent au-delà de l'université. Par exemple, le groupe dédié aux « interactions avec le territoire » élabore un plan de transition pour l'ensemble des acteurs socio-économiques entourant l'université.

L'UMONS constate sa progression sur l'ensemble de ces objectifs grâce au **rapport de durabilité**. Ce dernier offre un aperçu des actions menées pour rendre le modèle de l'UMONS plus « durable» [7].


CHARTE DU DÉVELOPPEMENT DURABLE
Université de Mons

L'UMONS, consciente de l'urgence à agir face à la crise écologique et des enjeux liés aux 17 objectifs de développement durable définis par les Nations-Unies, souhaite prendre des mesures ambitieuses et assumer sa responsabilité sociétale en tant qu'Université. Pour faire face aux défis actuels et s'affirmer en tant qu'institution résiliente, l'UMONS s'engage dans une démarche exemplaire et responsable vis-à-vis des générations futures.

Cet engagement implique l'ensemble des acteurs de la Communauté universitaire (étudiants, enseignants, chercheurs, personnel technique, ouvrier et administratif, ...) mobilisés dans une culture collective et participative. Soutenu par une gouvernance appropriée, il se concrétise tant au niveau des missions d'enseignement et de recherche qu'à celui de la gestion des campus, de la vie universitaire et des interactions avec la société. Cette volonté d'un engagement fort pour l'avenir se traduit par de nombreuses actions et initiatives, dans le respect des valeurs fondamentales de l'UMONS, telles que proclamées dans sa Charte Qualité.

En termes d'**enseignement** et de **formation**, l'Université s'engage à préparer ses étudiants à agir en acteurs responsables, tant au sein de leur milieu professionnel qu'en tant que citoyens. Ainsi, les chantiers essentiels entrepris permettront de :

- Promouvoir et valoriser l'intégration des enjeux socio écologiques au coeur de la formation, permettant à tous les publics étudiants (formations initiale, spécialisée et continue) d'acquérir les connaissances et compétences indispensables pour faire face, de façon responsable et créative, aux défis des transformations systémiques qui bousculent notre société ;
- Accompagner les enseignants dans cette démarche d'intégration ;
- Promouvoir la transversalité entre cursus, entre Facultés et Ecoles, notamment par le biais de travaux transdisciplinaires et d'événements dédicacés.

En tant que pôle d'excellence en matière de **recherche** et d'**innovation**, l'Université est un acteur clé dans la réponse aux grands défis planétaires, environnementaux et sociaux. Dans un monde nécessitant des processus de transformation accélérés, l'UMONS - tout en gardant son indépendance intellectuelle et tout en soutenant une recherche fondamentale libre - s'engage à:

- Consolider les approches transdisciplinaires existant de longue date dans ses activités de recherche, que ce soit au sein de ses Instituts de recherche ou dans les laboratoires associés ;
- Soutenir les activités de recherche, de création et d'innovation inclusives de la diversité des personnes, des disciplines et des approches ;
- Développer des activités de recherche résolument éthiques et responsables ;
- Stimuler, en collaboration avec le monde économique, les activités de recherche à fort impact en termes de développement durable ;
- Mettre le plus possible le citoyen au coeur des retombées environnementales et socio-économiques de la recherche ;
- Conscientiser les jeunes chercheurs à leur rôle sociétal.


SOMMAIRE

Figure 3 - Charte du développement durable de l'UMONS et rapport de durabilité - 2023 (source web.umons.ac.be)

1.3. L’institut de recherche Infortech

L'Institut de recherche en Technologie de l'Information et Sciences de l'Informatique de l'UMONS, plus connu sous le nom d'Infortech, constitue un centre d'excellence au sein de l'université. Il rassemble environ 120 chercheurs et 35 doctorants, répartis dans 14 services spécialisés [8].

Ces services se concentrent sur divers domaines tels que la détection, le formatage, la transmission, le stockage, l’analyse et l’exploitation des données et signaux, et ce, indépendamment du volume ou du mode de collecte. Afin de mener à bien ces missions, les travaux de recherche de l’institut concernent le Big Data et le Cloud computing, l'IA, le ML et le DL, le génie logiciel et algorithmique, ainsi que le traitement du signal.

Ainsi, les activités menées par Infortech sont caractérisées par leur interdisciplinarité et leur orientation collaborative. Cela implique la coopération avec d'autres organismes à des échelles allant du niveau provincial, régional et parfois jusqu'aux niveaux fédéral, européen et international.

Durant mon stage, j'ai eu l'opportunité de travailler au sein du service ILIA de cet institut.

1.3.1. Le service ILIA (Informatique Logiciel et Intelligence Artificielle)

Le service ILIA a établi ses quartiers au sein de la Faculté Polytechnique de Mons.

Figure 4 - Site de Houdain, faculté polytechnique de Mons (Source : Wikipédia)

- Activités d’enseignement :

D’une part, le service assure des activités d’enseignement dans des disciplines en informatique telles que l’algorithmique, l’apprentissage des langages informatiques comme le C++, Java, Python ainsi que des langages et frameworks du développement web et mobile. D’autre part, il assure l’enseignement de disciplines plus spécifiques comme l’IA, le Machine Learning, la gestion de bases de données, la cybersécurité, le génie logiciel ainsi que la gestion de projets.

- Activités de recherches :

Ce service est actif dans le domaine de la Data Intelligence et du traitement automatique des données. Il détient une expertise dans les domaines de l'IA, du Cloud et Edge Computing, du Big Data, de la vision par ordinateur, de la gestion des données dans l’Internet des objets (IoT) et du traitement d’images. Les membres du service ILIA sont impliqués dans de nombreuses recherches théoriques et appliquées dans leur domaine d’expertise. Ils entretiennent plusieurs collaborations nationales et internationales.

Dans le cadre de mon stage, j’ai travaillé sur l’un des projets de ce service, le projet RAMI. Il s’agit d’une plateforme de collecte des données médicales pour faciliter la prise en charge des patients.

2. Présentation du stage

2.1. La conceptualisation de l'architecture RAMI

L'architecture proposée par RAMi est composée de différentes couches. Tout d'abord, il y a la couche "Things" où les données médicales sont récupérées à partir des capteurs. Les capteurs sont reliés à des microcontrôleurs, qui sont de petits ordinateurs embarqués capables de traiter et de transmettre des données. Ces microcontrôleurs collectent les données et les structurent sous forme de messages en série temporelle. Une série temporelle étant une séquence de données mesurées à différents moments pour suivre l'évolution des valeurs au fil du temps ; les données envoyées sont de la forme **{timestamp, value}**. Ensuite, les microcontrôleurs, transmettent les données via Wi-Fi. Celles-ci rejoignent donc la couche "cloud" ; dans le modèle conceptualisé, c'est le broker MQTT qui rend cet accès possible [1].

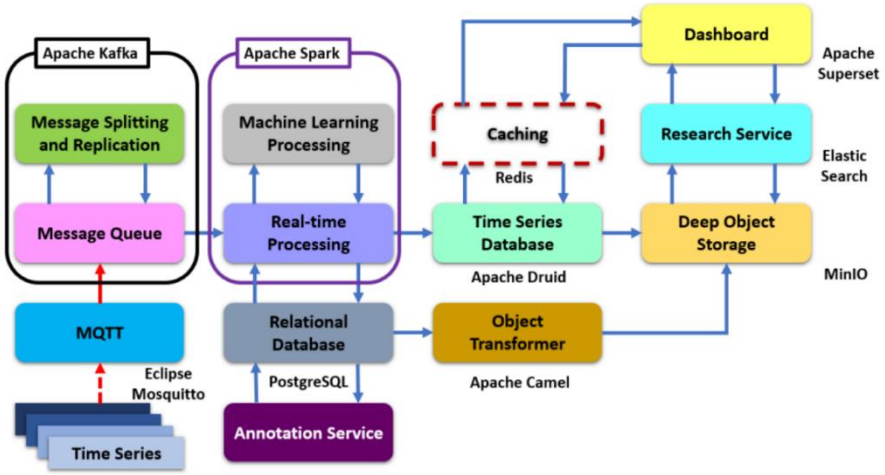


Figure 5 - Schéma des différents logiciels de la partie Cloud de RAMi (source : article sur RAMi p.10 [1])

Dans la deuxième couche, la couche Cloud, les données sont traitées et analysées. Tout d'abord, les données arrivent d'abord dans **Apache Kafka.** La zone tampon proposée par ce dernier assure le stockage temporaire et la distribution des données vers **Apache Spark.** Cette partie est responsable du traitement en temps réel des flux de données via un algorithme de Machine Learning. Pour gérer efficacement les données des séries temporelles, **Apache Druid** est utilisé pour stocker et interroger rapidement cette grande quantité de données. On recourt également à des SGBD plus classiques comme **PostgreSQL**. Étant donné que les données circulent entre différents systèmes, **Apache Camel** est utilisé pour les transformer et les rendre utilisables par ces derniers.

L'étape suivante concerne l'interrogation des données de la partie Cloud de RAMi. Dans un premier temps, on utilise **Redis** pour réduire le temps d'accès aux données fréquemment demandées. Afin de garantir un haut niveau de performance, d'accès et de sécurisation des données, on utilise le système de stockage d'objets haute performance proposé par **MinIO**.

Avec un accès aux données désormais facilité, il est maintenant question de réduire les temps d'attente liés à l'indexation et à la recherche de données, ce qui justifie l'utilisation d'**ElasticSearch**.

Il ne reste plus qu'à visualiser les données. Pour cela, **Apache Superset** est utilisé pour ses tableaux de bord interactifs et réactifs.

L'ensemble constitue une plateforme reposant sur la collaboration de différents composants logiciels open-source, dont la coordination et la synchronisation sont gérées par **Apache ZooKeeper** [1].

2.2. Avancée réelle du projet RAMI

Comme je l'ai mentionné plus tôt, RAMi est un projet qui a été initié par d'autres étudiants. Ils ont été les premiers à commencer l'implémentation de la plateforme, qui n'était jusqu'alors qu'à un stade conceptuel. Dans la suite de ce rapport, je me référerai à RAMi0 pour désigner la version de la plateforme telle que je l'ai reçue au début de mon stage. Les trois étudiants ayant mis en place RAMi0 sont Timothé, pour la partie électronique, et Thomas et Lilian, pour la partie informatique.

Dans cette section, je vais retracer brièvement l'historique des événements qui ont conduit à la conception de RAMi0. Cela permettra de mettre en lumière les défis qu'ils ont rencontrés et d'expliquer la structure actuelle de RAMi0.

Tout d'abord, Thomas est arrivé en premier pour son stage de fin d'études. La première chose que Saïd, mon tuteur, et Thomas ont faite a été de constater l'importance du nombre de logiciels et de microservices proposés par l'architecture RAMi. Ainsi, ils ont décidé de **procéder étape par étape**, en construisant d'abord un modèle fonctionnel avec les éléments essentiels, puis en y ajoutant progressivement des composants et des fonctionnalités.

Ensuite, Timothé et Lilian sont arrivés, et tous les trois ont développé ensemble une API et un site web pour collecter et afficher les données envoyées par des capteurs, donnant ainsi naissance à RAMi0.

Cette approche progressive a grandement influencé le développement du projet et m'a servi de paradigme pour les améliorations que j'ai apportées par la suite.

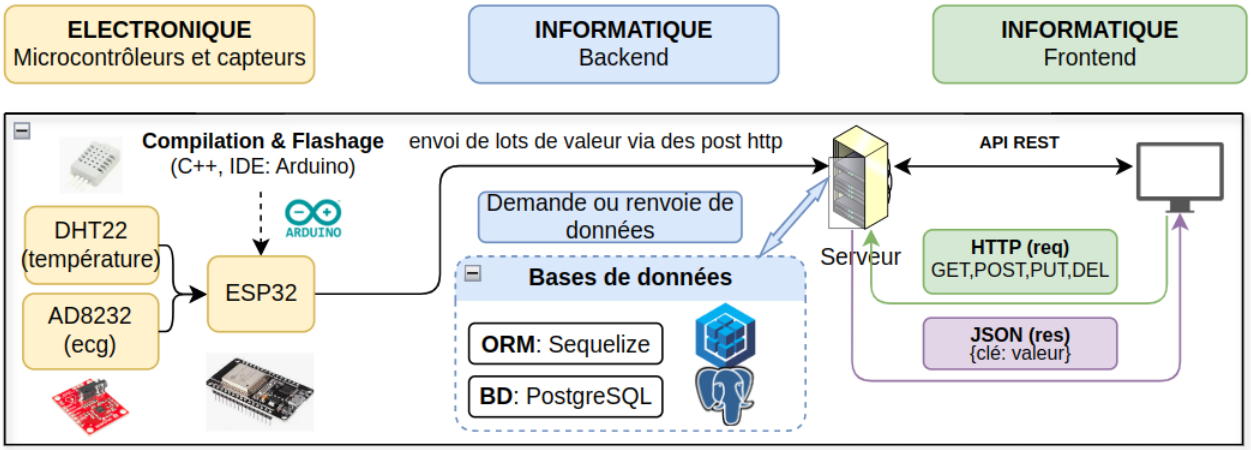


Figure 6 - Structure de RAMI0 telle que je l'ai reçue au début de mon stage

2.3. Descriptif avancé de la structure et des fonctionnalités de RAMi0

Tout d'abord, le projet est entièrement hébergé sur Gitlab. Afin de garantir la qualité et la fiabilité du code, la plateforme met en place des étapes d'intégration continue, comprenant des tests, des builds et des vérifications de conformité. Enfin l'utilisation des conteneurs Docker garantit la portabilité et la répétabilité du déploiement, comme c'est le cas de l'API et de la base de données, tous deux dockerisés.

2.3.1. Fonctionnalités

Tout d'abord, le modèle de la base de données a été conçu « pour la collecte et le stockage des données générées par divers capteurs » [9].

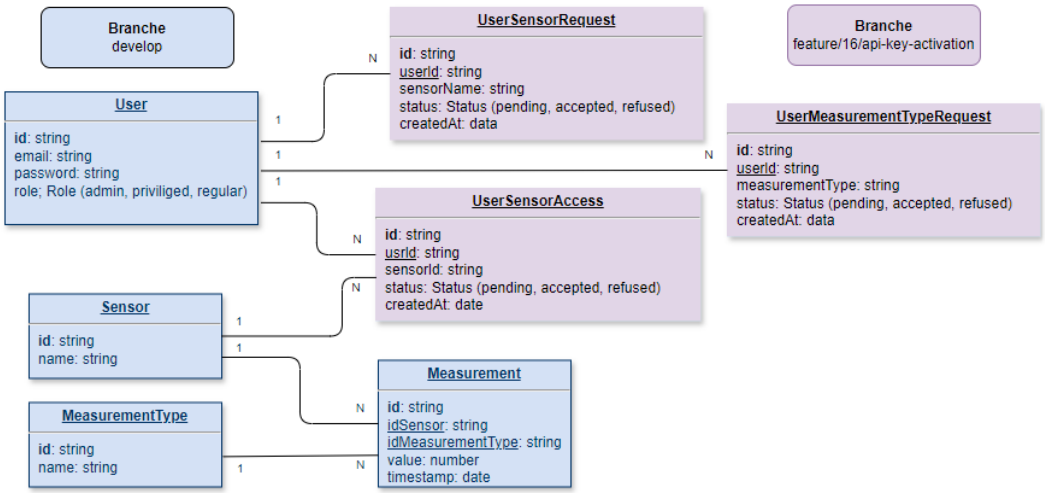


Figure 7 - Schéma de la base de données de RAMi0 (tenant compte des branches Git)

- Tableau de bord (branche develop)

Quant au tableau de bord, les anciens développeurs précisent qu'il : « devrait permettre leur présentation conviviale (des données), en temps réel, pour offrir une compréhension approfondie et une analyse proactive des différents types de mesures » [9]. Toutefois, je me permets de nuancer leur propos car ces graphiques ne propose pas d'affichage en temps-réel. En effet, le graphique doit être recharger manuellement pour voir les valeurs arriver. De plus les graphiques ont été également sujets à d'autres problématiques dont on parlera plus tard.

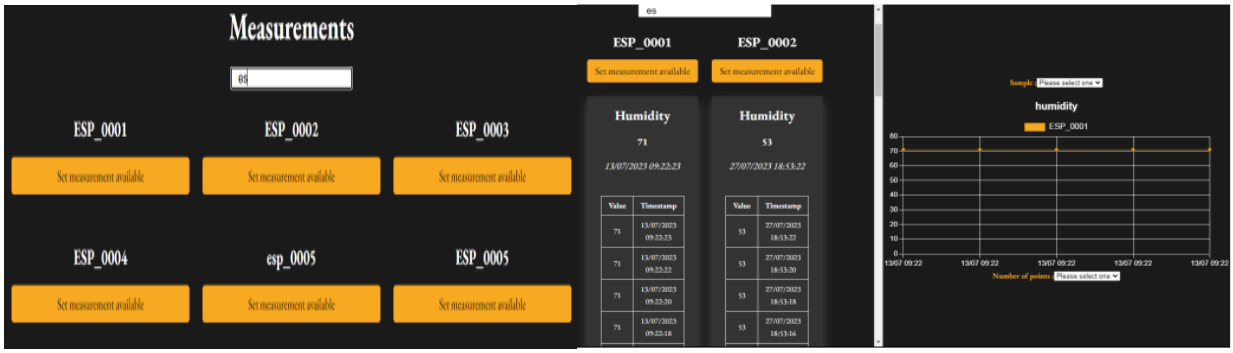


Figure 8 - Vue des capteurs et visualisation graphique des mesures (source: SOLER Lilian - Rapport de stage 4A – 08/08/2023)

- Gestion des accès aux capteurs (branche feature/16/api-key-activation)

D'après le rapport de Lilian: « Cette partie englobe l'authentification qui intervient lors de la création ou de l'accès aux mesures, ainsi qu'aux capteurs et aux types de mesures. Toute création de capteur ou de type de mesure requiert une demande initiale de la part de l'utilisateur, suivie d'une approbation par un administrateur. Une fois approuvées, les mesures générées par les capteurs sont accessibles uniquement aux utilisateurs associés à ces derniers. Une structure de rôles et de gestion de rôles a également été mise en place pour réguler les niveaux d'accès et les autorisations des utilisateurs. »

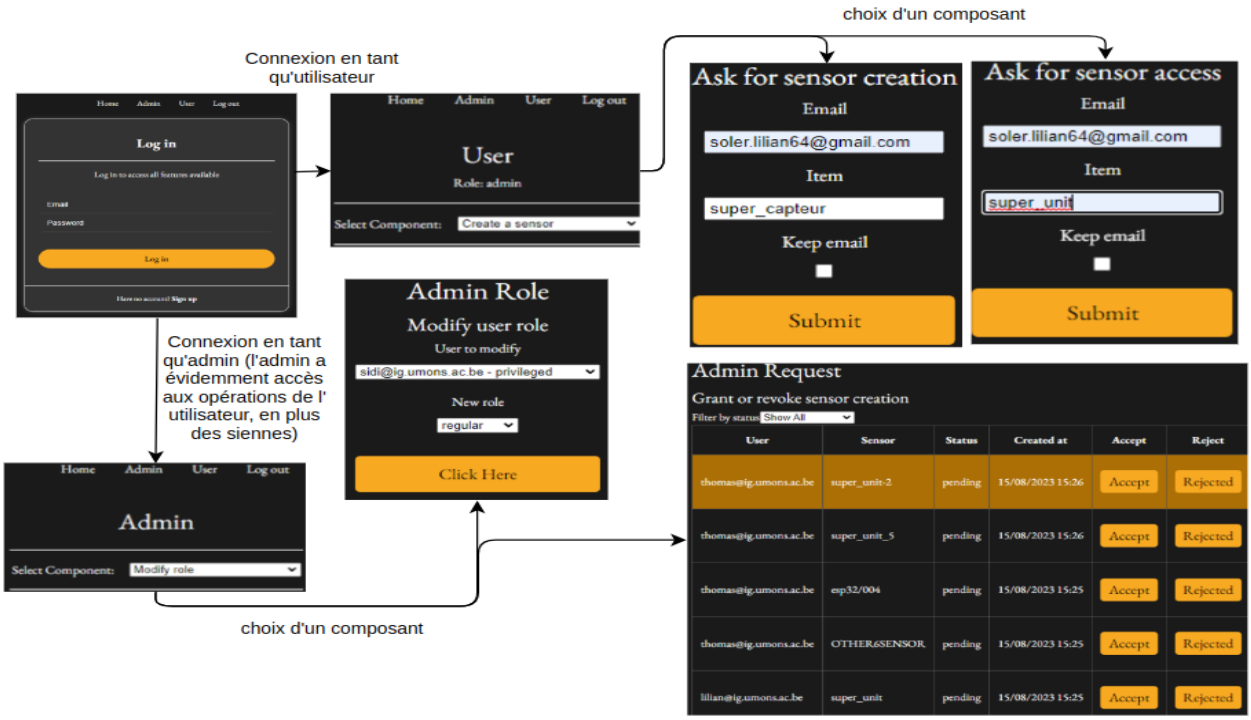


Figure 9 - Ensemble des actions possibles pour les utilisateurs et administrateurs concernant les demandes de création, d'accès et de modération des capteurs

2.3.2. Problématique

Comme nous avons pu le voir, RAMi0 propose de nombreuses fonctionnalités. Cependant, voici les observations qui m'ont été faites peu après mon arrivée :

Premièrement, on m'a fait part de l'impossibilité de constater les données arriver en temps réel sur les graphiques. Effectivement, les graphiques ne s'actualisaient pas à chaque arrivée de valeur ; il était nécessaire de terminer l'envoi des données avant de pouvoir redemander un affichage du graphique. À cela s'ajoutaient les choix d'implémentation des développeurs de RAMi0 : le capteur envoyait les données en temps réel, mais le microcontrôleur ne le faisait pas, stockant plutôt n valeurs du capteur avant de les envoyer par lots [9].

Aussi, avant mon arrivée, le site a été déployé durant un moment. Cela a été suivi par une série de tests, au cours desquels de nombreuses Raspberry Pi ont envoyé des données à l'API de RAMi0. Après ces envois, le prochain test consistait à afficher les graphiques. Ces derniers n'affichaient aucune valeur et restaient dans un état de chargement. Ce problème semblait survenir dans les cas d'envoi massif de données.

Ce sont de réelles problématiques car elles empêchaient RAMi0 de respecter les piliers de la fiabilité des données et de la contrainte du temps réel.

2.4. Les missions de mon stage

2.4.1. Objectif du stage

L'objectif de mon stage consiste avant tout à contribuer au développement de RAMi.

Dans cette logique, mon tuteur m'a fait part de sa volonté d'y **connecter deux capteurs**, un capteur ECG (AD8232) ainsi qu'une ceinture cardiaque équipée d'un capteur BPM POLAR. Je devais également trouver une solution aux problèmes mentionnés précédemment afin d'envoyer et d'afficher les valeurs des capteurs en temps réel sur les graphiques. Concernant le problème de chargement des graphiques lors de l'envoi massif de données, Tanguy m'a conseillé de me renseigner sur les bases de données en séries temporelles, largement utilisées dans le monde de l'IoT et conçues pour stocker de grandes quantités de données.

Saïd m'a également expliqué l'**approche pas à pas** concernant le développement de RAMi. En effet, RAMi0 a souvent utilisé des approches différentes de celles proposées lors de l'étape de conceptualisation ; on peut par exemple citer les POST HTTP de l'ESP32 au lieu de l'utilisation de MQTT. Il semblait donc important de commencer à mettre en place la **transition vers le modèle de conceptualisation initial**.

Il faut donc réussir à reproposer l'ensemble des fonctionnalités de RAMi0 concernant les mesures sous contrainte du respect des trois piliers imposés par l'étape de conceptualisation, c'est-à-dire la fiabilité des données, le respect de la contrainte du temps réel et l'importance de la sécurité des données. Je précise que la politique d'accès aux données reste la même que sous RAMi0 (voir figure 9).

En parallèle de ces missions, mon tuteur m'a indiqué certains détails d'implémentation, comme la mise en place de tableaux de bord. Bien sûr, la mission principale consiste à régler toutes ces problématiques en se rapprochant du modèle de conceptualisation initial.

2.4.2. Limites et enjeux

Une des limites du projet est le fait que la plateforme RAMi0 était déjà bien avancée à mon arrivée. Étant donné que j'ai forké le projet RAMi0, la question était de savoir dans quelle mesure la résolution de ses problèmes et la transition vers le modèle de conceptualisation affecteraient le code déjà existant. Par conséquent, il fallait toujours que je me demande jusqu'où je pouvais aller pour maintenir une cohérence sans altérer la qualité du code déjà existant.

L'autre limite/enjeu est que je reprends le projet seul. J'interviens aussi bien sur la partie électronique qu'informatique du projet. Je reprends le travail de trois étudiants, dont l'un avait des compétences en électronique. Là encore, cela constitue une limite et un enjeu, car il était essentiel que mon manque d'expérience dans ce domaine ne compromette pas la qualité du projet.

Mais bien sûr, les véritables enjeux/limites concernent avant tout les utilisateurs de la plateforme RAMi. Si les données envoyées par les capteurs ne sont pas fiables ou qu'elles arrivent avec un grand temps de latence, les conséquences peuvent être dramatiques. Si l'on se projette dans une utilisation médicale, un patient pourrait avoir des données en décorrélation totale avec la réalité. Cela pourrait fausser le jugement du personnel ainsi que l'entraînement de l'IA qui y sera un jour intégré. Enfin, la sécurité fait aussi partie intégrante des enjeux que la plateforme doit relever.

3. Réalisation de la mission

3.1. Partie électronique

3.1.1. Comparaison HTTP vs MQTT pour l'envoi des données des capteurs via les microcontrôleurs

Tout d'abord, le choix de recourir au protocole MQTT plutôt qu'à l'utilisation de HTTP par les microcontrôleurs était recommandé par le modèle conceptualisé. Cependant, cela ne constitue pas un argument suffisant pour justifier l'abandon de HTTP pour l'envoie des données. C'est pourquoi nous justifions la pertinence de MQTT par rapport à HTTP dans cette sous-partie.

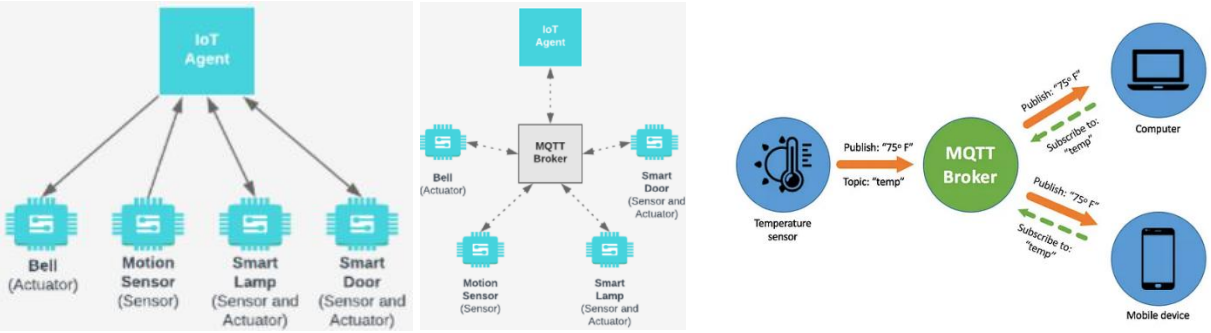


Figure 10 – À gauche, schéma des approches respectives de MQTT et HTTP (source : https://devopedia.org/mqtt#qst-ans-7). À droite, exemple d'envoi d'une température sur le topic "temp" (source : https://nitin-sharma.medium.com/).

Dans ce schéma, on considère que l'IoT agent est notre serveur backend et que les autres nœuds sont des capteurs (dont les valeurs sont publiées par leur microcontrôleurs respectifs).

a) Comparaison de la publication des températures d'un capteur avec MQTT et HTTP

MQTT (Message Queuing Telemetry Transport) est un protocole léger conçu pour les dispositifs IoT, utilisant un modèle publisher-subscriber (pub/sub) avec un broker qui gère la distribution des messages. Un capteur publie les températures sur un topic ("temp"), et les clients intéressés s'abonnent à ce topic. Le broker envoie automatiquement les valeurs publiées aux clients abonnés, permettant une communication efficace 1 à n en temps réel. L'ajout de capteur est trivial, il publie sur un topic, les clients intéressées s'y abonnent !

Certains brokers comme HiveMQ, permettent de se connecter via navigateur avec les "websockets over MQTT". Un client peut ainsi s'abonner à un topic, récupérer les données. Il nous reste à charge l'affichage en temps réel dans un graphique.

HTTP (HyperText Transfer Protocol) fonctionne sur un modèle request-response, où chaque client doit envoyer une requête au serveur pour obtenir les données. Ici, le capteur envoie les températures au serveur, et chaque client (PC, téléphone) doit faire une requête distincte pour récupérer ces données. Ce modèle est plus lourd et moins adapté aux scénarios nécessitant une communication fréquente ou simultanée entre plusieurs dispositifs. Cela implique une communication un-à-un répétée entre le serveur et chaque client.

b) Dernier argument : L'envoi de lots de valeurs par RAMi0 utilisant HTTP

Comme mentionné précédemment, les microcontrôleurs sous RAMi0 n'envoyaient pas les données en temps réel, mais par lots. Les développeurs de RAMi0 se sont rendus compte qu'à chaque envoi de données à l'API, le microcontrôleur devait se reconnecter ; c'est une caractéristique de HTTP.

Ainsi, pour des mesures nécessitant l'envoi de centaines d'informations par seconde, comme avec un ECG, il était impossible de maintenir une communication en temps réel en raison du temps nécessaire pour la reconnexion et la transmission. Pour pallier ce problème, ils ont choisi de stocker un certain nombre de valeurs avant de les envoyer, réduisant ainsi le nombre de reconnexions et le temps d'attente. Cependant, **cette approche entraîne une absence de traitement en temps réel**.

Nom	**MQTT**	**HTTPS**
Paradigme	**Publish-Subscribe** + brocker	**Request-Response (post, put, get, delete)**
Conso énergie/BP	Faible	Elevé
Communication	1 à n (ajout de capteur facile)	1 à 1 (le serveur gère la communication)
Taille des headers	~ 2 bytes	~ 8 bytes
Sécurité	SSL/TLS	TLS
Connexion	Etablie une fois au début et maintenue	Fermeture de la connexion après chaque requête (donc à établir à chaque envoi)

Figure 11 - Tableau récapitulatif des comparaisons entre les protocoles MQTT et HTTPS [10]

Comme le montre ce tableau récapitulatif, MQTT répond bien mieux aux besoins de notre plateforme RAMi.

Ainsi, les différentes explications justifient l'adoption du protocole MQTT pour l'envoi des données des capteurs. Les microcontrôleurs seront donc responsables de se connecter au broker et d'y transmettre les données des capteurs. Avec cette approche, on se rapproche du modèle conceptualisé et surtout, on peut réellement parler de traitement en temps réel.

3.1.2. Conception de l'intégration du Broker MQTT

L'adoption du protocole MQTT entraîne plusieurs conséquences. Tout d'abord, cette transition nécessite de modifier le code des microcontrôleurs et une partie du backend pour la gestion des messages arrivant sur les topics. De plus, profitant de ce changement, j'ai adopté le format des séries temporelles pour l'envoi des messages, conformément au modèle conceptualisé. Enfin, je voulais mettre en place une communication entre les microcontrôleurs et le serveur.

a) Quelle serait la nature de la communication ?

Le serveur envoie les commandes ping, start, et stop au microcontrôleur. Le microcontrôleur répond par pong ou pong.publishing (s'il est déjà en train de publier) selon son état. La commande start déclenche l'envoi de données, stop l'arrête. La communication se fait en JSON.

b) Comment mettre en place la communication ?

Cette question est importante : un seul topic ne suffit pas pour une communication efficace, car microcontrôleur et serveur seraient à la fois subscribers et publishers sur le même topic, recevant leurs propres messages ! On utilise des topics distincts où l'un écoute et l'autre parle, et vice versa. Cette méthode a été choisie pour isoler les échanges entre serveur et microcontrôleur. D'autres solutions existent, détaillées en annexe 3 avec leurs impacts sur la communication.

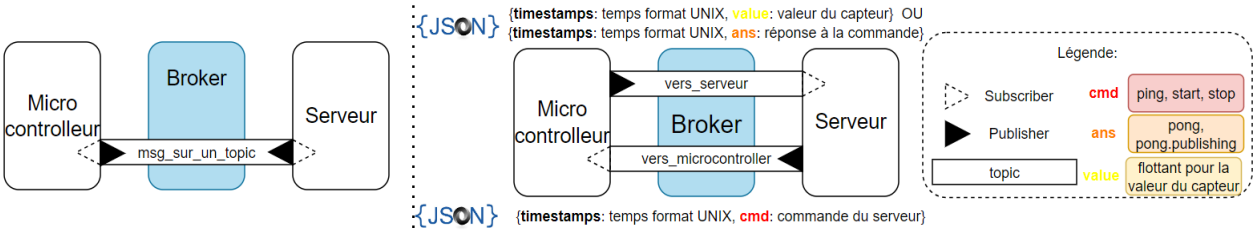


Figure 12 - À gauche : communication entre le microcontrôleur et le serveur avec un seul topic, à droite avec deux topics

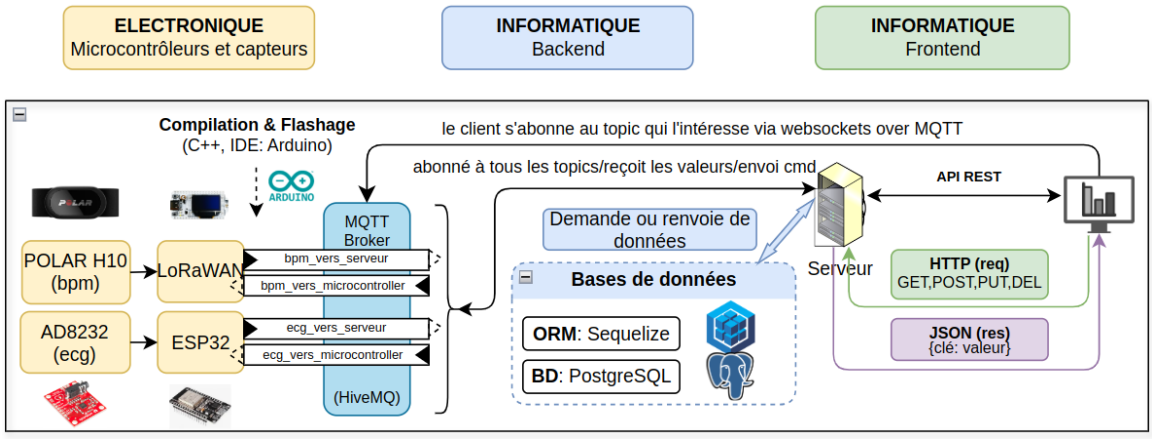


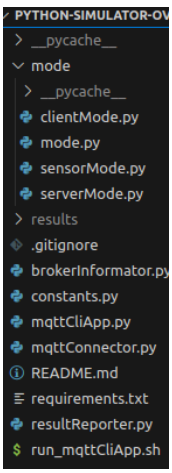
Figure 13 - Structure de RAMI actualisé intégrant le broker MQTT et les topics distincts

Le serveur s'abonne à tous les topics pour recevoir toutes les valeurs envoyées par chaque capteur via les microcontrôleurs et les ajoute dans la base de données. Quant au client, côté navigateur, il peut s'abonner au topic qui l'intéresse via WebSockets over MQTT.

3.1.3. Implémentation d'un simulateur Python

Avant d'implémenter cette solution sur les capteurs, j'ai vérifié que les contraintes fixées par Saïd et le modèle de conceptualisation étaient respectées. Une implémentation directe aurait aussi impliqué des modifications majeures du backend et du frontend, d'où ma décision de procéder par étapes. J'ai donc développé un programme Python pour simuler un capteur se connectant à un broker et envoyant des données sur un topic, ainsi qu'un serveur simulé pour recevoir ces données.

Ce code suit une approche orientée objet, utilisant des classes statiques pour les services et des classes objets pour les différents modes (Mode, SensorMode, ServerMode, ClientMode).



- *run_mqttCliApp.sh* : permet de lancer le programme Python plus facilement.
- *MqttCliApp* : permet l'interaction de l'utilisateur avec le programme dans la console.
- Constant : regroupe l'ensemble des constantes nécessaires au programme.
- *BrokerInformator* : regroupe les informations suivantes : (URL, port, username, password, et les booléens TLS et WS) pour chaque broker déclaré au sein de ce fichier.
- *MqttConnector* : se connecte au broker à partir des informations fournies par la classe BrokerInformator.
- *ResultReporter :* À la fin de la transmission, cette partie s'occupe de consigner dans un fichier Excel les valeurs envoyées par le capteur ainsi que leur moment d'envoi et fait de même pour les valeurs reçues par le serveur. Elle calcule les deltas pour estimer le temps de transit sur le broker et génère un graphique du temps de transmission par rapport au rang d'envoi de la donnée, permettant de vérifier la fiabilité et les contraintes de temps réel.
- Mode : Cette classe mère utilise les services proposés par MqttConnector afin de se connecter au broker. Elle se charge également de mettre en place les "topics distincts" (voir plus haut). Cette dernière met à disposition un attribut liste_time_value_pairs. À chaque message envoyé/reçu, on y enregistre le moment d'arrivée ainsi que la valeur associée.
- ServerMode : Hérite de Mode et est capable d'envoyer des commandes.
- SensorMode : Hérite de Mode et est capable de réagir aux commandes du serveur. On peut également définir la vitesse d'envoi des valeurs.

a) Test pour la vérification de la fiabilité et de la contrainte de temps réel

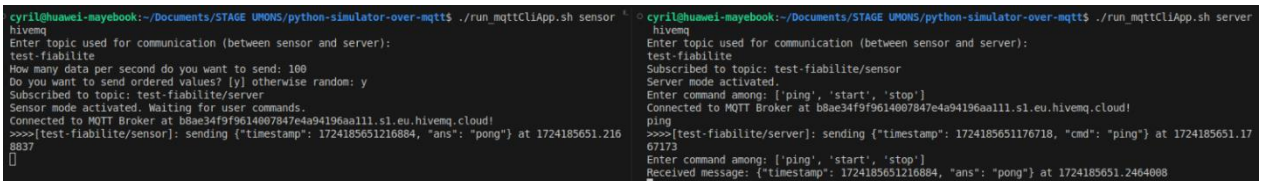


Figure 14 - Mise en place du test, à gauche capteur simulé et à droite serveur simulé

Côté capteur, le test consiste à envoyer des valeurs croissantes à une vitesse de 100 valeurs/sec durant environ 1 minute. Cette vitesse a été choisie car elle représente le débit maximal nécessaire pour un capteur.

Côté serveur, si les données sont reçues dans l'ordre, le système est jugé fiable. Si les délais entre la publication des messages {timestamp, value} et leur réception par le serveur sont acceptables, le système respecte les contraintes de fiabilité et de temps réel.

Mise en place : Pour chaque terminal, on saisit le topic à utiliser pour la communication. Comme je l'ai expliqué plus tôt, on met en place la distinction des topics : le capteur écoute le topic « test-fiabilite/server » alors que le serveur écoute « test-fiabilite/sensor ». Le serveur envoie un « ping », le capteur a bien répondu pong, on peut alors lancer le test avec la commande « start ».

b) Résultats

	A	B	C	D	E
1	sender_time	sender_value	mqtt_trans	receiver_time	receiver_value
2	22:27:31.216884	pong	0,029517	22:27:31.246401	pong
3	22:28:12.528821	start publishing	0,042251	22:28:12.571072	start publishing
4	22:28:12.534150	1	0,118189	22:28:12.652339	1
5	22:28:12.544465	2	0,108018	22:28:12.652483	2
6	22:28:12.554745	3	0,09778	22:28:12.652525	3
7	22:28:12.565126	4	0,087441	22:28:12.652567	4
8	22:28:12.575513	5	0,077089	22:28:12.652602	5
9	22:28:12.586039	6	0,066597	22:28:12.652636	6
10	22:28:12.596570	7	0,056103	22:28:12.652673	7
11	22:28:12.607078	8	0,045639	22:28:12.652717	8
12	22:28:12.617702	9	0,066195	22:28:12.683897	9
13	22:28:12.628092	10	0,055958	22:28:12.684050	10
14	22:28:12.638558	11	0,045554	22:28:12.684112	11
15	22:28:12.648978	12	0,03519	22:28:12.684168	12
16	22:28:12.659264	13	0,049556	22:28:12.708820	13
17	22:28:12.669681	14	0,040643	22:28:12.710324	14
18	22:28:12.679952	15	0,054333	22:28:12.734285	15
19	22:28:12.690356	16	0,044034	22:28:12.734390	16
20	22:28:12.700604	17	0,033841	22:28:12.734445	17
21	22:28:12.710888	18	0,052941	22:28:12.763829	18
22	22:28:12.721140	19	0,042804	22:28:12.763944	19
23	22:28:12.731416	20	0,032569	22:28:12.763985	20
24	22:28:12.741698	21	0,047778	22:28:12.789476	21
25	22:28:12.752106	22	0,037528	22:28:12.789634	22
26	22:28:12.762329	23	0,048989	22:28:12.811318	23
27	22:28:12.772621	24	0,051224	22:28:12.823845	24
28	22:28:12.782927	25	0,041142	22:28:12.824069	25
29	22:28:12.793214	26	0,056341	22:28:12.849555	26
30	22:28:12.803481	27	0,046213	22:28:12.849694	27
31	22:28:12.813933	28	0,06978	22:28:12.883713	28
32	22:28:12.824258	29	0,059625	22:28:12.883883	29
33	22:28:12.834590	30	0,049352	22:28:12.883942	30
34	22:28:12.844953	31	0,039059	22:28:12.884012	31
35	22:28:12.855333	32	0,048078	22:28:12.903411	32
36	22:28:12.865693	33	0,037902	22:28:12.903595	33
37	22:28:12.875980	34	0,053589	22:28:12.929569	34
38	22:28:12.886246	35	0,043452	22:28:12.929698	35
39	22:28:12.896624	36	0,033153	22:28:12.929777	36
40	22:28:12.906919	37	0,045705	22:28:12.952624	37
41	22:28:12.917219	38	0,035641	22:28:12.952860	38
42	22:28:12.927611	39	0,047736	22:28:12.975347	39

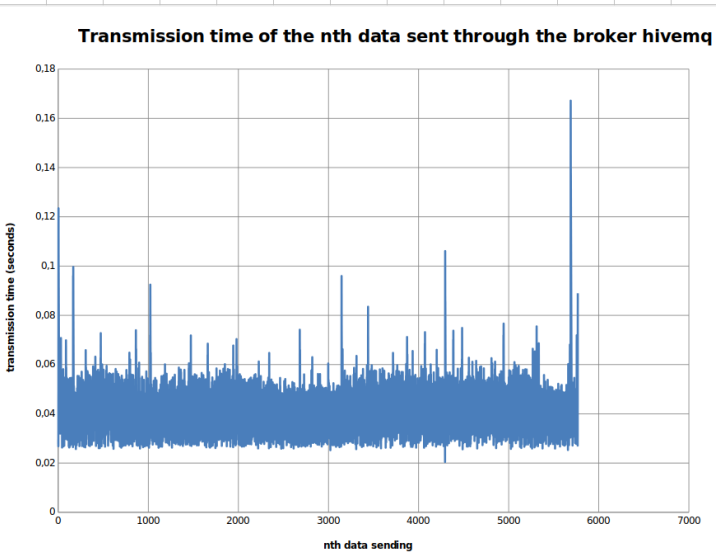


Figure 15 - Résultat du test

On remarque dans l'ensemble du fichier Excel que toutes les valeurs restent dans l'ordre. Compte tenu de la vitesse à laquelle les données ont été envoyées, on peut considérer que le test de fiabilité est concluant. Quant au test de contrainte de temps réel, nous pouvons nous référer au graphique de droite. Celui-ci montre que la plupart des valeurs circulent du capteur vers le serveur avec un délai compris entre 0,02 et 0,09 seconde. Enfin, la sécurité est garantie car les données sont envoyées avec TLS (Transport Layer Security) ; un protocole de sécurité qui protège les données échangées.

3.1.4. Implémentation du code C++ au sein des deux capteurs

Après avoir démontré que la structure fonctionnait bien, j'ai travaillé sur deux paires (capteur, microcontrôleur) : Le capteur ECG AD8232 avec l'ESP32 et le capteur BPM POLAR avec le LoRaWAN32.

Nom	AD8232	Capteur bpm POLAR H10
Description	Module de capteur ECG (électrocardiogramme) compact et faible puissance, conçu pour extraire, amplifier et filtrer les signaux bioélectriques des battements cardiaques humains.	Utilise la détection optique pour fournir des lectures précises et en temps réel des battements par minute, souvent utilisé pour le suivi des performances sportives et des activités physiques.
Utilisation	Connexion "physique" à une esp32. On récupère les données analogiques du capteur sur le pin 34 de l'esp32.	Connexion Bluetooth à une LoRaWAN. La LoRaWAN récupère alors les données envoyées par le capteur POLAR.
Flux de données	Centaines de valeurs/sec pendant 5 à 10 minutes.	1 valeur/sec pendant quelques secondes à 1 minute.

Figure 16 - Tableau présentant les caractéristiques des capteurs AD8232 et POLAR H10

Comme indiqué, l'ESP32 nécessite une connexion physique avec le capteur. Selon les instructions laissées par Timothé, la sensibilité élevée du capteur AD8232 (de l'ordre du mV) nécessite une pile et un convertisseur pour l'alimentation (voir ci-dessous).

Figure 17 - Utilisation des couples (AD8232-ESP32), (POLAR H10- LoRaWAN32) et structure de "sensors-over-mqtt", le dossier regroupant le code de tous les microcontrolleurs

Les microcontrôleurs sont reliés à mon PC via un câble USB (voir ci-dessus), ce qui me permet de les alimenter et de flasher le code compilé. Le code d'interaction avec le broker étant commun à tous les microcontrôleurs, j'ai donc créé le fichier MQTTCommonOperations.hpp/.cpp. Le fichier SpecificConstants.hpp/.cpp, quant à lui, contient les constantes spécifiques à chaque microcontrôleur. On y retrouve le nom et le mot de passe du Wi-Fi et du broker, le port, les topics distincts, le nombre de valeurs par seconde à envoyer, etc. Voir annexe 2, pour plus d'information sur le code des microcontrôleurs.

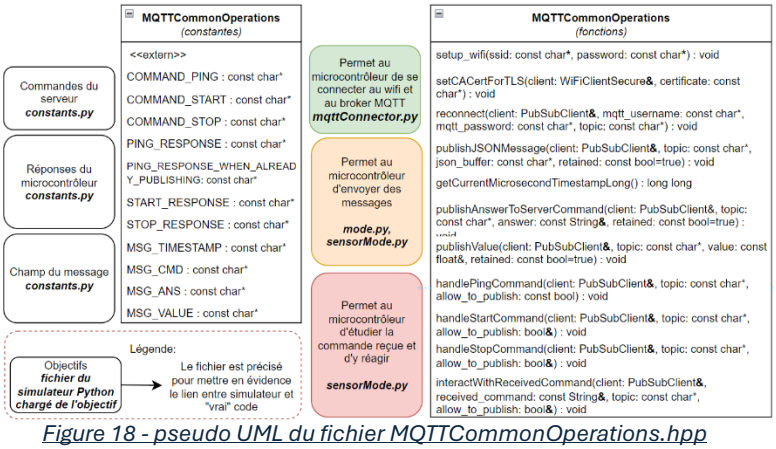


Figure 18 - pseudo UML du fichier MQTTCommonOperations.hpp

Étant donné les fortes contraintes en termes de ressources, notamment en terme de mémoire, j'ai opté pour un passage par référence et par pointeur dans la rédaction des fonctions. Cette approche permet d'économiser les ressources en évitant des copies inutiles. De plus, l'utilisation du mot-clé « const » assure la sécurité en empêchant toute modification involontaire des données.

a) Résultats de l'interaction entre l'ESP32 et le serveur simulé

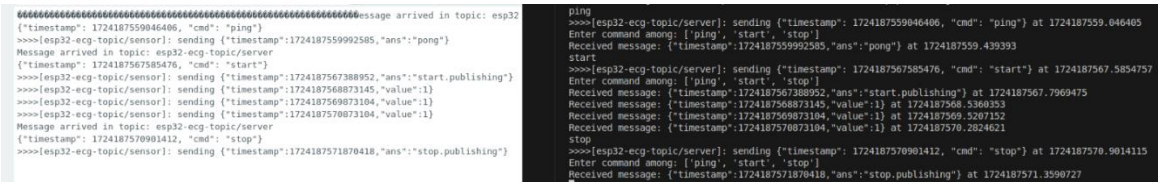


Figure 19 - Interaction entre une ESP32 à gauche et serveur simulé, à droite

On observe que le capteur à gauche reçoit correctement les commandes de l'utilisateur et envoie les valeurs lorsqu'il reçoit la commande « start ». Je précise que dans cet exemple, les valeurs envoyées sont intentionnellement des constantes. Je voulais juste vérifier que le capteur échangeait correctement avec le serveur simulé. Ensuite, j'ai pu mettre en place la structure complète, permettant au microcontrôleur de récupérer les valeurs du capteur (voir ci-dessous). L'IDE Arduino permet de visualiser les valeurs envoyées par les microcontrôleurs sous forme de graphiques :

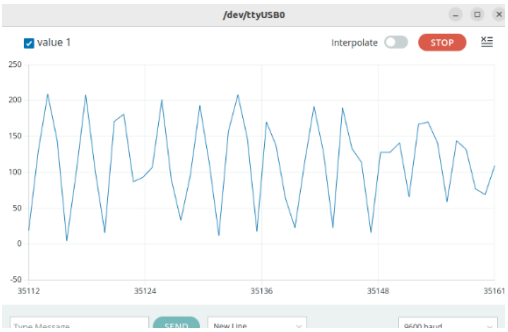


Figure 20 - Graphique "Serial Plotter" des valeurs transmises par le couple AD8232-ESP32, une fois les électrodes en place

Malheureusement, le bruit rend le capteur ECG inutilisable, malgré la détection correcte des tensions aux bornes de chaque composant avec un multimètre. J'ai également tester différents capteurs, piles, électrodes … Cependant, l'extraction des valeurs fonctionne bien.

En revanche, pour le capteur POLAR, tout fonctionne parfaitement.

À la fin de la phase dédiée à l'électronique, je suis passé au développement backend. La première mission consistait à établir la communication avec les capteurs et à la tester à l'aide du serveur simulé.

3.2. Partie backend

3.2.1. Structure générale du backend de RAMI0

Comme mentionné précédemment, j'ai commencé par étudier la structure de RAMI0 avant d'y apporter ma contribution. Les technologies utilisées sont les suivantes : Express.js, Javascript et TypeScript, Docker. Voici les étapes d'un cycle de développement en backend :

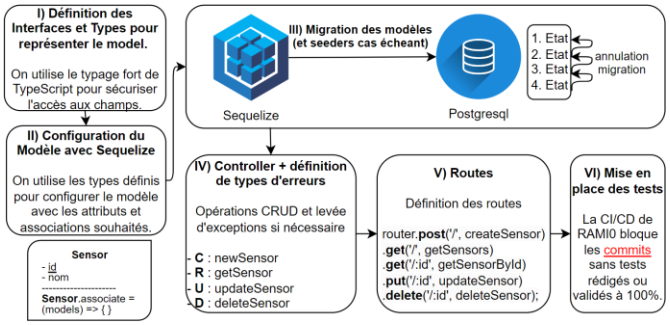


Figure 21 - Étapes d'un cycle de développement backend

Connaissant la structure, je devais maintenant proposer une solution pour les points suivantes :

- Comment mettre en place le broker côté backend et gérer les évènements ?
- Comment faire pour qu'un client puisse utiliser un capteur ?
- Comment mettre en place la base de données en séries temporelles (en réponse au problème de lenteur mentionné précédemment) ?

Nous répondrons à ces questions dans les sous-parties correspondantes. Bien sûr, pour chaque ajout de nouvelles tables ou de champs, j'ai dû repasser par toutes les étapes du développement backend !

3.2.2. Implémentation du broker MQTT côté backend et mise à jour de la table « Sensor »

Premièrement, pour le broker, j'ai envisagé de créer une classe MqttServer. Comme pour le simulateur, cette classe permet au serveur de se connecter au broker, de s'abonner au topic et d'envoyer des commandes. De plus, cette classe gère les événements de connexion, de déconnexion et d'erreur, afin de rendre la connexion du serveur au broker plus résiliente. Comme mentionné sur la figure 13, le serveur s'abonne à tous les topics afin de pouvoir recevoir les valeurs des capteurs. Pour cela, j'ai ajouté un champ topic dans la table « Sensor ». Le topic dans la base de données ne tient pas compte de l'idée des « topics distincts » mentionnée précédemment. Ainsi, cette classe gère la transcription topic vers topics/sensor ou topic/server selon le besoin.

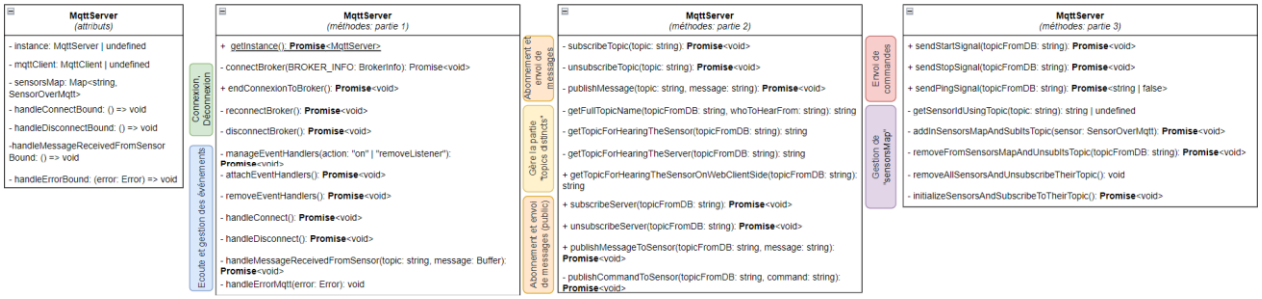


Figure 22 - UML de la classe MqttServer

J'insiste ici sur le nombre de fonctions asynchrones, celles qui attendent la résolution d'une promesse. MQTT étant une bibliothèque asynchrone, cela a considérablement complexifié les tests.

Bien que l'analyse ne soit pas exhaustive de cet UML, j'aimerais au moins aborder un point. Lors de la première connexion, on rassemble l'ensemble des topics de chaque capteur dans le dictionnaire **<string, SensorOverMqtt>**, où la clé correspond au topic du capteur. Ainsi, lors de la réception d'un message, on récupère le topic. Cette clé nous permet de récupérer l'objet SensorOverMqtt dont les attributs sont id et topic. Enfin, une fois l'id, le timestamp et la valeur récupérés, on appelle la fonction permettant l'ajout dans la bases de données. La partie ci-dessous détail ce fonctionnement.

3.2.3. Ajout de l'hypertable « sensordata » et de la table « Session » pour l'utilisation des capteurs

- Problème des données massives

Pour résoudre le problème de stockage des données massives, j'ai opté pour une base de données en séries temporelles, spécialement conçue pour gérer de grandes quantités de valeurs variant dans le temps. Parmi les options disponibles, j'ai choisi TimescaleDB, car elle est une extension de PostgreSQL. Cela garantit que tout le code écrit en PostgreSQL reste compatible avec TimescaleDB.

TimescaleDB offre une fonctionnalité appelée « hypertables ». Elles transforment une table PostgreSQL classique en une table partitionnée et la gère automatiquement [11]. Pour créer une hypertable, on commence par créer une table PostgreSQL classique, puis on utilise la fonction *create_hypertable(nomTable, champDePartitionnement).* Cette fonction convertit la table en hypertable, où champDePartitionnement est généralement une colonne temporelle comme time ou une colonne géospatiale, et la table est partitionnée en conséquence.

Dans notre cas, j'ai créé l'hypertable « sensordata » avec les champs time, id, et value, représentant la valeur reçue à un instant « time » par le capteur identifié par « id ». Le champ time est utilisé comme clé de partitionnement.

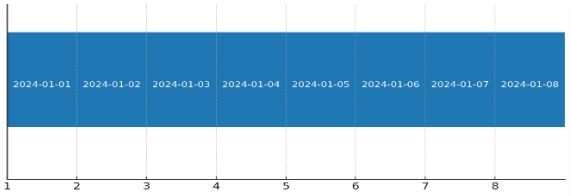


Figure 23 - Exemple de partitionnement de l'hypertable "sensordata"

TimescaleDB utilise différentes stratégies pour partitionner les données dans une hypertable. Si aucune durée n'est spécifiée, le système ajuste automatiquement la taille des partitions en fonction de la quantité de données insérées et de la fréquence des requêtes. Il est également possible de définir manuellement la taille des partitions. Par exemple, dans le schéma, les données sont partitionnées par jour.

- Utilisation des capteurs

Tout d'abord, j'ai défini la règle suivante : à un instant t, un capteur ne peut être utilisé que par une seule personne. Pour représenter cela, j'ai conçu le modèle "Session" qui relie un utilisateur à un capteur pour une période allant de t1 à t2. Il reste à déterminer à quel moment une session est validée, c'est-à-dire à quel moment elle est inscrite dans la base de données. On en parlera dans la partie frontend. Le modèle « session » est donc en mesure d'interroger la table « sensordata » pour obtenir les données correspondant à l'intervalle de temps pendant lequel le client a utilisé le capteur.

J'ai également ajouté les champs firstName, LastName, dateOfBirth et sex à la table User. Voici le schéma de base de données.

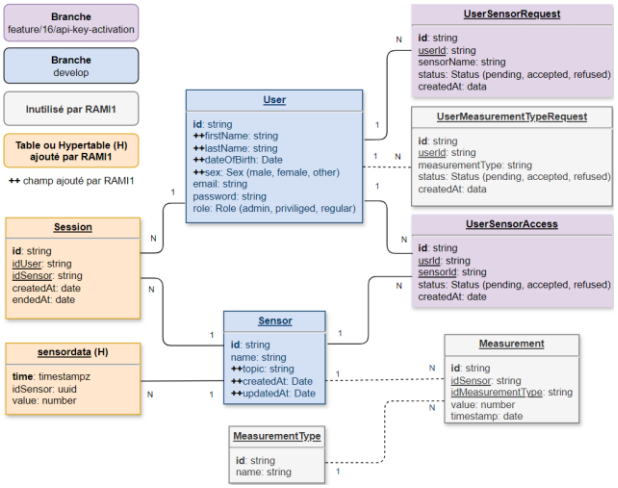


Figure 24 - Schéma de la base de données de RAMI1

On constate que plusieurs tables ne sont plus utilisées. En effet, la table Measurement définissait les opérations CRUD pour recevoir et interagir avec les données d'un capteur via HTTP. Le passage de HTTP à MQTT rend cette partie obsolète.

RAMI1 n'utilise pas MeasurementType et UserMeasurementTypeRequest pour des raisons précisées en annexe 4.

Enfin, les modèles appartenant à une autre branche ont été intégrés en fusionnant la branche sur laquelle j'ai travaillé, afin de pouvoir interagir avec ces modèles et leurs fonctionnalités (controllers …).

Voici le schéma adapté de la structure telle qu'elle est actuellement, prenant en compte les principaux changements dans la base de données résultant du passage de HTTP à MQTT.

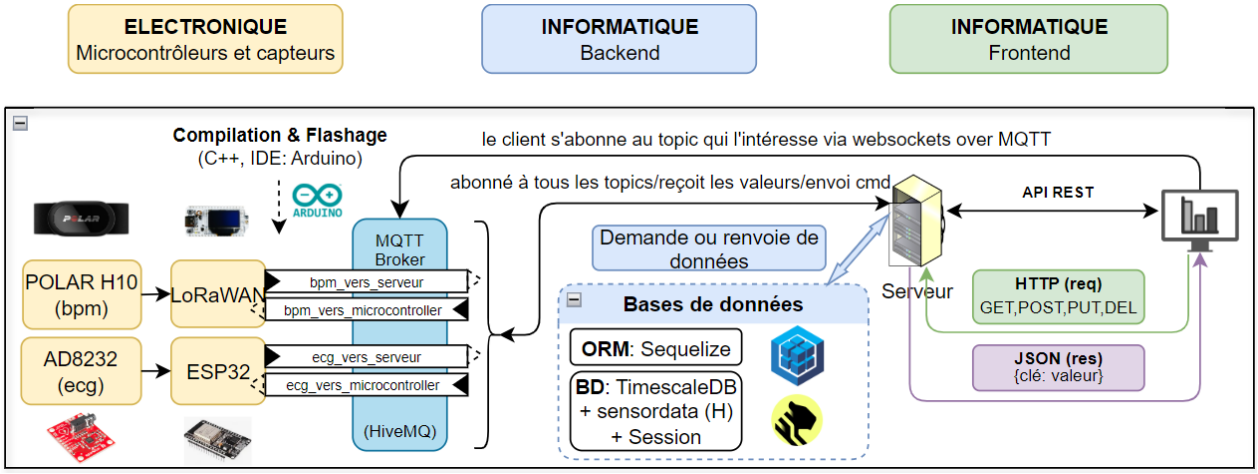


Figure 25 - Structure actuelle de RAMI

3.3. Partie frontend

3.3.1. Les composants Web

En frontend, on utilise le framework Vue.js avec Typescript. Ce dernier adopte le paradigme des composants web. À l'origine, le frontend d'une application web était exclusivement rédigé en HTML et CSS. Ce sont des langages purement déclaratifs, c'est-à-dire que l'on déclare le résultat sans préciser comment y parvenir. Par ailleurs, les développeurs étaient limités aux balises HTML préexistantes comme <p>, <h1>, <ul>, etc.

Avec l'arrivée de JavaScript, la logique a été introduite dans les applications web, donnant naissance aux composants web. Ces composants encapsulent le code HTML/CSS dans une structure réutilisable. En plus, JavaScript permet de définir des traitements logiques au sein de ces composants, rendant leur réutilisation encore plus flexible pour les développeurs.

3.3.2. Étapes de développement front end

Voici les étapes de développement front end pour un composant:

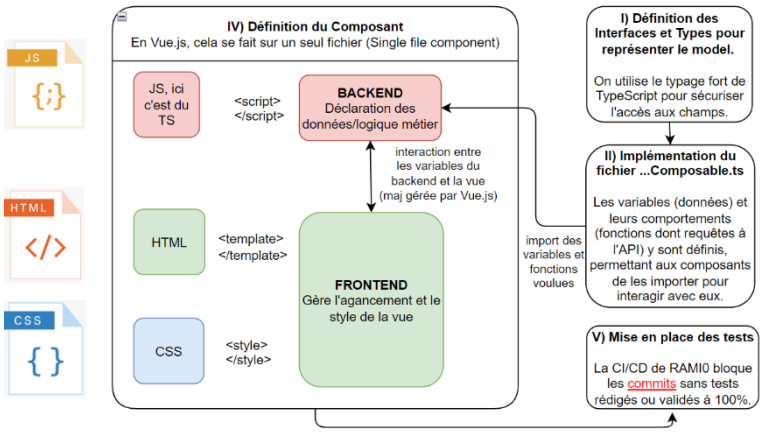


Figure 26 - Étapes de développement frontend

On peut considérer qu'un **composant** web est constitué d'une partie front, celle vue par l'utilisateur, et d'une partie back, qui gère les données.

Un **composable**, quant à lui, est un morceau de code réutilisable qui peut être intégré dans plusieurs composants pour partager des fonctionnalités communes. Ces composables permettent de centraliser la logique et de simplifier la maintenance, en rendant service à plusieurs composants tout en réduisant la redondance.

3.3.3. Mise en place d'une structure favorisant la réutilisabilité

Etant donné que Vue.js suit le pattern des composants web axé sur la réutilisabilité, j'ai dû en tenir compte dans la structure que j'ai proposée. Avec l'ajout des modèles « sensordata » et « session », il était essentiel de concevoir une structure réutilisable. J'ai d'abord réfléchi à mes besoins :

Je voulais afficher la liste des sessions en arrivant sur la page d'accueil (dashboard) ou lors d'un clique sur un utilisateur ou sur un capteur (si on possède les droits).

Aussi, lorsqu'un utilisateur clique sur l'une des sessions, il accède au graphique correspondant. Ce graphique doit afficher les valeurs récupérées en base de données une fois la session terminée.

Le second mode permet au graphique de recevoir les valeurs en temps réel. Voici une représentation graphique des composants, en vert, la vue et en rouge, la partie backend du frontend.

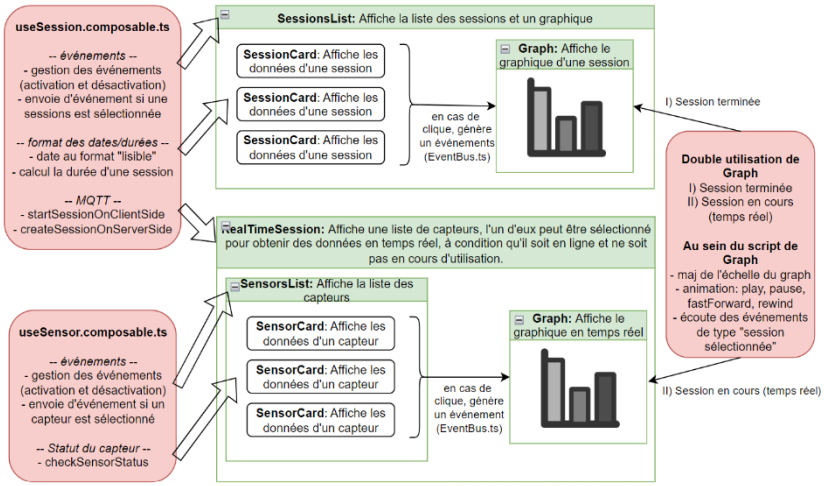


Figure 27 - Représentation graphique des composants et leurs interactions avec leurs composables/script (non exhaustif)

Voici la GUI, les données sont fictives et générées par des seeders (voir annexe 6) sauf pour les sessions temps réel (figure 31):

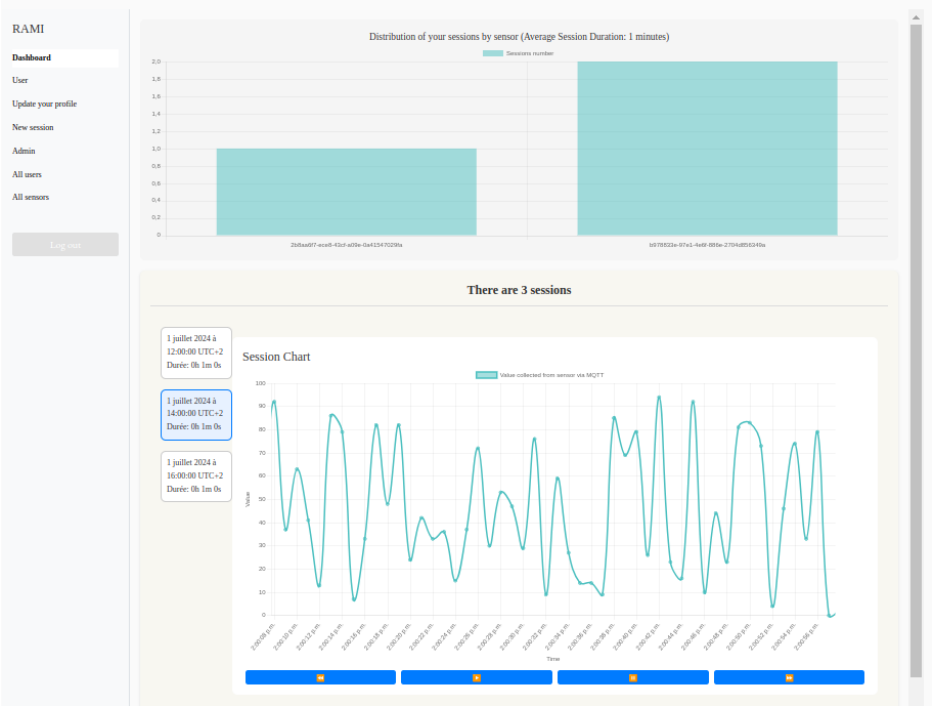


Figure 28 - GUI: Utilisation du composant « SessionsList » au sein de la page « Dashboard » (accueil)

Figure 29 - GUI: Utilisation du composant « SessionsList » au sein de la page « All users »

Figure 30 - GUI: Utilisation du composant « SessionsList » au sein de la page « All sensors »

Ce composant est donc réutilisé dans 3 vues différentes.

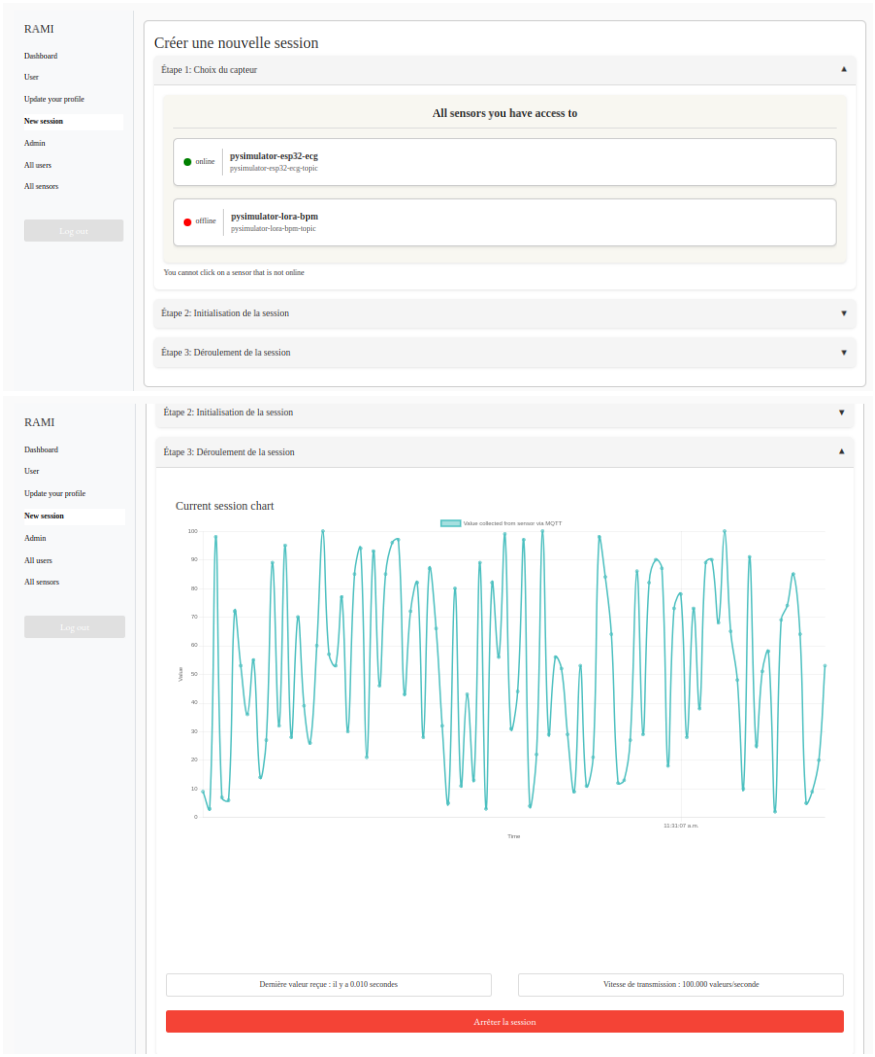


Figure 31 - GUI: Utilisation du composant « Graph » au sein de la page « New session»

Les sections « User » et « Admin » ont été présentées plus tôt dans ce rapport (voir figure 9). La partie « Update your profile » est disponible en annexe 5.

Voici maintenant les explications concernant le fonctionnement d'une session. Sur la figure 27, on y voit les fonctions « startSessionOnClientSide » et « createSessionOnServerSide » dans la partie « useSession.composable.ts puis -- MQTT -- » . Après avoir sélectionné un capteur en ligne à l'étape 1 et cliqué sur "commencer" à l'étape 2, la session est ouverte uniquement côté client. Donc, l'utilisateur est abonné au topic du capteur et reçoit ses valeurs sur son graphique temps réel. Les informations telles que la date de début de session et l'identifiant de l'utilisateur sont recueillies. Ce n'est qu'après que l'utilisateur clique sur "arrêter la session" que la fonction createSessionOnServerSide est utilisée. En résumé, une session est considérée comme valide et donc écrite en base de données lorsque l'utilisateur l'a démarrée **et** terminée.

Enfin, bien que les captures aient été réalisées avec des capteurs simulés, j'ai également testé l'application avec les vrais capteurs et cela a également fonctionné.

4. Les apports personnels du stage

4.1. Analyse des missions réalisées

Premièrement, rappelons que le modèle conceptualisé de RAMI, paru en 2022, était une initiative du service ILIA (Informatique, Logiciel et Intelligence Artificielle). L'architecture proposée par le service fait intervenir l'ensemble de ses compétences, puisqu'il s'agit d'un logiciel interagissant avec le monde de l'IoT et qui à terme, souhaite utiliser ces données médicales pour entraîner ses IA. C'est donc l'objectif final de RAMI.

Outre les améliorations apportées à la plateforme initial, mon stage a permis la transition de l'utilisation de HTTP à MQTT, un protocole largement utilisé dans le monde de l'IoT. En plus de converger vers le modèle conceptualisé, RAMI1 prend désormais en charge :

- La communication en temps réel entre le microcontrôleur et le serveur via un broker MQTT.
- L'abonnement du client, côté navigateur, à un topic via MQTT over WebSockets.
- La capacité à stocker et accéder aux données massives envoyées par les microcontrôleurs grâce à l'utilisation de TimescaleDB, une base de données en séries temporelles.
- L'interaction du client avec les capteurs qu'il peut utiliser, avec les sessions réalisées selon les niveaux d'accessibilité définies par RAMI0.

On propose ainsi une interaction fiable, en temps réel et sécurisée. En résumé, c'est une avancée supplémentaire vers le modèle conceptualisé.

4.2. Analyse du travail

- Connaissances utilisées :

Lors de ce stage, j'ai travaillé seul sur les parties électronique et informatique du projet. Cette expérience m'a permis de mettre en pratique de nombreuses compétences acquises au cours de mon cursus.

En ce qui concerne la partie **électronique**, j'ai rédigé du code en C++, en tenant compte des contraintes matérielles des microcontrôleurs, qui sont bien plus limitées que celles d'un ordinateur classique. La matière « Architecture des ordinateurs » m'a été utile, notamment parce qu'elle m'a rappelé que les passages par copie consomment plus de registres. Enfin, MQTT étant un protocole nouveau pour moi, c'est probablement la compétence que j'ai le plus développée au cours de ce stage. Il en va de même pour l'utilisation de l'IDE Arduino et la manipulation des microcontrôleurs comme l'ESP32. Étant des domaines liés à l'électronique, j'avais naturellement moins de compétences dans ces domaines au départ.

Côté **backend**, bien que je n'aie pas entièrement saisi la matière « Architecture orientée services » en cours, ce stage m'a permis de tout comprendre en la mettant en pratique. J'ai pu appliquer ces connaissances pour travailler avec TypeScript et mettre en place un backend complet. La matière « Sécurité » a également été précieuse, car elle m'a sensibilisé aux choix de sécurité à adopter et aux bonnes pratiques, comme le hachage des mots de passe, par exemple.

Les cours de bases de données m'ont également été très utiles pour manipuler PostgreSQL et TimescaleDB. De plus, j'ai appliqué de nombreux concepts de programmation orientée objet, notamment dans le simulateur et la classe MqttServer ainsi les cours correspondants m'ont été précieux.

Côté **frontend**, j'ai pu tirer parti de ma connaissance des composants web et d'Angular, acquise lors du projet optionnel de spécialité ET4, pour mieux appréhender Vue.js.

Conclusion

Au cours de mon stage, j'ai eu l'opportunité de contribuer à l'évolution du projet RAMi, une architecture temps réel dédiée au suivi médical à distance des personnes âgées en utilisant l'Internet des Objets Médicaux (IoMT). Mon travail a principalement consisté à aligner RAMi0, la première version développée en 2023, avec le concept original de RAMi.

Plusieurs missions ont été réalisées pour atteindre cet objectif. D'une part, j'ai effectué des choix technologiques déterminants concernant la communication des données, avec la transition de HTTP vers MQTT. Il s'agit un protocole largement utilisé dans l'IoT, garantissant une communication plus efficace et fiable. J'ai ensuite implémenté un simulateur pour vérifier que la nature de cette transition respectait les piliers et contraintes de fiabilité, de temps réel et de sécurité.

D'autre part, j'ai redéfini le format des messages envoyés par les microcontrôleurs afin qu'ils se conforment au format des séries temporelles. Après m'être assuré que les microcontrôleurs communiquaient correctement avec le serveur simulé, j'ai implémenté la classe MqttServer dans le backend pour permettre au serveur de gérer l'intégralité de la communication via MQTT. S'en est suivi l'adaptation du schéma de la base de données pour une meilleure gestion et stockage des données en séries temporelles grâce à TimescaleDB. De plus, j'ai amélioré le frontend en intégrant un graphique en temps réel, offrant ainsi aux utilisateurs une visualisation immédiate des données reçues.

Ainsi, les principaux résultats obtenus sont la transition réussie de RAMi0 vers un système plus aligné avec le concept original de RAMi. Ce système permet la communication en temps réel des couples capteur-microcontrôleur avec le serveur via MQTT et avec l'interface utilisateur. Désormais RAMi intègre pleinement les capacités de l'IoMT tout en respectant les contraintes mentionnées précédemment.

Ce début de convergence vers le modèle conceptualisé ouvre de nombreuses perspectives pour l'avenir de RAMi. Une évolution naturelle consisterait à poursuivre cette convergence en intégrant des technologies d'intelligence artificielle pour mettre en place le suivi personnalisé des patients. Pour ce faire, une première étape serait d'utiliser Apache Kafka pour gérer la file d'attente des messages, assurant ainsi une communication fluide et efficace entre les topics MQTT et les modules d'analyse. Ensuite, l'IA pourrait être dockerisée et déployée dans Apache Spark, permettant un traitement des données à grande échelle et en temps réel. Tout cela dans le respect du modèle conceptualisé (voir figure 5).

Mais avant tout cela, il sera essentiel de commencer par corriger le problème de bruit sur l'ESP32 et rédiger un cahier des charges. Ce document devra impérativement constituer la ligne directrice de RAMi et anticiper les besoins de la plateforme avec le plus de recul possible. Il est crucial de concevoir le système dans sa globalité et en gardant à l'esprit sa finalité.

Les annexes 2, 3, 4 et 8 soulèvent certains points à clarifier. Il sera également important de commencer à réfléchir davantage à la sécurité des données et à la gestion des identités numériques au sens des normes belges et européennes. Plus nous anticipons ces aspects, plus le code sera clair et cela nous permettra d'éviter des situations complexes, tels que les changements d'approche après implémentation.

Enfin, RAMi doit également être envisagé dans sa globalité. Dounia Messaoudi, ancienne élève de Polytech Paris-Saclay, a réalisé son stage ET4 à Polytech Mons en 2022. Le sujet de son stage « Interface applicative pour le traitement et l'analyse de données IoT » concernait l'implémentation d'une application mobile multiplateforme pour RAMi. Il sera également nécessaire de statuer sur l'avenir de RAMi en tenant compte de l'ensemble de ces plateformes.

Il reste donc du chemin à parcourir, mais je suis convaincu que le projet est sur la bonne voie. J'ai appris bien plus que ce que j'espérais et je suis honoré d'avoir pu contribuer à un tel projet.

Glossaire

ILIA	Informatique, Logiciel et IA
Thales LAS France	Thales Land and Air Systems
RAMI	Real-Time Internet of Medical Things Architecture for Elderly Patient Monitoring
RAMI0	Version de RAMI reçue avant mon stage (code des capteurs, du backend et du frontend)
RAMI1	Version de RAMI telle qu'elle après mon stage
IA	Intelligence Artificielle
IoT (Internet of Things)	Interconnexion via Internet d'objets physiques dotés de capteurs et de logiciels leur permettant de collecter et échanger des données.
IoMT (Internet of Medical Things)	L'Internet des Objets Médicaux (IoMT) est une sous-branche spécialisée de l'Internet des Objets (IoT). L'IoMT permet de collecter, transmettre et analyser en temps réel les données de santé des patients, facilitant ainsi le diagnostic, le traitement, et le suivi personnalisé.
Logiciels open source	Logiciels dont le code source est librement accessible, modifiable, et redistribuable par quiconque.
Backend	La partie d'une application qui gère la logique, le traitement des données, les interactions avec la base de données, fonctionnant côté serveur et restant invisible pour l'utilisateur final.
Frontend	La partie visible d'une application avec laquelle l'utilisateur interagit directement, incluant l'interface utilisateur et les éléments graphiques
Microcontrôleur	Petit ordinateur intégré sur une puce, capable d'exécuter des tâches spécifiques, souvent utilisé dans des dispositifs embarqués.
Série temporelle	Ensemble de données collectées à différents points dans le temps, souvent utilisées pour analyser des tendances ou des patterns.
Base de données en séries temporelle	Base de données optimisée pour stocker et interroger efficacement des séries temporelles
MQTT	Protocole de messagerie léger conçu pour les communications machine-à-machine (M2M) et IoT via l'utilisation de topic et du système d'abonné.
Broker MQTT	Serveur qui gère la distribution des messages entre les clients dans un réseau MQTT, facilitant la communication entre les dispositifs.
Microservices	Architecture logicielle où une application est composée de petits services indépendants, communiquant souvent via des APIs.
GitLab	Plateforme DevOps qui permet de gérer le cycle de vie du développement logiciel : contrôle de version, intégration continue et le déploiement.
Intégration continue (CI/CD)	Pratique de développement qui automatise les tests, la validation et le déploiement du code pour garantir la qualité et accélérer les livraisons.
Conteneurs Docker	Environnements légers et portables qui encapsulent une application et ses dépendances, facilitant son déploiement cohérent sur différentes machines.
API	Interface permettant à des applications de communiquer entre elles via des requêtes et des réponses standardisées.
Raspberry Pi	Ordinateur monocarte compact
HTTP	Protocole de communication basé sur les requêtes - response
ESP32	Microcontrôleur Wi-Fi et Bluetooth intégré
LoRaWAN32	Microcontrôleur basé sur la technologie LoRaWAN, utilisé pour des communications longue portée à faible consommation d'énergie
ECG AD8232	Capteur de bioélectricité conçu pour surveiller l'activité cardiaque en capturant des signaux ECG (électrocardiogramme
BPM POLAR H10	Moniteur de fréquence cardiaque portable, utilisé pour mesurer les battements par minute (BPM)
Websockets over MQTT	Technologie combinant WebSockets et MQTT pour permettre une communication bidirectionnelle en temps réel entre un serveur et un client web.
Composants web	Éléments réutilisables d'une interface utilisateur, encapsulant HTML, CSS, et JavaScript pour une utilisation modulaire dans les applications web.

Bibliographie

[1] : DEBAUCHE Olivier, NKAMLA PENKA Jean Bertin, MAHMOUDI Saïd, LESSAGE Xavier, HANI Moad, MANNEBACK Pierre, KANKU LUFULUABU Uriel, BERT Nicolas, MESSAOUDI Dounia, GUTTADAURIA Adriano, « RAMi: A New Real-Time Internet of Medical Things Architecture for Elderly Patient Monitoring, Information », (2022), p.10-14 consulté en mai 2024 à l'adresse https://www.mdpi.com/2078-2489/13/9/423

[2] : Wikipédia : « Régions linguistiques de Belgique » (date non précisée), page 1 : consulté en juin 2024 https://fr.wikipedia.org/wiki/R%C3%A9gions_linguistiques_de_Belgique

[3] : Wallonne Bruxelles Campus : « Les universités francophones de Belgique » (date non précisée) : consulté en juin 2024 https://www.studyinbelgium.be/fr/les-universites-francophones-de-belgique

[4] : Polytech Mons : « De l'Ecole des Mines à la Faculté Polytechnique de Mons » (date non précisée), consulté en juin 2024 https://web.umons.ac.be/fpms/fr/a-propos-de-la-faculte/historique/

[5] : UMONS : « Quelques dates » (date non précisée), consulté en juin 2024 https://web.umons.ac.be/fr/universite/lumons-en-bref/quelques-dates/

[6] : UMONS : « Présentation de l'Université de Mons » (date non précisée), consulté en juin 2024 https://web.umons.ac.be/fr/universite/lumons-en-bref/presentation-de-lumons/

[7] : UMONS : « Le Développement Durable » (date non précisée), consulté en juin 2024 https://web.umons.ac.be/fr/universite/universite-responsable/le-developpement-durable/

[8] : Infortech : « Institut de recherche en Technologies de l'Information et Sciences de l'Informatique de l'UMONS » (date non précisée), consulté en juin 2024 https://web.umons.ac.be/infortech/fr/

[9] : Lilian SOLER : « Création de microservices pour l'architecture RAMi », (2023), lecture complète du rapport de stage, consulté en avril 2024.

[10] : GeeksforGeeks : « Différence entre les protocoles MQTT et HTTP » (2021), consulté en avril 2024 https://www.geeksforgeeks.org/difference-between-mqtt-and-http-protocols/

[11] : Dreams of Code : « Solving one of PostgreSQL's biggest weaknesses » (2023), consulté en mai 2024 https://www.youtube.com/watch?v=ruUlK6zRwS8

Annexes

Annexe 1 : Explication de la pertinence du simulateur

Le simulateur a permis de résoudre plusieurs problématiques. Lors de l'implémentation du code sur les capteurs, chaque modification nécessitait une compilation et un flashage, ce qui était chronophage. De plus, il était difficile de tester efficacement le code. Si j'avais choisi de passer directement de HTTP à MQTT au niveau des capteurs, cela aurait aussi impliqué de modifier toute la partie backend, notamment en ajoutant la connexion à un broker, avant même de pouvoir vérifier si le code des capteurs fonctionnait correctement. C'est pourquoi j'ai décidé de mettre en place ce simulateur afin d'effectuer une transition progressive :

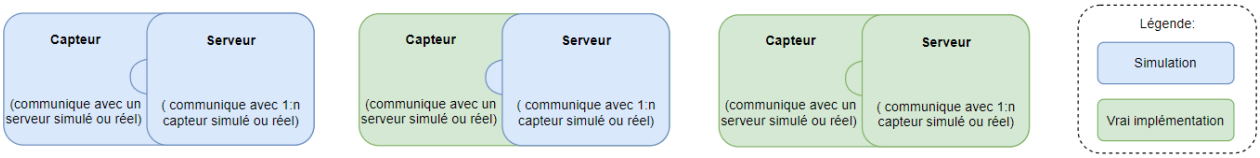


Figure 32 - De gauche à droite, étapes successives d'implémentation suivie lors de mon stage

À chaque étape, j'ai vérifié que la communication se déroulait correctement au sein de RAMI, en m'assurant que chaque modification fonctionnait comme prévu.

Les futurs développeurs de RAMI pourront utiliser le simulateur de différentes manières :

- Simplification des étapes de développement :

Électronique : Si un contributeur spécialisé en électronique souhaite travailler sur le code des capteurs, il pourra utiliser le simulateur pour simuler un serveur, plutôt que de démarrer tout le backend et le frontend de RAMI1 pour vérifier si les capteurs communiquent correctement avec le serveur.

Informatique : Les développeurs backend et frontend pourront simuler les capteurs sans la contrainte d'avoir un microcontrôleur connecté à leur PC ou un capteur sur eux (électrode, ceinture cardiaque).

Bien sûr, le simulateur ne remplace pas les vrais microcontrôleurs et le véritable serveur, mais il simplifie le développement côté électronique et informatique.

Pour que cet outil reste efficace, il est essentiel de le maintenir à jour. Je propose donc que les futurs développeurs suivent cette pratique

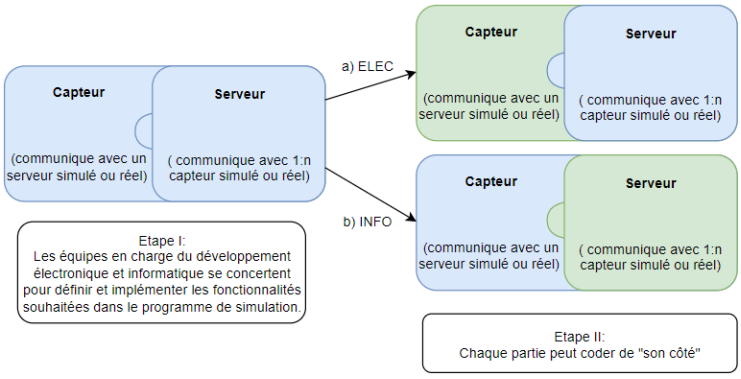


Figure 33 - Proposition de processus de développement

- Simulation d'un grand nombre de capteurs lors du déploiement de la plateforme (il faudra tout de même réfléchir à quelque modifications pour faciliter l'utilisateur du simulateur pour ce cas).
- Réalisation de benchmarks du broker MQTT en utilisant le graphique de ResultReporter. Plus généralement, on peut évaluer le comportement du broker lors de scénarios spécifiques, comme l'envoi d'un grand volume de données (un ecg dure 5 à 10 minutes et envoi 100 valeurs/sec).

Annexe 2 : Démonstration d'utilisation de *SpecificConstants.hpp* et *MQTTCommonOperations.hpp* par le code principal de l'ESP32. (voir cette page et la page suivante)

SpecificConstants.hpp

esp32-mqtt > SpecificConstants.hpp

```hpp
#ifndef SPECIFIC_CONSTANTS
#define SPECIFIC_CONSTANTS

#if !defined(PROGMEM)
#define PROGMEM
#endif

/******
 * Usage of PROGMEM
 * ----------------
 * PROGMEM is used to store variables in flash memory (program memory) instead of SRAM.
 * This is particularly useful on microcontrollers with limited SRAM.
 * To decide whether a string should be stored in PROGMEM, we consider its length and its access frequency.
 * - If the string is long and accessed rarely, it should be stored in PROGMEM.
 * - Else, it should remain in SRAM for performance reasons.
 ******/

// ---------------------------------- PART SPECIFIC TO A MICROCONTROLLER ----------------------------------------------
// For instance, the wifi password may be different according to where the microcontroller is going to be used.

/****** WiFi Connection Details and root certificate *******/
// PROGMEM beacause these settings are used only once at the beginning
extern const char* SSID PROGMEM;
extern const char* PASSWORD PROGMEM;
extern const char *ROOT_CA PROGMEM;// may be common

/****** MQTT Broker Settings; PROGREM because same as wifi, except for the topics constants which we access very often *******/
extern const char* MQTT_BROKER PROGMEM;    // may be common
extern const char* MQTT_USERNAME PROGMEM;  // may be common
extern const char* MQTT_PASSWORD PROGMEM;  // may be common
extern const int MQTT_PORT;                // may be common
extern const char* MQTT_TOPIC;
extern const char* MQTT_TOPIC_TO_SPEAK_ON;
extern const char* MQTT_TOPIC_TO_LISTEN_ON;

/****** Settings; PROGREM would be useless here *******/
extern const unsigned int NUMBER_OF_VALUES_PER_SECOND;
extern const long INTERVAL;
extern unsigned long previousMillis;
extern bool allow_to_publish;

#endif // SPECIFIC_CONSTANTS
```

MQTTCommonOperations.hpp

all-microcontrollers > MQTTCommonOperations.hpp

```hpp
#ifndef MQTT_COMMON_OPERATIONS
#define MQTT_COMMON_OPERATIONS

#include <WiFi.h>
#include <NTPClient.h>
#include <PubSubClient.h>
#include <ArduinoJson.h>
#include <WiFiClientSecure.h>

/******
 * Usage of PROGMEM
 * ----------------
 * PROGMEM is used to store variables in flash memory (program memory) instead of SRAM.
 * This is particularly useful on microcontrollers with limited SRAM.
 * To decide whether a string should be stored in PROGMEM, we consider its length and its access frequency.
 * - If the string is long and accessed rarely, it should be stored in PROGMEM.
 * - Else, it should remain in SRAM for performance reasons.
 ******/

// ---------------------------------- PART COMMON TO ALL MICROCONTROLLERS ----------------------------------------------
// For example, the different microcontrollers are all capable of understanding the same commands and launching the same tyoe of a
// That's because they all speak the same language (if this change, update this part...)

/****** Commands; NO PROGREM HERE, because we want the microcontroller to respond quickly to commands
(those are frequently compared to what we received from the servor)*******/
extern const char* COMMAND_PING;
extern const char* COMMAND_START;
extern const char* COMMAND_STOP;
// possible answers
extern const char* PING_RESPONSE;
extern const char* PING_RESPONSE_WHEN_ALREADY_PUBLISHING;
extern const char* START_RESPONSE;
extern const char* STOP_RESPONSE;
// type of the message
extern const char* MSG_TIMESTAMP;
extern const char* MSG_CMD;
extern const char* MSG_ANS;
extern const char* MSG_VALUE;

/****** NTP Client Settings; PROGREM because these settings are configured only once at the beginning *******/
extern const char* NTP_SERVER PROGMEM;
extern const long GMT_OFFSET_SEC;
extern const int DAYLIGHT_OFFSET_SEC;

/*********************************** Function prototypes ************************************/
// Wifi and security
void setup_wifi(const char* ssid, const char* password);
void setCACertForTLS(WiFiClientSecure& client, const char* certificate);
// Mqtt (connexion, command reception and message publication)
void reconnect(PubSubClient& client, const char* mqtt_username, const char* mqtt_password, const char* topic);
void publishJSONMessage(PubSubClient& client, const char* topic, const char* json_buffer, const bool& retained=true);
long long getCurrentMicrosecondTimestampLong();
void publishAnswerToServerCommand(PubSubClient& client, const char* topic, const String& answer, const bool& retained=true);
void publishValue(PubSubClient& client, const char* topic, const float& value, const bool& retained=true);
void handlePingCommand(PubSubClient& client, const char* topic, const bool& allow_to_publish);
void handleStartCommand(PubSubClient& client, const char* topic, bool& allow_to_publish);
void handleStopCommand(PubSubClient& client, const char* topic, bool& allow_to_publish);
void interactWithReceivedCommand(PubSubClient& client, const String& received_command, const char* topic, bool& allow_to_publish);
```

rami1_esp32_AD8232_ecg.ino

esp32-mqtt > rami1_esp32_AD8232_ecg > rami1_esp32_AD8232_ecg.ino

```ino
#include "SpecificConstants.hpp"
#include "MQTTCommonOperations.hpp"

/**** esp32 sensor pin Settings *******/
const int pin_lo_minus = 2;
const int pin_lo_plus = 15;
const int sdn = 18;
const int analog_pin = 34;

/**** Secure WiFi Connectivity Initialisation *****/
WiFiClientSecure espClient;

/**** MQTT Client Initialisation Using WiFi Connection *****/
PubSubClient client(espClient);

/**** FUNCTIONS *****/
/***** Callback Method, allow the sensor to react when Receiving MQTT messages ****/
void callback(char *topic, byte *payload, unsigned int length) {
    Serial.print("Message arrived in topic: ");
    Serial.println(topic);

    String received_message;
    for (int i = 0; i < length; i++) {
        Serial.print((char) payload[i]);
        received_message += (char)payload[i];
    }
    Serial.println();

    // Parse the received JSON message
    DynamicJsonDocument doc(1024);
    DeserializationError error = deserializeJson(doc, received_message);
    if (error) {
        Serial.print("deserializeJson() failed: ");
        Serial.println(error.c_str());
        return;
    }

    String received_command = doc[MSG_CMD];
    interactWithReceivedCommand(client, received_command, MQTT_TOPIC_TO_SPEAK_ON, allow_to_publish);
}
```

```ino
/**** MAIN PROGRAM *****/

void setup() {
    // Set software serial baud to 115200;
    Serial.begin(9600);
    pinMode(pin_lo_plus, INPUT); // Setup for leads off detection LO +
    pinMode(pin_lo_minus, INPUT); // Setup for leads off detection LO -
    pinMode(sdn ,OUTPUT);
    pinMode(analog_pin, INPUT);
    digitalWrite(sdn, HIGH);

    setup_wifi(SSID, PASSWORD);
    setCACertForTLS(espClient, ROOT_CA);      // enable this line and the the "certificate" code for secure connection

    configTime(GMT_OFFSET_SEC, DAYLIGHT_OFFSET_SEC, NTP_SERVER); // For timestamp

    // Connecting to mqtt broker
    client.setServer(MQTT_BROKER, MQTT_PORT);
    client.setCallback(callback); // how to answer to mqtt messages
}

/******** Main Function ************/
void loop() {
    if (!client.connected()){ // check if client is still connected
        reconnect(client, MQTT_USERNAME, MQTT_PASSWORD, MQTT_TOPIC_TO_LISTEN_ON);
    }
    client.loop(); // Loop as long as we are not connected

    if((digitalRead(pin_lo_plus) == 1)||(digitalRead(pin_lo_minus) == 1)){ // Look for any connection problems
        Serial.println('!');
    }
    else {
        // send the value of analog
        unsigned long currentMillis = millis();
        if (currentMillis - previousMillis >= INTERVAL) {
            previousMillis = currentMillis;

            if (allow_to_publish) {
                float ecgValue = analogRead(analog_pin); // Read the analog value
                publishValue(client, MQTT_TOPIC_TO_SPEAK_ON, ecgValue, true);
            }
        }
    }
}
```

Figure 34 - code des fichiers SpecificConstants.hpp, MQTTCommonOperations.hpp et rami1_esp32_AD8232_ecg.ino

Chaque microcontrôleur, à travers son programme principal, effectue un « include » pour importer le fichier des opérations MQTT, qui définit les opérations communes à tous les microcontrôleurs, ainsi que le fichier des constantes, spécifique à chaque microcontrôleur.

J'aurais pu déclarer le prototype de la fonction callback dans le .hpp des opérations MQTT et l'implémenter dans le fichier .cpp correspondant. Cependant, j'ai choisi de la laisser dans le fichier principal pour souligner qu'il faudra prendre une décision **sur son avenir**.

Actuellement, j'ai supposé que chaque microcontrôleur devait réagir de la même manière aux commandes du serveur. Mais comment gérer une commande qui, à l'avenir, pourrait être spécifique à un type de **capteur** en particulier ? Dans ce cas, il est préférable de s'assurer que seul le microcontrôleur associé au capteur concerné soit capable de réagir à cette commande…

setup : Exécutée une seule fois au démarrage du microcontrôleur, cette fonction initialise les variables, configure les broches et établit les connexions nécessaires avant le début du programme principal.

loop : Exécutée en boucle après setup, cette fonction répète continuellement le code, surveillant les entrées, contrôlant les sorties, et gérant les événements en temps réel.

Par ailleurs, je pense qu'il serait pertinent d'améliorer la communication entre le serveur et le capteur en remplaçant les chaînes de caractères telles que « start », « stop » ou « ping » par des codes hexadécimaux. L'utilisation de l'hexadécimal pour représenter les commandes, par exemple 0xA pour « start » et 0xB pour « stop », offre plusieurs avantages. Cela permet de réduire la taille des messages et d'améliorer l'efficacité du traitement. De plus, cette méthode permet de définir jusqu'à 16 commandes différentes avec une seule place hexadécimale, tout en offrant la possibilité d'extension si nécessaire (ajout d'une place). Dans ce cas, soit n le nombre de place, le nombre de commandes possibles est 16^n.

Cependant, avant de mettre en œuvre cette modification, il est essentiel de mener une analyse approfondie des besoins futurs de la plateforme pour déterminer le nombre exact de commandes nécessaires. En parallèle, il est également important de renforcer la gestion des erreurs du côté électronique pour garantir la robustesse du système (et donc de définir les codes d'erreur correspondants).

Annexe 3 : Comment choisir le nombre de topics ?

Comme vous l'avez lu, pour la communication entre le serveur et le couple microcontrôleur/capteur, j'utilise ce que j'ai appelé les « topics distincts ».

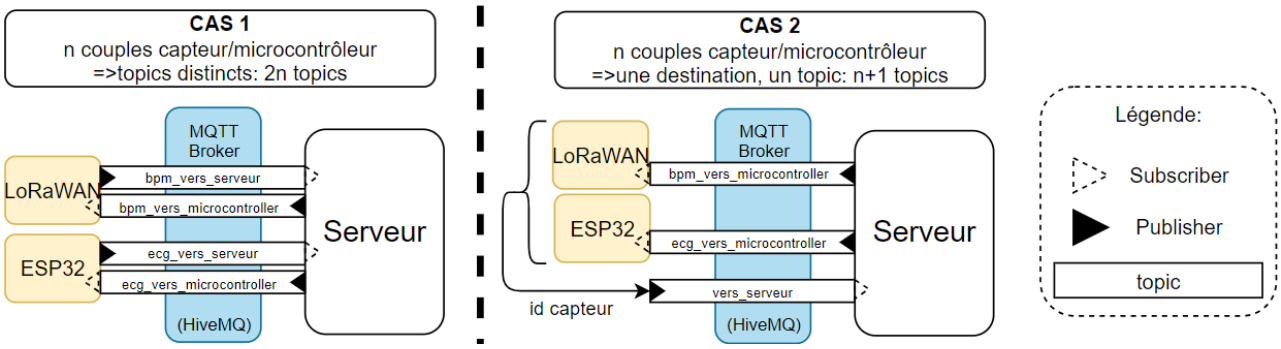


Figure 35 - Comparaison des politiques possibles pour le choix du nombre de topics

Nous sommes donc dans le cas 1. Mais si chaque capteur envoie ses données au serveur, pourquoi ne pas utiliser un topic dédié aux communications vers le serveur ? C'est le cas 2.

Je n'ai pas choisi cette solution pour éviter de créer un « single point of failure ». Imaginez que n utilisateurs utilisent chacun un capteur en même temps, et que chaque capteur envoie 100 valeurs/sec au serveur. Cela reviendrait à faire circuler 100n données/sec sur un seul topic. Si ce topic tombe en panne, toutes les communications échoueraient !

De plus, contrairement au cas 1 où il y a autant de topics que de couples microcontrôleur/capteur, dans le cas 2, il n'est pas possible de savoir directement quel capteur envoie les données. Il est donc nécessaire d'inclure l'identifiant du capteur dans la série temporelle transmise au serveur. En général, pour n capteurs, si le nombre de topics est **strictement** inférieur à 2n, l'identifiant du capteur doit être inclus dans la série temporelle. Pour éviter cela et le risque de « single point of failure », j'ai utilisé la solution du cas1. En effet, les brokers sont capables de gérer des milliers, voire des millions de topics sans problème.

Il est important de noter que ce choix ne dépend pas des politiques de répartition de charge définies par les brokers. Je souhaitais que la solution reste indépendante des technologies de répartition utilisées.

Quoiqu'il en soit, il faudra également configurer le broker pour gérer les autorisations d'accès, les droits de publication et de souscription des clients, en s'assurant que seuls les dispositifs autorisés peuvent publier ou s'abonner à un topic spécifique. (Impossible de le faire sur la version gratuite du broker HiveMQ que j'ai utilisé).

Aussi, il est nécessaire de revoir la manière dont les topics sont nommés. Actuellement, les topics sont nommés selon le schéma : nom_capteur + « -topic ». Une réflexion sur l'amélioration de ce schéma de nommage est nécessaire. Exemple /nom_capteur/nom_mesure/id_sensor.

Annexe 4 : Avenir des modèles MeasurementType et UserMeasurementTypeRequest

Comme mentionnée dans la partie 3.2.3, RAMI1 n'utilise pas ces modèles. UserMeasurementTypeRequest reste tout de même accessible par l'utilisateur coté frontend. Actuellement, RAMI1 ne les utilise pas car l'ajout d'une métadonnée d'unité aux séries temporelles les rendrait superflues…

Tous ces points et bien d'autres doivent être examinés, en tenant compte des besoins spécifiques de RAMI. Il est absolument essentiel de définir un cahier des charges qui prenne en compte ces considérations.

Annexe 5 : Mise à jour des vues de création de compte et définition d'une vue pour mise à jour des informations de l'utilisateur.

Comme j'ai ajouté les champs firstName, LastName, dateOfBirth, sex, j'ai mis à jour la vue de création de compte et crée celle concernant la mise à jour des champs. Ces deux vues utilisent le fichier FormBuilder.ts que j'ai rédigé pour construire plus facilement leur formulaires respectifs.

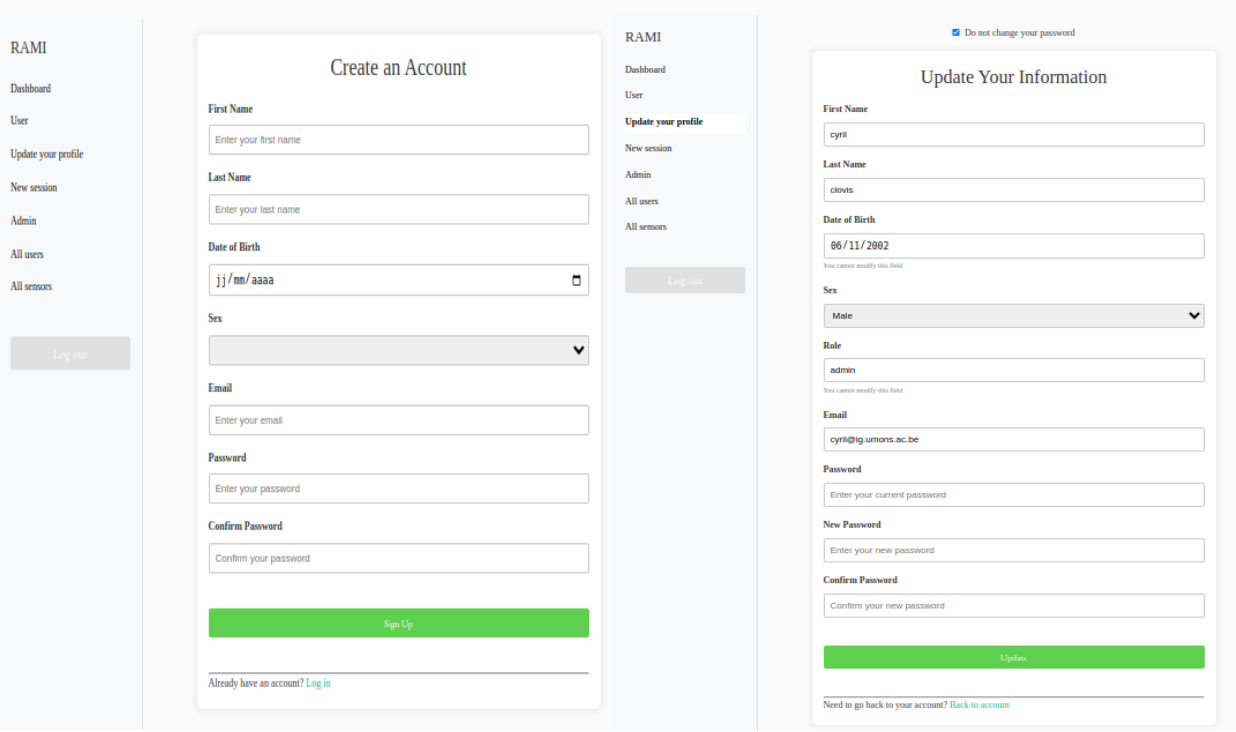


Figure 36 - Vues concernant la création de compte et la mise à jour des informations de l'utilisateur

Annexe 6 : Détail sur les seeders, c'est-à-dire les données, ici fictives, qui peuple la bases de données

J'ai configuré les seeders pour simuler l'ensemble de la situation suivante : être administrateur, faire la demande de création pour deux capteurs, « pysimulator-lora-bpm » et « pysimulator-esp32-ecg » et les acceptés. Ensuite, j'ai accordé l'accès à ces capteurs à d'autres utilisateurs.

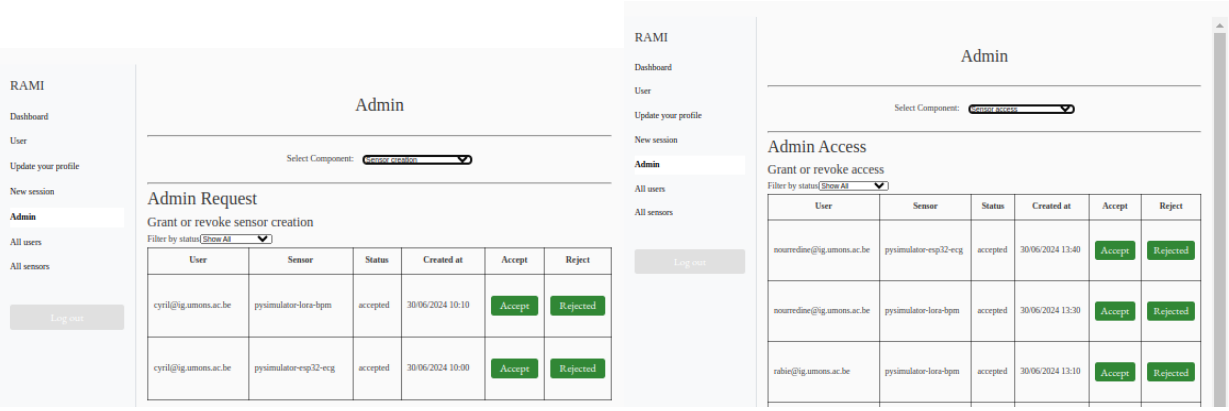


Figure 37 – Vues administrateur montrant les demandes de création et d'accès aux capteurs

Annexe 7 : Quelques détails sur la classe MqttServer

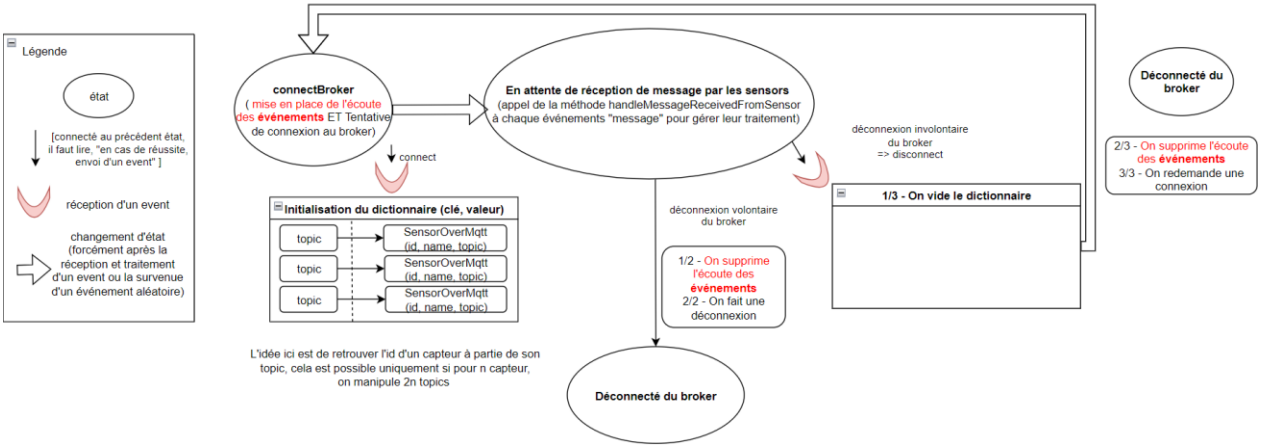


Figure 38 - Fonctionnement de la gestion des événements de la classe MqttServer sous forme d'automate fini

Annexe 8 : Autres propositions d'amélioration

- Electronique
- Voir annexes 2 et 3

- Backend
- Voir annexes 3 et 4
- Ajout d'un modèle « évaluation » pour permettre l'évaluation d'une session.
- Ajout d'un modèle « problème » permettant à un utilisateur de signaler un problème lors de l'utilisation du couple microcontrôleur-capteur.

- Frontend
- Ajout de critères de recherche par nom, âge, etc.
- Intégration d'un système de messagerie pour aider les utilisateurs à manipuler les capteurs. (On pourrait utiliser MQTT pour une communication en temps réel ou mettre en place un chatbot).