Système de tracking COVID-19 pour les cours à l'UniFR

Une architecture pour la gestion et l'organisation des données de tracking dans un cadre universitaire

Travail de Bachelor

Bastian Stadelmann

Septembre 2021

Supervisé par :

Prof. Dr. Jacques Pasquier-Rocha
Software Engineering Group

UNI FR
UNIVERSITÉ DE FRIBOURG
UNIVERSITÄT FREIBURG

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

Software Engineering Group

Remerciements

Merci à Luna pour sa patience et son grand soutien durant l'écriture de ce travail
Merci à Daniel pour ses conseils et ses bonnes idées
Merci à mes parents et à ma famille pour leur aide précieuse

Finalement, merci à M. Pasquier et à M. Gremaud pour leur supervision et leur grande efficacité pendant la réalisation de ce projet, ainsi que la confiance qu'ils m'ont accordée.

Table des matières

Liste des figures

Liste des codes source

1
Introduction

1.1. But du projet

La pandémie de COVID-19 a profondément affecté la vie de chacun d'entre nous durant ces deux dernières années. La volonté de retracer les chaînes d'infections, afin de limiter leur propagation, constitue la raison pour laquelle les restaurants ont adopté différents systèmes de tracking. Les désormais familiers QR codes disposés sur les tables permettent donc de signaler sa présence au sein d'un établissement et d'au besoin pouvoir être averti d'un éventuel risque encouru.

Lors d'une discussion au sujet du (fort attendu) retour à la normale, je me suis rendu compte qu'un tel système n'existait pas au sein de l'Université. Le but de ce travail est donc de remédier à ce constat.

1.2. Objectifs

Création d'une architecture qui permet de saisir et de stocker les données des professeurs et des étudiants, ainsi que les informations relatant leurs présences aux cours de l'Université.

1.3. Organisation du travail

Chapitre 1 : Introduction

Suite aux buts et objectifs de ce travail énoncés ci-dessus, l'introduction en présente également la structure, ainsi qu'un bref résumé de chaque chapitre.

Chapitre 2 : Analyse des besoins

Dans ce chapitre, les exigences de la solution sont énumérées, ainsi que les étapes prévues pour les satisfaire.

Chapitre 3 : Réalisation

Dans ce chapitre, l'application et son utilisation sont présentées, puis le processus de sa création. Les méthodes et langues de programmation choisies, ainsi que les difficultés rencontrées pendant sa création sont ensuite passées en revue.

Chapitre 4 : Analyse du résultat

Dans ce chapitre, le résultat final est analysé. La solution est évaluée principalement par rapport à son utilité en situation réelle, ainsi que son ergonomie.

Chapitre 5 : Expansion

Dans ce chapitre, les possibilités d'expansion du travail sont mises en évidence. La solution est comparée à d'autres systèmes du même type et les fonctionnalités supplémentaires qui pourraient bénéficier au projet sont proposées.

Chapitre 6 : Conclusion

Pour terminer, ce chapitre synthétise les différents éléments abordés et met en valeur les bénéfices du présent travail pour mon avenir professionnel.

2
Analyse des besoins

2.1. Plan initial

Avant de commencer à concevoir les différentes parties de mon système, il a évidemment fallu créer un plan pour avoir une bonne idée initiale de ses composants, de leurs fonctions et de leurs interactions.

La première étape de ce projet fut d'établir une structure claire pour les tables de la base de données. En effet, avoir une vue d'ensemble des différents éléments et de leur agencement permettait d'éviter certaines erreurs pouvant engendrer de grandes pertes de temps par la suite. L'idée initiale pour mes données reposait sur les principes suivants : premièrement, je voulais stocker des présences d'élèves à leurs cours respectifs. Ceci implique logiquement de capturer leurs **informations personnelles**, dont un moyen de contact en cas de besoin, dans ce cas leur adresse e-mail universitaire.
Deuxièmement, il me fallait une manière de relier les étudiants à un **cours** pour définir s'ils y avaient été exposés à des risques (et par qui). Finalement, une entité retenant le lien entre un étudiant et un cours, une **présence**, était nécessaire. Ceci est réalisé avec une relation plusieurs-à-plusieurs, utilisant une table de liaison.

Ce sont donc là les trois modèles présents dans ma base de données (fig. 2.1).

Dans un second temps, le choix des différents composants m'a occupé. La structure de données étant établie, il était maintenant nécessaire de se concentrer sur les sources et les destinations de ces données.

Il m'a toujours paru évident qu'un tel projet nécessitait un site internet. Je me suis décidé à faire fonctionner ce site ainsi que la base de données depuis un point centralisé. Ainsi, le premier élément de mon infrastructure était en place : Un serveur, écrit en Python, avec une base de données rattachée à celui-ci.

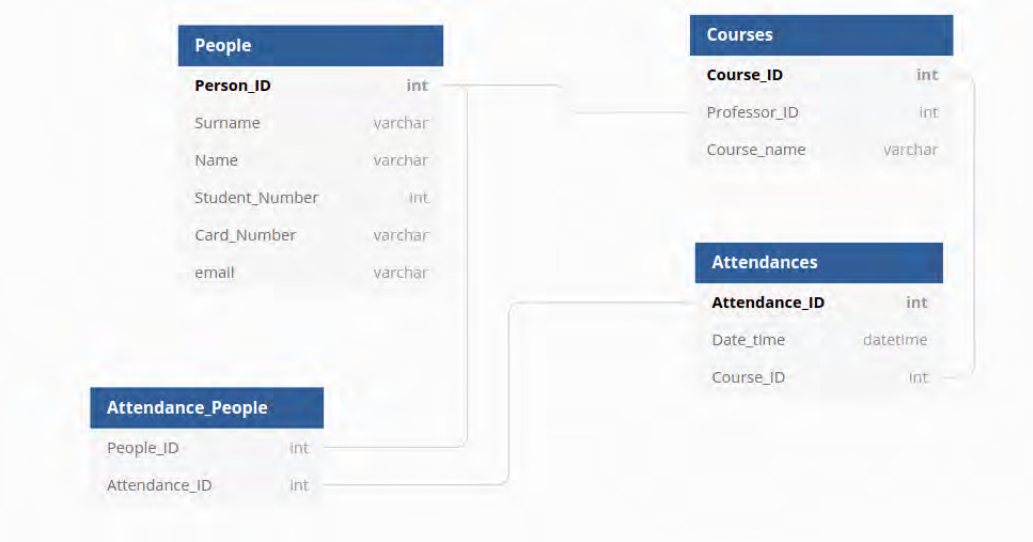

Figure 2.1. – Schéma initial de la base de données [1]

Ce serveur serait donc la pièce centrale du projet, étant émetteur ou récepteur de toute communication.

Cependant, d'après moi, un site internet n'est pas une manière très confortable de noter ses présences en cours. Il fallut donc créer d'autres éléments qui rendraient le processus plus agréable. Un système de code QR, à l'image de beaucoup d'autres solutions de tracking semblables, m'est immédiatement venu à l'esprit. L'étudiant peut ainsi scanner le code QR, ce qui permet d'enregistrer sa présence. Cette méthode a l'avantage de ne pas mettre l'étudiant en contact d'un matériel qu'il n'a pas déjà touché auparavant. Il utilise simplement son propre smartphone.

Une troisième méthode pour s'inscrire dans le cours, comme discuté avec mon responsable de travail de Bachelor, serait d'utiliser la carte d'étudiant, plus précisément la puce qui se trouve à l'intérieur. L'idée étant la suivante : l'étudiant vient en cours, passe sa carte sur un lecteur et sa présence est immédiatement enregistrée. Ici, l'utilité de cette méthode est surtout de donner une autre option à l'étudiant. Si son smartphone n'a plus de batterie, il ne peut ni aller sur le site, ni scanner le code QR à l'entrée de la salle. Il est donc utile d'avoir une méthode indépendante se trouvant de toute façon dans la salle de cours où il se rend.

Tous ces scénarios sont repris et détaillés dans la section Use-cases ci-après.

2.2. Use-cases : Étudiant

Nous allons maintenant passer en revue les possibilités proposées aux étudiants souhaitant utiliser le présent système pour signaler leurs présences aux cours. Il s'agit de scénarios

1. Diagramme créé avec dbdiagram.io[1]

théoriques, leurs applications pratiques définitives étant présentées plus bas dans ce travail.

Scénario 1 :

Luna arrive à l'Université au milieu du mois de septembre pour reprendre les cours après les vacances d'été. Par e-mail, elle a été informée par l'Université que les cours en présentiel reprennent et qu'un système de tracking est disponible dans les salles de classe.

Lorsqu'elle entre dans l'auditoire où son cours de neurosciences cognitives se déroule, elle remarque un ordinateur sur une table près de la porte. Les explications afin d'utiliser le système y sont inscrites sur une feuille. Luna décide d'utiliser l'application, parce que c'est l'option la plus pratique pour elle. Elle la télécharge et l'ouvre pour la première fois.

Elle arrive sur l'écran principal où un message l'informe qu'elle doit créer un profil. Elle passe sur la fenêtre de profil et introduit ses données personnelles dans le formulaire (nom, prénom, adresse e-mail, numéro d'étudiant). Elle sauvegarde ses données, qui restent sur son téléphone pour la prochaine utilisation de l'application. Elle passe ensuite sur la fenêtre QR, où elle peut scanner le code à l'entrée. Ses données sont envoyées au serveur pour y être stockées. Une confirmation apparaît pour lui signaler que sa présence au cours a bien été enregistrée.

Scénario 2 :

Jonas arrive le matin pour son cours de statistiques. En entrant dans la salle, connaissant déjà bien le système, il sort son téléphone pour scanner le code QR comme à son habitude. Il se rend cependant compte que son téléphone est déchargé. Il attrape son porte-monnaie, sort sa carte d'étudiant et la passe sur le lecteur de cartes. À l'écran près du lecteur, un message apparaît lui demandant de donner son numéro d'étudiant, qu'il tape sur le clavier. Un message l'informe ensuite que sa carte a été liée à son compte et que sa présence est enregistrée, le remercie et lui souhaite un bon cours.

Scénario 3 :

Oliver, étant prévoyant, a lu les mails concernant le système de tracking et s'est déjà rendu sur le site qui y était mentionné. Sur la page principale, il a lu les explications et a simplement pu cliquer sur un lien qui l'a emmené vers un formulaire pour créer un profil. Ici aussi, il entre ses données personnelles.

Le jour du cours, Oliver arrive à la salle et utilise son smartphone pour retourner sur le site. Il clique sur le lien qui lui propose d'enregistrer une présence. Une fois fait, il inscrit simplement son nom ainsi que celui du cours et les données sont envoyées au serveur. Un message lui confirme que sa présence a été sauvegardée avec succès. Il part s'asseoir et attend le début du cours.

Scénario 4 :

Noémie entre en classe et comme à son habitude, sort sa carte d'étudiant et la passe sur le scanner. Étant déjà enregistrée dans le système, celui-ci lui confirme la présence. Elle peut désormais ranger sa carte et aller s'installer.

Scénario 5 :

Maxime s'apprête à entrer au cours de diététique. N'étant quant à lui pas très prévoyant, il n'a ni lu le mail concernant le système de tracking, ni créé de profil sur le site, ni d'ailleurs chargé son téléphone pendant la nuit.

Il arrive donc avec pour seul moyen sa carte d'étudiant. En la passant sur le scanner, un message apparaît à l'écran lui demandant de donner son numéro d'étudiant pour lier sa carte à son profil.

Après avoir entré le numéro, un autre message l'informe qu'il n'a pas de profil, mais qu'il peut en créer un directement. Il n'a plus qu'à taper les trois informations manquantes, c'est-à-dire son nom, son prénom et son adresse e-mail. Cela étant fait, les informations sont envoyées au serveur et enregistrées. Maxime a désormais un profil complet sur le serveur.

2.3. Use-cases : Professeur

Après avoir exposé les scénarios d'utilisation pour les étudiants, nous allons à présent décrire ceux spécifiques aux professeurs, moins nombreux et relativement simples.

Scénario 1 :

M. Pasquier prépare sa salle pour donner son cours de Génie logiciel aux étudiants. Après avoir accompli ses tâches habituelles, il allume l'ordinateur qui sert à enregistrer les présences et lance le programme dédié à cette tâche.

Le programme lui demande l'ID de son cours, qu'il tape sur le clavier et qu'il confirme en appuyant sur entrée. Le programme initialise le lecteur de cartes et crée le code QR spécifique à son cours pour les utilisateurs de l'application, avant de l'afficher à l'écran.

Tout est prêt pour accueillir les étudiants. Une fois le cours fini, il termine simplement le programme et éteint l'ordinateur.

Scénario 2 :

Mme Rettig, étant professeur du cours de psychologie de l'adolescence, est chargée de mettre en place le nécessaire pour le déroulement du cours, y compris le système de tracking.

Elle se rend sur le site internet et y enregistre son profil. Ensuite, elle clique sur le lien pour enregistrer un nouveau cours. Elle doit simplement taper son nom et s'enregistrer en tant que professeur.

Une fois cela accompli elle peut, comme dans le scénario précédent, allumer la borne et donner l'ID du cours.

3
Réalisation

3.1. Moyens techniques

Dans ce chapitre, nous allons aborder les moyens techniques, c'est-à-dire les langues de programmation et l'équipement, qui ont été utilisés pour pouvoir réaliser les objectifs présentés dans le chapitre précédent.

3.1.1. Aiohttp

Aiohttp[2] est un module pour Python qui remplit la fonction de serveur et de manière plus générale, fournit beaucoup de méthodes et de classes rendant la communication par internet plus simple et plus intuitive.

Dans le cadre de ce travail, il était non seulement important d'avoir un serveur central qui gère les communications avec les autres éléments, mais aussi de permettre l'accès au site internet facilement et clairement. C'est dans un contexte comme celui-ci qu'aiohttp brille : ce module permet à l'utilisateur de mettre en place des "routes" (des chemins d'accès vers une page ou une ressource spécifique) et de les lier à des fonctions qui géreront les données à recevoir ou envoyer pour chacune d'entre elles.

```
1 import views
2
3 def setup_routes(app):
4     app.router.add_get('/', views.index)
```

```
    app.router.add_get('/people', views.people)
    app.router.add_get('/people/add', views.add_person)
    app.router.add_get('/courses', views.courses)
    app.router.add_get('/courses/add', views.add_course)
    app.router.add_get('/attendances', views.attendances)
    app.router.add_get('/attendances/add', views.add_attendance)
    app.router.add_get('/signal', views.signal)
    app.router.add_get('/contact', views.contact)
    app.router.add_get('/stats', views.stats)
    app.router.add_post('/qrcodeinterface', views.save_by_QR)
    app.router.add_post('/proxmarkinterface', views.proxmark_interface)
    app.router.add_post('/addPersonData', views.save_person)
    app.router.add_post('/addCourseData', views.save_course)
    app.router.add_post('/addAttendanceData', views.save_attendance)
    app.router.add_static('/css', "HTML_templates/css")
    app.router.add_static('/images', "HTML_templates/images")
```

Listing 3.1 – Les routes du serveur

On voit ici que les routes sont ajoutées une par une, en utilisant la méthode appropriée à son utilisation ('get' ou 'post', qui représentent les méthodes HTTP[3] et finalement 'static', pour les fichiers statiques qui doivent être accessibles par d'autres parties du site).

On définit ensuite le chemin relatif auquel la page doit être disponible et ensuite la fonction qui est responsable de gérer les accès à cette page, qu'on appelle une *view*.

Ces views sont généralement des fonctions très courtes, renvoyant à l'utilisateur une page HTML, ou faisant un court appel à la base de données pour en renvoyer les informations. Dans notre cas, on fait aussi appel à des *handlers*, qui vont gérer réellement les informations de manière plus complexe.

```
async def trySaveNewCourse(data : dict):
    # Validate the dict
    try :
        assert(data['pi'] != "")
        assert(data['cn'] != "" and len(data['cn']) > 2)
    except AssertionError :
        print("Invalid course data. Course was not saved.")
        return
    except :
        print("Other error during saving of course data. Course was not saved.")
        return

    professorID = data['pi']

    await Courses.get_or_create(Professor_ID=professorID, Course_name=data['cn'])
```

Listing 3.2 – Handler pour sauvegarder un cours

Pour finir, aiohttp a l'avantage important d'être programmé de manière asynchrone, c'est à dire que ses fonctions peuvent être exécutées sans bloquer le reste du processus. Pour une librairie qui pourrait, dans le cas d'un système de tracking pour les cours universitaires, gérer des milliers d'accès par seconde, c'est un avantage primordial.

3.1.2. TortoiseORM

Pour aborder ce que fait TortoiseORM[4] et comment il fonctionne, il faut d'abord expliquer le concept d'un ORM.

Un ORM (ou "Object–relational mapping") est une couche intermédiaire entre le code de haut niveau, orienté objet que l'on peut avoir dans un projet de programmation, et une base de données relationelle.

Par défaut, un langage de programmation tel que Python ou Java et une base de données n'ont pas du tout la même représentation interne d'un même élément. En Python, on sera tenté de faire un objet avec des propriétés internes. En revanche, dans un modèle relationnel comme c'est le cas en SQL par exemple, on représente les données en tant qu'entrées dans des tableaux, qui sont liées symboliquement par des références (des "foreign key").

Un ORM permet donc de traduire une approche orientée objet pour une base de données relationnelle. Il permet, depuis Python, de trouver des données et de les recevoir directement sous forme d'objet.

Pour en revenir à TortoiseORM, il y a plusieurs raisons motivant le choix de ce module en particulier : il a une bonne performance pour les interactions avec la DB, il est lui aussi asynchrone et permet de définir les objets que l'on voudra utiliser plus tard.

```python
from tortoise.models import Model
from tortoise import fields

# Model classes for the DB

class People(Model):
    Person_ID = fields.IntField(pk=True)
    Surname = fields.TextField()
    Name = fields.TextField()
    Student_Number = fields.IntField(unique=True)
    Card_Number = fields.TextField(default="-1")
    Email = fields.TextField()
    Attended: fields.ManyToManyRelation["Attendance"] = fields.ManyToManyField(
        "models.Attendance",
        related_name="attendants",
        through="Attendance_People",
        description="Attendants of the course"
    )

    def __str__(self):
        return f"{self.Surname} {self.Name} : {self.Person_ID}"
```

Listing 3.3 – Definition de l'objet 'People' avec TortoiseORM

Comme on le voit, chaque champ est défini avec son nom et un type. D'autres paramètres sont utilisables, tels que *pk* (qui définit le champ en question comme clé primaire), *default* (qui définit une valeur par défaut si aucune autre n'est donnée) ou encore *unique* (qui détermine qu'aucun autre objet de ce type ne peut avoir la même entrée dans ce champ).

Il est aussi possible avec TortoiseORM d'établir des relations plusieurs-à-plusieurs, en déterminant les tables de liaison.

3.1.3. Proxmark3

Afin de pouvoir scanner les cartes des étudiants, un lecteur de cartes m'a été fourni par le superviseur de ce travail. Il s'agit d'un Proxmark 3 RDV4 (fig. 3.1), qui est à la base un outil de test RFID pour la cybersécurité. Il est utilisé par des groupes de *pentesting* (ou test d'intrusion en français) pour découvrir et documenter des points faibles dans les algorithmes de chiffrage de cartes de crédits, de cartes d'accès ou autres.

Figure 3.1. – Proxmark3 RDV4 utilisé pour ce projet

Dans notre cas, cet outil sera utilisé pour pouvoir lire les informations présentes dans la puce de la carte d'étudiant et lier le numéro interne de la carte au numéro d'étudiant qui est inscrit dessus. Si cela avait été possible, j'aurais préféré pouvoir trouver le numéro d'étudiant directement à partir des données lues sur la carte, mais elles sont chiffrées et les déchiffrer dépasserait largement le cadre de ce travail.

Afin de pouvoir utiliser le lecteur de cartes, il faut installer un outil en ligne de commande (fig. 3.2). Cet outil, conçu par le RfidResearchGroup, permet d'interagir avec le lecteur de cartes pour lui dire quel genre de carte on souhaite lire et ce qu'il faut faire avec la carte en question[5].

Pour notre utilisation, le numéro 'MSN' (**M**anufacturer **S**erial **N**umber) de la carte suffira. Ce numéro est une sorte de numéro de série, il est par définition unique sur chaque carte, ce qui est très important pour pouvoir l'assigner à une carte spécifique.

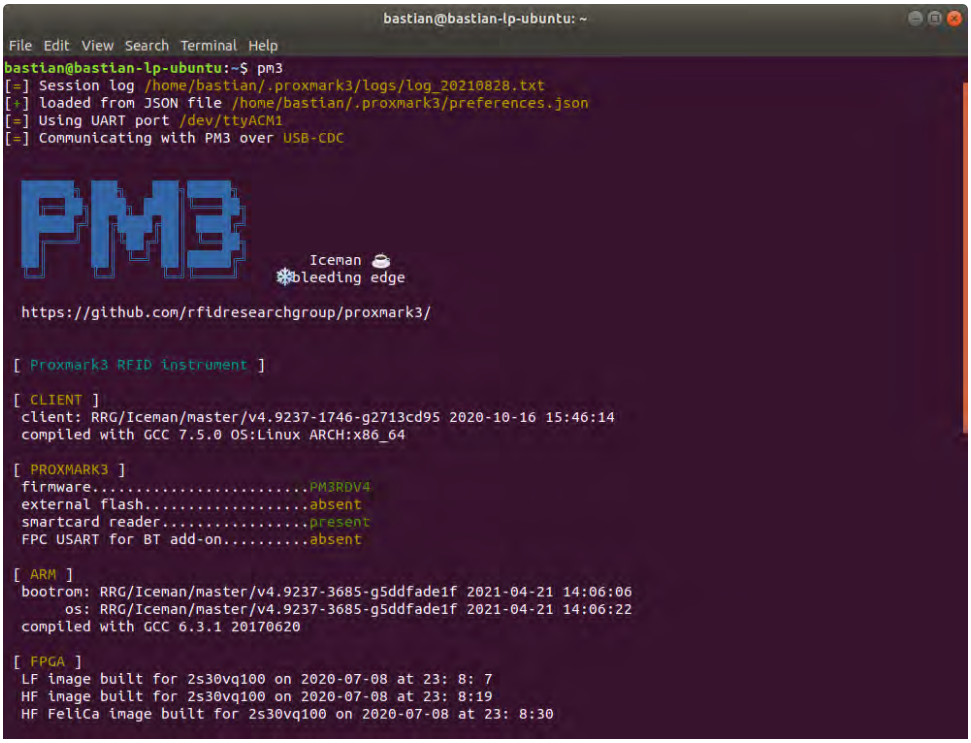

Figure 3.2. – Outil de recherche *pm3*

Ainsi, on lance cet outil régulièrement depuis un client séparé qui s'occupe de la lecture de cartes, en gérant les différents cas possibles, comme présentés dans les Use-cases (chapitre 2.2)

3.1.4. Flutter

La dernière étape importante de réalisation pour ce projet est la réalisation d'une application mobile. Il a donc fallu chercher des langages de programmation adaptés à cette tâche. Après quelques recherches, une possibilité attrayante s'est révélée : Flutter[6].

Flutter n'est pas, à proprement parler, une langue de programmation mais plutôt un framework spécialisé pour la conception d'applications mobiles, basé sur la langue de programmation Dart[7]. Le principe maître de Flutter est de concevoir son application comme un arbre. Plusieurs éléments de base (appelés *Widgets*) sont disponibles (boîte de texte, bouton, gestionnaire d'onglets, etc) et il faut les imbriquer les uns dans les autres afin d'obtenir une structure fonctionnelle.

```
1  class QRFormState extends State<QRForm> {
2
3    @override
4    Widget build(BuildContext context) {
5      return MaterialApp(
6          home: Scaffold(body: Builder(builder: (BuildContext context) {
7        return Container(
8            alignment: Alignment.center,
9            child: Flex(
10             direction: Axis.vertical,
```

```
          mainAxisAlignment: MainAxisAlignment.center,
          children: <Widget>[
            ElevatedButton(
                onPressed: () => scanQR(), child: Text('Start QR scan')),
            Text("$infoText", style: new TextStyle(
              fontSize: 20
            ),),
          ],
        ));
    }));
  }
```

Listing 3.4 – Constructeur de l'onglet de lecture pour le code QR

Au-delà de son aspect particulier, le code écrit avec Flutter est assez lisible et permet surtout de mettre en place une application très rapidement, en utilisant peu de lignes de code.

3.2. Éléments de programmation

Dans ce chapitre sont présentés quelques éléments de programmation du projet qui sont intéressants ou importants.

Validation
Comme dans tout projet informatique utilisant des données venant d'humains, il est impératif de valider ces données avant de les utiliser. Dans notre cas, il a fallu choisir des règles de validation très larges et faciles à implémenter dans plusieurs langues de programmation (Dart pour l'application mobile, Python pour le serveur et la borne avec le lecteur de cartes, JavaScript/HTML pour le site).
Un exemple de validation très basique est déjà visible dans le Listing 3.2 (au niveau du serveur), en voici un exemple pour la borne.

```
def inputEmail(): # email must be at least five chars long, use ascii chars only and
    contain an @ and a .
    returnValue = input("Please type your full email and press enter:")
    while(len(returnValue) < 5 or not returnValue.isascii() or "@" not in returnValue
        or "." not in returnValue):
        returnValue = input("This email is not valid. Please try again:")

    return returnValue
```

Listing 3.5 – Entrée d'adresse mail par boucle de validation en Python

Une boucle comme celle-ci permet d'obliger l'utilisateur à entrer des données valides sans casser le flux du programme.

Objets Flutter
Flutter permet de mettre en place des objets rapidement et efficacement. Il s'agit là d'une première expérience avec un tel outil de programmation. La manière dont les éléments de l'application sont conçus, ainsi que la façon de leur donner les propriétés voulues, est plutôt intéressante.

```
new TextFormField(
    onSaved: (value) {
        model.lastName = value!;
    },
    controller: lastNameController,
    keyboardType: TextInputType.name,
    decoration: new InputDecoration(
        hintText: 'Your last name',
        labelText: 'Last Name'
    ),
    validator: (value) {
        if(value!.isEmpty) {
            return 'Please enter your last name.';
        }
        return null;
    })
```

Listing 3.6 – Création d'une boîte de texte dans Flutter

On peut voir qu'il est possible de définir beaucoup de détails, de la validation à la décoration, ainsi que des centaines d'autres réglages sont disponibles pour ce genre d'objet. Flutter permet de personnaliser très fortement chaque élément mis à disposition par le framework.

3.3. Réalisation des Use-cases

Afin de réaliser les Use-cases, il a fallu visualiser à un niveau plus technique comment chacune des composantes en question (site web, application, lecteur de cartes) fonctionne.

Ainsi, les diagrammes UML suivants ont été conçus afin de servir de référence lors de leur programmation.

Site web

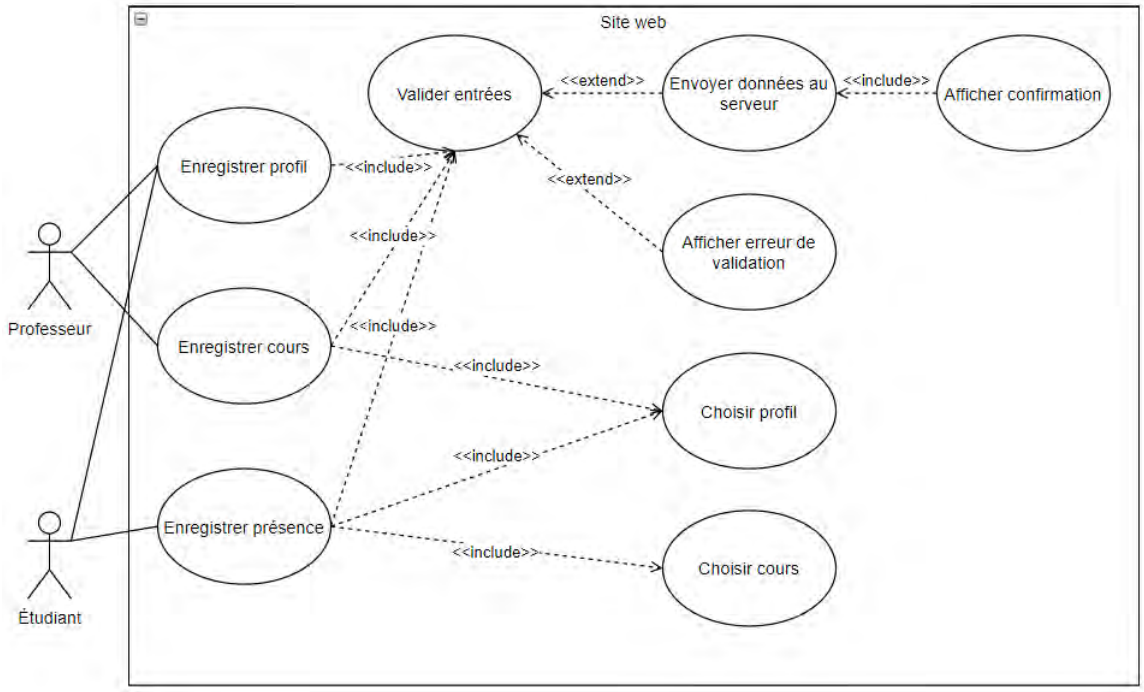

Figure 3.3. – Diagramme UML : Site web

Application

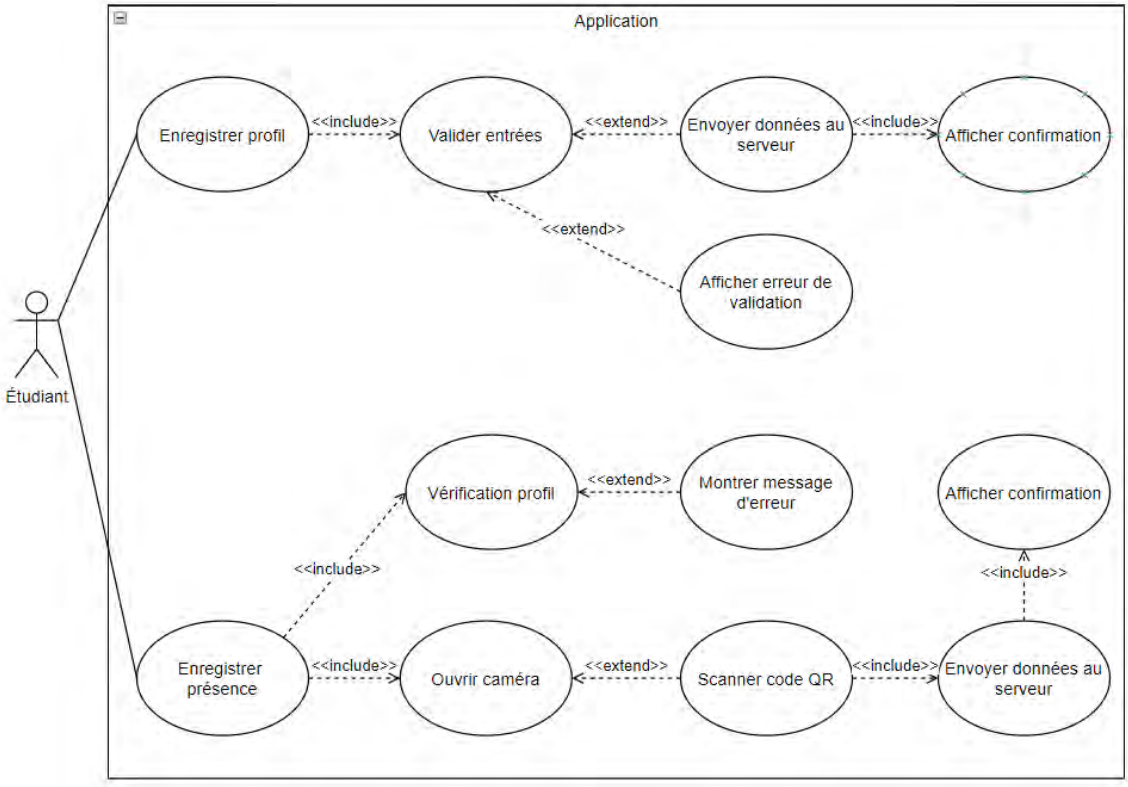

Figure 3.4. – Diagramme UML : Application

Lecteur de cartes

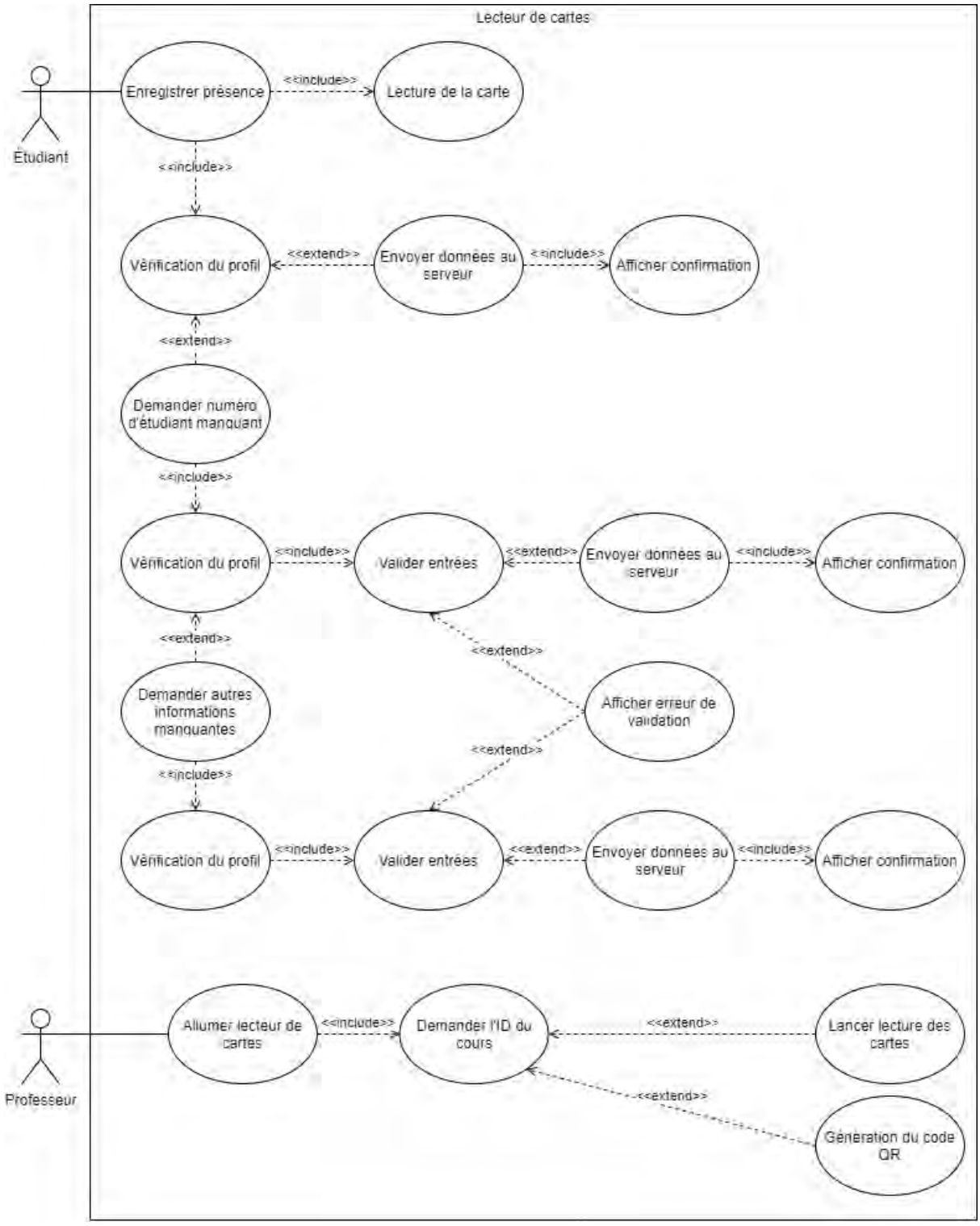

Figure 3.5. – Diagramme UML : Lecteur de cartes

3.4. Difficultés rencontrées

Nous allons maintenant voir quelques-unes des difficultés rencontrées pendant le développement de la solution.

Base de données
Comme déjà discuté dans la section 2.1, une base de données est nécessaire au bon fonctionnement de ce projet et sa structure est cruciale pour l'organisation du reste du code.

En mettant en place les entités utilisées (People, Course, Attendance), une relation plusieurs-à-plusieurs était nécessaire entre les personnes et les cours. Bien que cette opération soit réalisable sur TortoiseORM, la documentation concernant de tels cas est minime. Pour ne rien arranger, cet ORM propose plusieurs manières de faire très différentes pour toute opération souhaitée, ce qui n'a pas rendu facile la recherche d'exemples sur internet.
La fonctionnalité a fini par marcher, mais ce fut bien plus laborieux qu'espéré.

Application mobile
Dans son essence, l'application mobile est plutôt simple. Il s'agit de deux vues différentes, l'une d'entre elle servant à sauvegarder un profil (avec validation, stockage permanent des données), l'autre à scanner le code QR et contenant la logique pour l'envoi des données au serveur.

Cependant, plusieurs problèmes ont été rencontrés lors de sa création. Tout d'abord, le stockage permanent des données de l'utilisateur ne fut pas aussi simple que prévu. En effet, les applications n'ont pas le droit, par défaut, de stocker des données localement. Pour cela, il faut en activer les permissions, ce qui peut vite s'avérer plus complexe que prévu étant donné que chaque type de smartphone fonctionne différemment et que les fichiers de configuration prévus à cet effet sont plutôt difficiles à trouver.
Ensuite, Flutter ne propose pas de service de stockage local, il faut donc trouver un package capable de le faire. Tous ceux que j'ai pu trouver pour accomplir cette tâche étaient des modules fonctionnant de manière asynchrone, ce qui complique beaucoup la tâche. En effet, ayant peu d'expérience avec la programmation asynchrone et n'en ayant aucune en Flutter, ce fut une épreuve de taille.

Plus tard, il a aussi été très difficile de trouver un module pour scanner des codes QR facilement, sans devoir en implémenter la moitié soi-même. Dans le principe, cela m'aurait intéressé, mais une telle entreprise aurait presque mérité son propre travail de Bachelor.

Proxmark3
Le lecteur de carte fourni par mes superviseurs fonctionne très bien et remplit parfaitement sa fonction pour ce projet. Cependant, le programme auquel j'ai recouru pour le faire fonctionner n'est pas adapté à cette utilisation.
En testant le lecteur de cartes pendant l'implémentation, il est vite devenu apparent que ce programme, destiné à la recherche pour la cybersécurité, est très lourd et n'accepte pas vraiment d'interface avec d'autres programmes.

Il a donc fallu programmer en Python une boucle qui lance le programme dans le bon mode pour scanner les cartes de l'Université (des cartes Legic Prime), cela toutes les secondes. Il est nécessaire d'attendre, car le programme prend du temps pour se lancer et communiquer avec le lecteur de cartes et cela fournit une seule lecture. Avec un délai d'une seconde, on arrive à vérifier assez souvent si une carte est présente, sans pour autant avoir deux instances du programme fonctionnant en même temps (cela fait bloquer les nouvelles instances et ne fonctionne donc plus du tout).
Cette interface devrait donc, pour des raisons de performance principalement, être révisée.

4
Analyse du résultat

Dans ce chapitre, nous allons analyser le résultat, discuter des Use-cases et de leur bonne réalisation, ainsi que tenter de déterminer si la solution est agréable à utiliser et si elle est utile en situation réelle.

4.1. Aboutissement des Use-cases

Cette section est consacrée aux Use-cases définis dans les chapitres 2.2 et 2.3. Ainsi, il sera plus facile de se faire une idée de leur réalisation.

4.1.1. Étudiant

Scénario 1 :
Dans le premier scénario, il s'agit essentiellement de l'utilisation de l'application mobile dans son ensemble.
Tout d'abord, Luna, notre étudiante, ouvre l'application et entre ses données de profil (fig. 4.1), elle appuie sur "Save my profile" et obtient une confirmation (fig. 4.2).
Elle passe ensuite sur la page de lecture QR (fig. 4.3) et appuie sur le bouton qui allume la caméra. Luna scanne le code devant elle et reçoit une confirmation (fig. 4.4) quelques instants plus tard. Elle peut maintenant ranger son téléphone et aller en cours.

Comme abordé brièvement dans le chapitre 3.2, l'application utilise de la validation pour empêcher la saisie de données incomplètes ou erronées. Un texte d'alerte en rouge apparaît alors dans la boîte de texte contenant l'erreur.

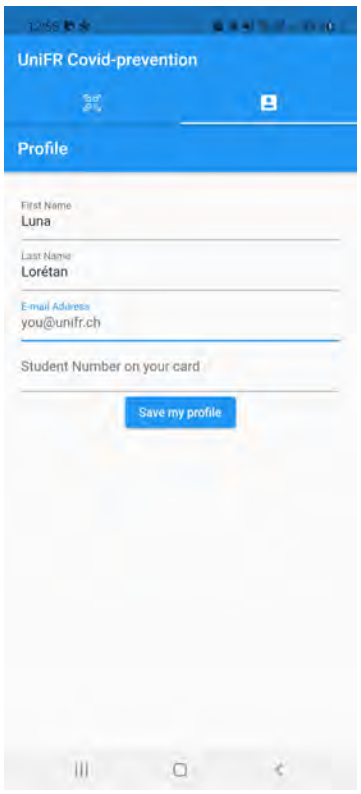

Figure 4.1. – Entrée des données personnelles dans la page de profil

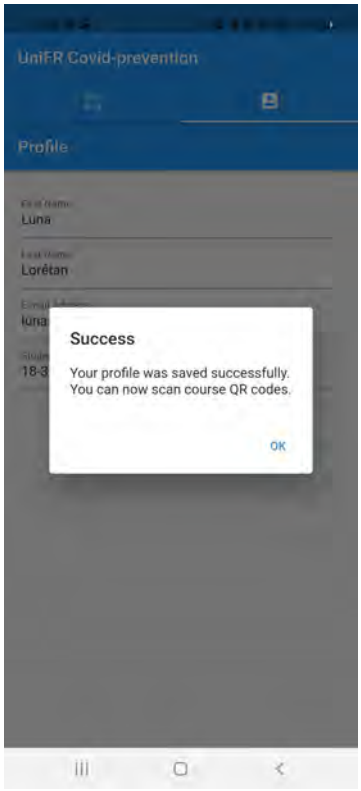

Figure 4.2. – Confirmation de l'enregistrement du profil

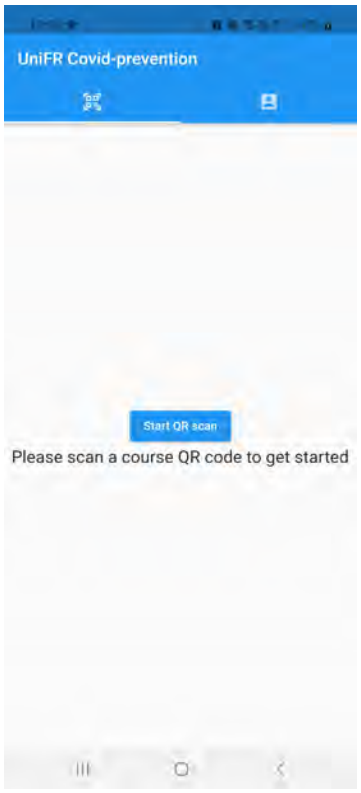

Figure 4.3. – Page de lecture de code QR

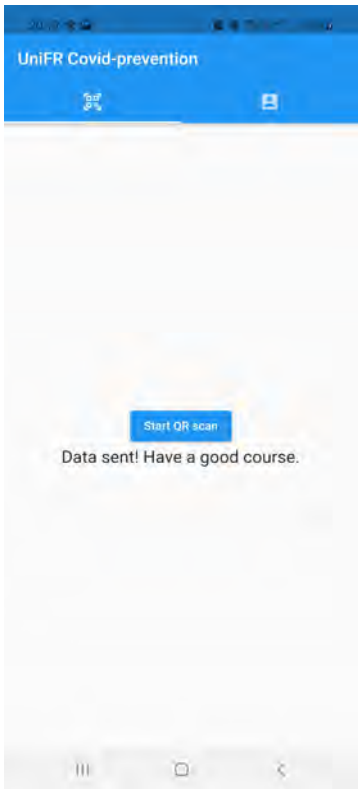

Figure 4.4. – Confirmation de l'envoi des données

Figure 4.5. – Lecteur Proxmark3 en train de lire une carte UniFR

Scénario 2
Jonas, en arrivant dans la salle de cours et en constatant qu'il n'a plus de batterie sur son smartphone, dépose sa carte sur le lecteur (fig. 4.5)

Dans le cas de ce scénario, la liaison du système de lecteur de cartes avec l'un des deux autres (site internet, application) est bien visible. En effet, lorsqu'un profil est créé sur le site ou l'application, il n'est pas possible de scanner le numéro interne de la carte de l'étudiant, celui-ci étant différent du numéro d'étudiant. Ici, le numéro interne de Jonas n'est lié à aucune entrée dans la base de données, il est donc nécessaire de demander une information supplémentaire (le numéro d'étudiant, car il est unique et relativement court).
Une fois que Jonas a entré ce numéro, celui-ci est retrouvé dans la base de données et son profil est complété avec le numéro interne de la carte.

Scénario 3
Ici, l'utilisation seule du site est démontrée.
Le soir avant son cours, Oliver se rend sur le site et arrive sur la page d'accueil (fig. 4.6) et y lit les instructions. Par la suite, il se rend dans la page de création de profil (fig. 4.7). Le jour du cours, il retourne sur le site internet et accède à la page d'enregistrement de présences (fig. 4.8).

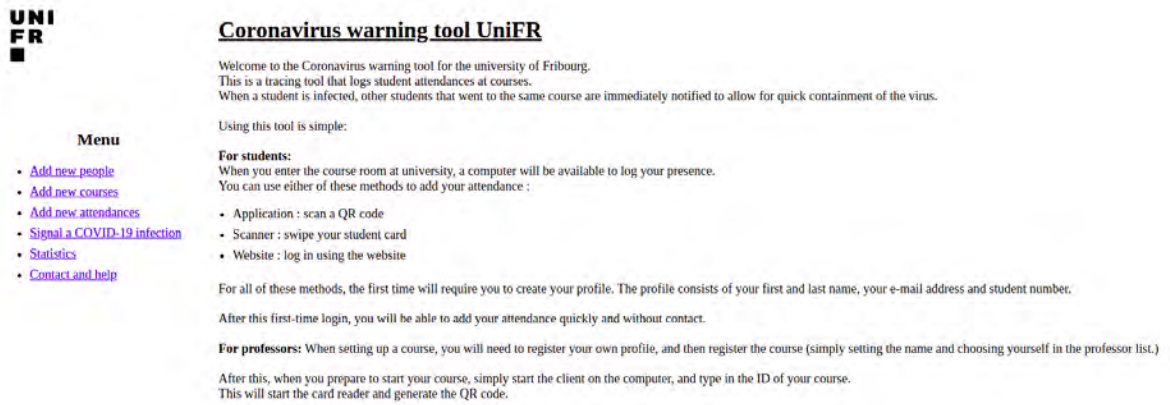

Figure 4.6. – Page d'accueil du site du projet

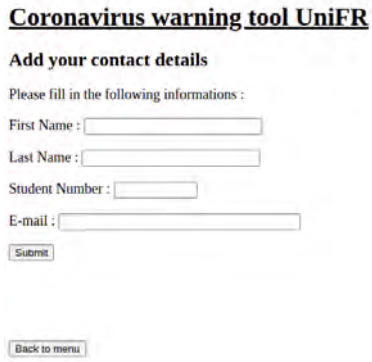

Figure 4.7. – Page de création de profil sur le site

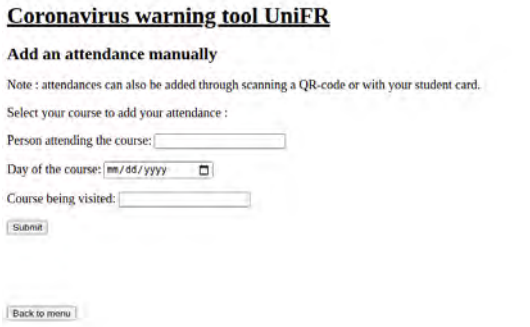

Figure 4.8. – Page d'enregistrement de présences

Scénario 4
On voit dans ce Use-case un exemple court et simple de ce qui était recherché avec ce travail. Noémie, ayant déjà un profil avec sa carte dans le système, peut simplement passer sa carte sur le lecteur et tout le reste est réglé.

Scénario 5
Ce scénario démontre principalement la capacité pour les étudiants de se créer leur profil, peu importe le moyen utilisé. Si, comme dans le scénario 2, le numéro interne de la carte lue par le lecteur n'est pas dans la base de données, il va être demandé d'entrer le numéro d'étudiant. Cependant, si celui-ci n'est pas connu non plus, on va alors demander les informations manquantes pour créer un profil entier, soit l'adresse e-mail, le nom et prénom de l'étudiant (fig. 4.9). Une fois le profil enregistré, la présence au cours est sauvegardée aussi.

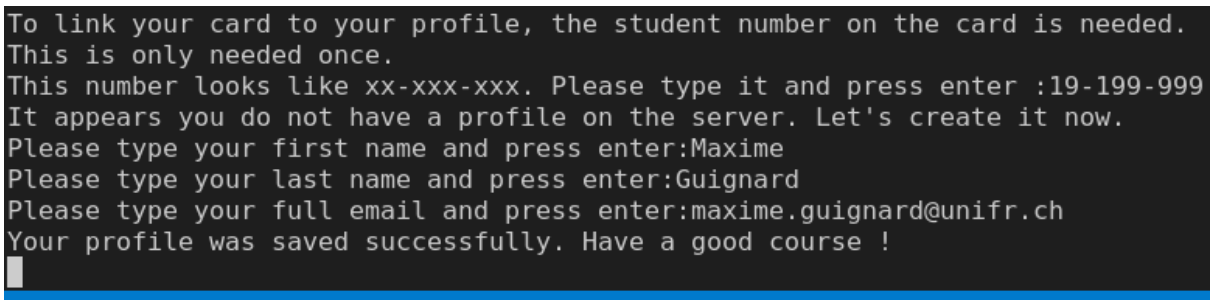

Figure 4.9. – Création d'un profil entier depuis le lecteur de cartes

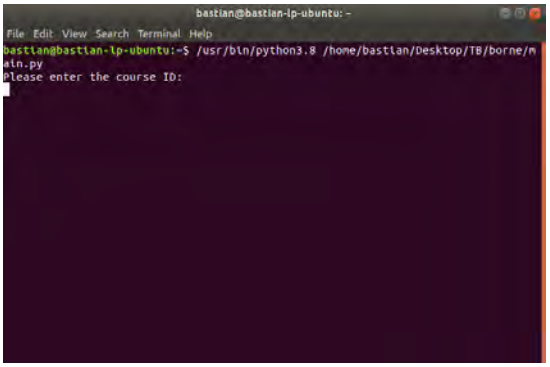

Figure 4.10. – Lancement initial du programme pour la lecture des cartes

Figure 4.11. – Confirmation avec code QR

4.1.2. Professeur

Concentrons-nous maintenant sur les Use-cases dédiés aux professeurs.

Scénario 1
M. Pasquier, afin de préparer l'utilisation du lecteur de cartes, ainsi que pour afficher le code QR utilisé par l'application mobile, lance le programme dédié à cet effet dans la salle de classe. Celui-ci lui demande d'entrer l'identifiant unique de son cours (fig. 4.10). Suite à cela, un court message lui confirme que le lecteur de cartes est prêt et un code QR apparaît à l'écran (fig. 4.11).

Scénario 2
Pour une première utilisation du système au début du semestre, Mme Rettig se rend sur le site internet et son profil étant déjà créé, accède à la page de création de cours (fig. 4.12).
Il est intéressant de noter que pour le choix du professeur, le site ne permet pas d'entrer quelqu'un qui n'est pas enregistré. Il faut impérativement choisir un nom dans la liste, une saisie automatique permet de trouver plus rapidement la personne correspondante.

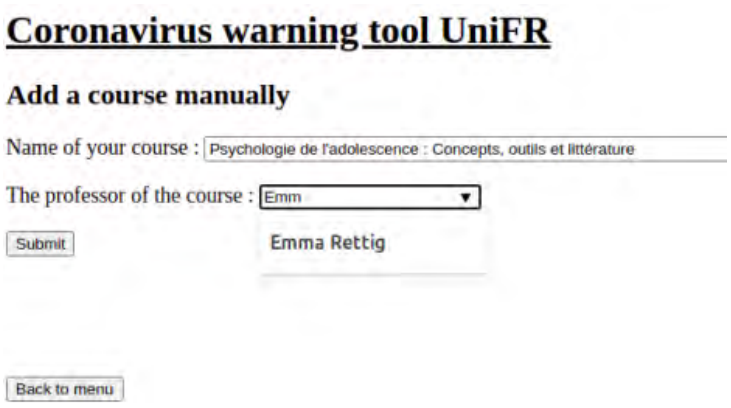

Figure 4.12. – Page de création de cours

4.2. Utilité sur le terrain

Il s'agit maintenant de parler de l'utilité de la solution sur le terrain.
Bien que ce projet soit avant tout un prototype à l'heure actuelle, il remplit sa fonction et permet d'enregistrer les présences de manière effective.
Dans la réalité, si le but est d'intégrer ce système à l'Université de Fribourg, l'utilisation d'un lecteur de cartes représente le problème majeur de cette solution. En effet, le Proxmark3 utilisé dans le cadre de ce travail est un outil de recherche et n'est donc pas vraiment viable dans une application plus classique comme celle d'authentification des étudiants. Un tel lecteur de cartes coûte environ 400 CHF et il est à noter que le risque de vol ou de mauvaise utilisation des lecteurs est assez élevé.
Il pourrait être pertinent de s'informer sur la possibilité d'utiliser à la place les lecteurs déjà présents à l'extérieur de nombreuses salles de cours.

En dehors de ce problème, la solution est tout à fait adéquate dans le cadre d'un enregistrement de présences. Si un système de signalement des infections lui était ajouté, son utilité sur le terrain serait d'autant plus intéressante.

4.3. Ergonomie de la solution

L'ergonomie de la solution dans son ensemble dépend en partie du type d'outil utilisé lors de l'authentification. Nous allons donc analyser l'ergonomie de chacune des méthodes prises à part.

Site internet
Le site internet remplit dans l'ensemble bien sa fonction. Différentes mesures ont été prises pour rendre son utilisation simple et efficace, notamment la saisie automatique mentionnée auparavant (fig. 4.12).

De plus, l'arborescence du site est excessivement simple : les trois pages qui servent à enregistrer respectivement les profils, les cours et les présences sont reliées par la page

d'accueil. Il n'y a ni sous-menu ni interface compliquée, tout est conçu afin d'être très vite compris. Ceci a été pensé pour permettre aux utilisateurs une meilleure autonomie et une plus grande satisfaction lors de l'utilisation du site.

Il faut néanmoins noter qu'il reste pour l'heure minimaliste visuellement parlant et qu'il pourrait donc bénéficier de quelques retouches afin de le rendre plus agréable à naviguer.

Lecteur de cartes
Le programme qui gère le lecteur de cartes fonctionne entièrement dans le terminal. Cela signifie que pour l'utilisateur standard qui n'est pas un étudiant en informatique, il n'est pas très agréable à voir.

Cependant, dans la plupart des cas, l'interaction avec le terminal n'est pas requise (sauf lors d'une première utilisation, comme discuté dans les Use-cases). Un changement envisageable serait de permettre l'ouverture directe du site internet lorsqu'une entrée de l'utilisateur est indispensable.

En dehors de ce point, l'utilisation du lecteur de cartes est satisfaisante. Il suffit de poser sa carte quelque part sur le lecteur, d'attendre environ une seconde et c'est terminé.

Application mobile
L'application mobile est selon moi, de loin la méthode la plus agréable à utiliser. Le design intégré de base dans Flutter permet une conception simple d'applications esthétiques. Son utilisation rend leur création plus fluide et naturelle qu'en CSS, par exemple, où tout est possible, mais rien n'est vraiment facile ni intuitif.

Sa simplicité et son attrait visuel en font un outil réellement ergonomique et pratique.

5
Expansion

Dans cette section, nous allons tout d'abord décrire les propriétés d'autres solutions du même genre, afin de pouvoir discuter des avantages et des inconvénients par rapport à notre projet. Dans un second temps, nous discuterons des fonctionnalités supplémentaires qui pourraient y être ajoutées.

5.1. État de l'art

De nombreux autres systèmes de tracking existent et nous allons à présent en passer en revue quelques-uns. Le présent travail s'en inspire évidemment, à la différence près que celui-ci a été conçu pour répondre aux besoins spécifiques de l'Université.

ok-visit
La solution ok-visit[8] (parfois ok-resto selon son utilisation) est celle utilisée à l'heure de l'écriture de ce travail par la Mensa de l'Université de Fribourg. Le déroulement est relativement simple et contient deux manières d'enregistrer sa présence : soit en téléchargeant l'application ok-visit, pour pouvoir ensuite scanner le code QR de l'établissement lorsqu'on y rentre ; ou alors en scannant directement le code QR sans pour autant avoir l'application, puis en suivant les instructions sur le site.
Une troisième méthode pour les personnes n'ayant pas de smartphone existe, il est possible de demander à l'établissement une carte avec un code numérique écrit dessus. Cette carte peut ensuite être montrée lors de l'arrivée et du départ.

En termes d'ergonomie et de présentation, ok-visit adopte un style minimaliste et simple (fig. 5.1), les éléments à l'écran sont peux nombreux, donc facilement compris et assimilés.

Il est intéressant de noter que, par la nature d'une visite dans un établissement tel qu'un restaurant, il n'est pas possible d'anticiper la durée de la présence. Il faut donc, dans tous les cas, annoncer son départ à l'aide d'un bouton dans l'application, d'un autre code QR ou encore d'une autre présentation de la carte.

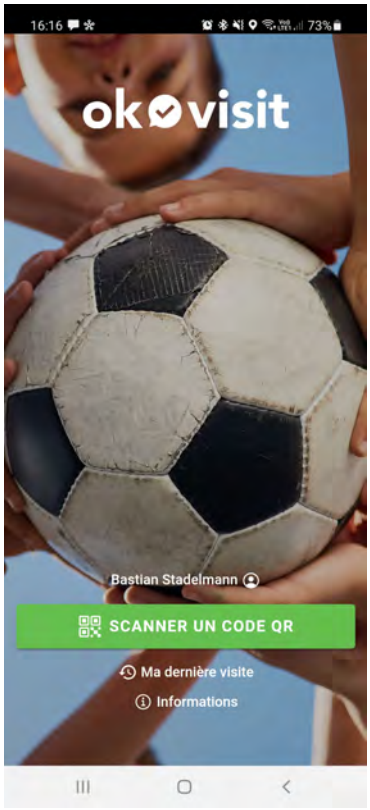

Figure 5.1. – Application ok-visit

Ce problème ne se pose pas dans notre système, car la longueur d'une visite est déterminée par celle du cours auquel l'étudiant assiste.

En somme, ok-visit est assez semblable à notre projet dans son exécution et dans son aspect visuel, bien que plus adapté au cadre d'un restaurant qu'à celui de l'Université.

SocialPass
SocialPass est une autre application utilisée pour le tracking du COVID-19[9]. Dans celle-ci, de la même manière que dans la notre, les données personnelles sont entrées par l'utilisateur et un code QR est scanné ensuite afin d'enregistrer une visite.
Un deuxième système existe en parallèle : après la saisie des données d'une personne, l'application peut générer un pass, celui-ci peut ensuite être lu par l'établissement lors de la visite.

Un désavantage notoire de SocialPass réside dans le fait qu'une inscription sans l'application n'est, à priori, pas possible. Un utilisateur sans smartphone (ou avec une batterie vide) devrait donc discuter avec les membres de l'établissement, trouver un autre smartphone pour s'inscrire ou, bien pire, se résoudre à ne pas s'inscrire du tout.

SocialPass ne déroge pas à la règle, son design est également conçu pour être simple et efficace. Son utilisation est simple et intuitive, ce qui est aussi l'effet visé pour notre système (fig. 5.2).

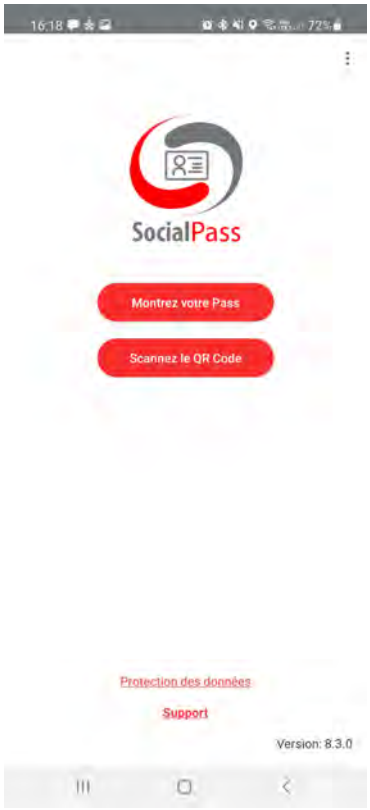

Figure 5.2. – Application SocialPass

En conclusion, notre solution remplit les mêmes fonctions de base que les autres applications de tracking utilisées communément en Suisse. Elle est graphiquement un peu moins intéressante, mais propose plusieurs manières pour l'utilisateur d'enregistrer sa présence, ce qui lui donne un certain avantage. De plus, sa conception est entièrement pensée pour répondre aux besoins du milieu universitaire. La possibilité d'enregistrement par le biais du lecteur de cartes offre notamment une alternative utile lors d'éventuels problèmes rencontrés avec les autres méthodes.

5.2. Fonctionnalités supplémentaires

Nous allons à présent aborder les différentes fonctionnalités qui auraient potentiellement pu venir compléter ce projet. Celles-ci ont toutes été considérées puis mises de coté, faute de temps ou parce qu'elles s'éloignaient de l'objectif de la solution à proprement parler.

Signalisation d'infection et avertissements par e-mail
Cette feature est de loin celle qui aurait été la plus utile à rajouter au projet. En effet, dans l'état actuel, le système permet d'enregistrer les présences, mais il n'est pas possible de signaler une infection au COVID-19 ou d'en avertir les autres étudiants.

Il était prévu dans l'application ainsi que le site un menu pour signaler une infection, avec la date du test positif. Le serveur ayant notifié cette information, aurait ensuite envoyé l'un des deux types de mails suivants aux élèves ayant eu un contact plus ou moins

direct avec la personne malade. Un premier message de plus haute priorité en cas de contact direct (un étudiant ayant été dans le même cours est malade) et un second pour les contacts indirects (un élève ayant eu contact direct avec un malade est, par la suite, rentré en contact avec un autre étudiant).

Rajouter cet élément aurait fait de ce travail un véritable système de tracking à part entière.

Notifications pour l'application
En lien avec la feature précédente, si le système d'alerte avait été mis en place, il aurait été intéressant d'envoyer aux utilisateurs de l'application mobile des notifications en plus des mails. En effet, les étudiants consultent probablement davantage leur téléphone que leurs mails universitaires. La prise en compte plus rapide de ces notifications permettrait donc d'éviter de plus nombreux contacts et donc la propagation des risques.

Interface graphique pour le lecteur de cartes
Le lecteur de cartes, bien qu'étant une vraie solution pour l'enregistrement de présences, n'est ni très ergonomique ni très abouti esthétiquement parlant. Il aurait alors été intéressant d'intégrer celui-ci dans une interface graphique pour en rendre l'utilisation plus agréable.

6 Conclusion

Dans ce travail, nous avons suivi la mise en place des concepts de base nécessaires à la création d'un système de tracking pour le COVID-19, spécialisé pour le cadre des cours à l'Université. Nous avons ensuite retracé la réalisation de ces concepts et leur implémentation aux différents niveaux de la solution, avec les problèmes et les défis que cela a pu impliquer.
Par la suite, nous avons analysé le résultat final, notamment par rapport aux aspects fonctionnels et ergonomiques, l'avons comparé à d'autres systèmes similaires et avons finalement présenté des ajouts potentiels au système.

Ce travail m'a permis de développer de nouvelles compétences sur de nombreux moyens informatiques. Par exemple, je n'avais encore jamais travaillé aussi directement avec une base de données ou un ORM. J'ai donc pu apprendre à manier ces nouveaux outils et à en appréhender plus pratiquement les avantages et les inconvénients.

Je sais à présent créer une application mobile avec un langage de programmation qui m'était auparavant totalement inconnu, ce qui m'a motivé à développer de nouvelles connaissances rapidement en fournissant un travail concret et utile.

Finalement, l'avantage le plus important pour mon avenir professionnel selon moi réside dans la vision d'ensemble et la capacité à planifier que j'ai acquises durant ce travail. Sachant que je me lançais dans un projet d'une plus grande envergure que jusqu'à présent, j'ai dû apprendre à agencer beaucoup d'éléments afin de les faire fonctionner ensemble. Il s'agit là d'une compétence qui va certainement s'avérer précieuse à l'avenir.

Pour conclure, ce travail m'a donné l'opportunité de créer un projet intéressant et utile, tout en développant d'importantes nouvelles capacités.

A License of the Documentation

Bibliographie

[1] Outil pour créer des schémas de base de données. https://dbdiagram.io/home (dernière consultation le août 28, 2021).

[2] Documentation d'aiohttp. https://docs.aiohttp.org/en/stable (dernière consultation le août 28, 2021).

[3] Méthodes HTTP. https://www.restapitutorial.com/lessons/httpmethods.html (dernière consultation le août 30, 2021).

[4] Documentation de TortoiseORM. https://tortoise-orm.readthedocs.io/en/latest (dernière consultation le août 28, 2021).

[5] Repository Github de l'outil Proxmark3. https://github.com/RfidResearchGroup/proxmark3 (dernière consultation le août 28, 2021).

[6] Page de présentation de Flutter. https://flutter.dev/?gclsrc=ds&gclsrc=ds (dernière consultation le août 28, 2021).

[7] Site web du langage de programmation Dart. https://dart.dev/ (dernière consultation le août 30, 2021).

[8] Site internet de ok-visit. https://ok-visit.ch (dernière consultation le août 28, 2021).

[9] Site internet de SocialPass. https://www.socialpass.ch (dernière consultation le août 28, 2021).

[10] Free Documentation Licence (GNU FDL). http://www.gnu.org/licenses/fdl.txt (dernière consultation le août 29, 2021).