

SMART PARKING LI-FI

Revue de projet finale – Rapport commun – V. 3 – 31/05/2018



Lancelot POULIN-PONNELLE

Quentin MERMAZ

BTS SNIR 2

Créé le 5/03/2018

Vincent BRUNAUD

Thomas DA SILVA

SESSION 2018

Table des matières

1	Introduction.....	4
2	Partie commune : analyses et modélisation	6
2.1	Diagrammes des cas d'utilisation.....	6
2.2	Modèle conceptuel de la base de données.....	8
2.3	Diagramme de séquence général.....	9
2.4	Architecture réseau.....	9
2.5	Gestion de projet.....	10
2.6	Diagramme de déploiement.....	10
3	Parties personnelles	11
3.1	Partie ACCÉDER au parking – smartphone	11
3.1.1	Principe de fonctionnement.....	11
3.1.2	Synoptique de l'architecture matérielle et logicielle	12
3.1.3	Travail réalisé.....	14
3.1.4	Conclusion	33
3.2	Partie SURVEILLER l'état du parking – carte embarquée	34
3.2.1	Spécifications.....	34
3.2.2	Synoptique de l'architecture matérielle et logicielle	35
3.2.3	Travail réalisé.....	36
3.2.4	Conclusion	50
3.3	Partie SUPERVISER sur l'application PC.....	51
3.3.1	Spécifications.....	51
3.3.2	Présentation des technologies utilisées	51
3.3.3	Synoptique de l'architecture matérielle et logicielle	52
3.3.4	Travail effectué.....	53
3.3.5	Conclusion	64
3.4	Partie SYNCHRONISER web service – base de données	65
3.4.1	Synoptique de ma partie	65
3.4.2	Diagramme de Gantt	65
3.4.3	Travail réalisé.....	66
3.4.4	Conclusion	74
4	Conclusion générale	75
5	Annexes	77

Table des illustrations

Figure 1: Structure simplifiée du projet	4
Figure 2: Cas d'utilisation global.....	6
Figure 3: Cas d'utilisation ACCÉDER parking	6
Figure 4: Cas d'utilisation SURVEILLER parking	7
Figure 5: Cas d'utilisation SUPERVISER sur PC.....	7
Figure 6: Modèle conceptuel de données.....	8
Figure 7: Architecture réseau.....	9
Figure 8: Diagramme de Gantt	10
Figure 9: Diagramme de déploiement.....	10
Figure 10: Structure de la partie ACCÉDER parking.....	13
Figure 13: Raspberry PI 3.....	13
Figure 11: Lampe à LED	13
Figure 12: Module Li-Fi.....	13
Figure 14: Photorésistance et flash de téléphone	14
Figure 15: Tablette avec un récepteur Li-Fi Oledcomm (audio).....	14
Figure 16: Diagramme de Gantt de la première itération de la partie ACCÉDER au parking.....	14
Figure 17: Diagramme de Gantt de la deuxième itération de la partie ACCÉDER au parking	15
Figure 18: Extraction et décompilation de l'archive Java « lifi_lib.jar ».....	15
Figure 19: Méthode Java pour lire l'audio.....	15
Figure 20: Nouvelle méthode C#.....	16
Figure 21: Logiciel de conversion Java en C#	16
Figure 22: Diagramme de classe simplifié de la bibliothèque Li-Fi	17
Figure 23: Résultat sur la tablette sous la lampe Li-Fi.....	17
Figure 24: Câblage des broches Raspberry UART.....	18
Figure 25: Câblage TX de la Raspberry vers l'émetteur Li-Fi	18
Figure 26: Lampe à LED dirigée vers le récepteur Li-Fi branché en USB	19
Figure 27: Console du programme d'envoie UART	19
Figure 28: Console de réception du port COM correspondant au récepteur Li-Fi.....	20
Figure 29: Maquette de l'application	21
Figure 30: Interface graphique de l'application	22
Figure 31: Package NuGet PCLStorage	22
Figure 32: Requête HTTP en C#.....	23
Figure 33: Diagramme de classe simplifié de l'IHM de l'application smartphone	25
Figure 34: Algorithme d'envoi de données avec le flash.....	25
Figure 35: Données Li-Fi reçus sur le smartphone.....	26
Figure 36: Montage pour le relevé de la courbe de charge d'un condensateur.....	27
Figure 37: Courbe de charge d'un condensateur	27
Figure 38: Courbe du rapport résistance/éclairement de la LDR.....	28
Figure 39: Câblage de la photorésistance sur la Raspberry	28
Figure 40: Programme pour la détection du flash du smartphone.....	29
Figure 41: Console affichant les temps de charge du condensateur	29
Figure 42: Structure de la partie SURVEILLER parking	34
Figure 43: Une Raspberry PI 3	35
Figure 44: Carte microSD SanDisk	35
Figure 45: Capteur à ultrasons HC-SR04.....	35
Figure 46: LEDS	35

Figure 47: RasbianOS.....	36
Figure 48: Win32Diskimager	36
Figure 49: Code::Blocks	36
Figure 50: Câblage du capteur à ultrasons.....	37
Figure 51: Ultrasons émis	37
Figure 52: Trame envoyée par le capteur	38
Figure 53: Principe du capteur d'obstacle.....	38
Figure 54: Résultat des distances calculés	39
Figure 55: Structure de la partie SYNCHRONISER web service - BDD	65
Figure 56: Diagramme de Gantt de la partie SYNCHRONISER.....	65
Figure 57: Modèle MWB de la base de données	66
Figure 58: Jeux de test de la table "Réservation".....	66
Figure 59: Script de test MySQL	67
Figure 60: Seconde version du modèle MWB de la base de données	67
Figure 61: Second jeux de test de la table "Réservation"	68
Figure 62: Gestionnaire de bases de données en ligne.....	69
Figure 63: La base de données sur le serveur	69
Figure 64: Communication Client <-> Web Service <-> BDD.....	70
Figure 65: Fichier de connexion du web service	70
Figure 66: Fichiers du web service PHP	70
Figure 67:Statut et adresse du web service	72
Figure 68:Fichiers en ligne.....	72
Figure 69:Algorithme pour générer nombre unique ET aléatoire	73
Figure 70:Algorithme codé en PHP	73

1 Introduction

Ce projet consiste à fournir un équipement informatique permettant de contrôler les entrées d'un parking en utilisant la technologie Li-Fi. L'accès à ce dernier s'effectuera par l'intermédiaire de réservations faites au préalable par l'utilisateur qui se verra attribuer une place à l'horaire indiqué.

Chaque utilisateur peut réserver une place de parking à l'aide d'une application téléchargeable sur son smartphone. Lorsqu'il se situe devant l'entrée du parking il présente son smartphone devant la borne Li-Fi afin de se voir attribuer une place. Ces dernières sont surveillées par un système de détection de présence qui indique leurs disponibilités. Ces informations sont sauvegardées sur un serveur, et, de plus, ces données sont administrées via un logiciel de supervision.

Le projet est découpé en quatre parties :

- **Partie ACCÉDER au parking - smartphone par *Lancelot Poulin-Ponelle*** : authentification de l'utilisateur, réservation d'une place de parking, accès au parking et attribution d'une place par communication Li-Fi, affichage des informations de l'utilisateur.
- **Partie SURVEILLER état du parking - carte embarquée par *Quentin Mermaz*** : détection de la disponibilité des places à l'aide de capteurs, échange d'informations avec la base de données et communication socket avec l'application supervision.
- **Partie SUPERVISER avec l'application sur PC par *Vincent Brunaud*** : authentification de l'utilisateur, gestion des utilisateurs et de leurs réservations, sauvegarde de la base de données, communication socket avec la carte embarquée afin de vérifier l'état de l'installation.
- **Partie SYNCHRONISER web service - base de données par *Thomas Da Silva*** : échange d'informations avec l'application smartphone via un web service, gestion de la base de données, communication depuis la base de données vers le poste de supervision et la carte embarquée.

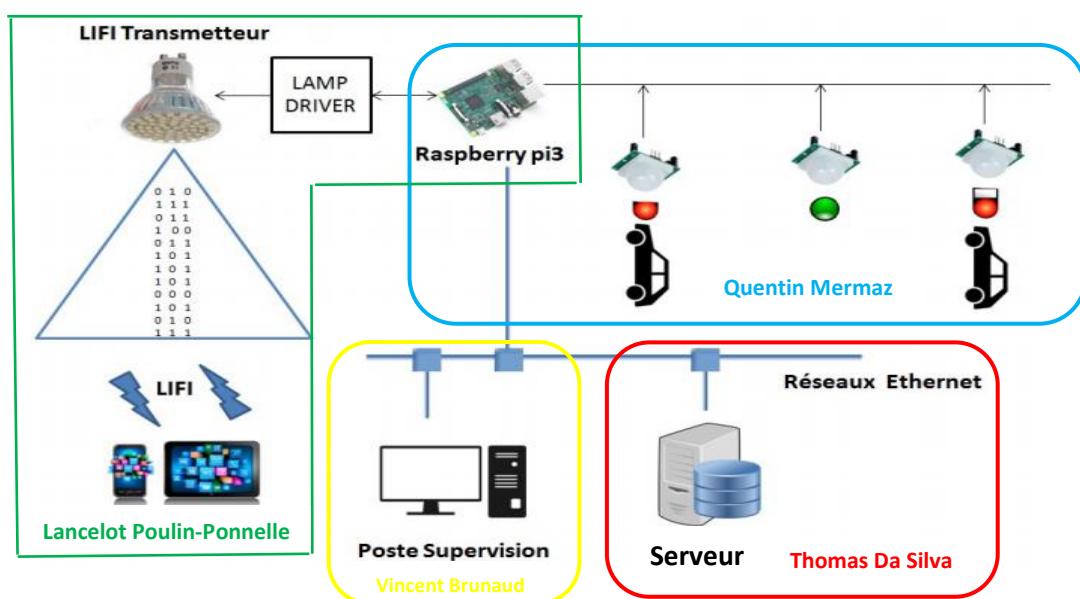


Figure 1: Structure simplifiée du projet

Pour ce qui est de la partie commune nous avons commencé par effectuer une analyse initiale afin de conceptualiser le projet à l'aide des différents diagrammes. Nous avons également étudié sa structure et réalisé un modèle abstrait de la base de données. Pour finir nous avons représenté les différentes tâches qui constituent le projet sur toute sa durée.

Dans un premier temps nous verrons les différentes analyses effectuées par l'équipe afin d'illustrer au mieux le cahier des charges. Nous vous présenterons ensuite les parties personnelles du projet avant de dresser un bilan de celui-ci.

2 Partie commune : analyses et modélisation

L'objectif de cette partie est de mettre en perspective le travail effectué en équipe : la mise en commun de nos différents points de vue afin de s'accorder sur les tâches à réaliser et d'obtenir une meilleure analyse du projet.

2.1 Diagrammes des cas d'utilisation

- Cas d'utilisation global : on peut retrouver ici les quatre parties du projet énoncés plus tôt

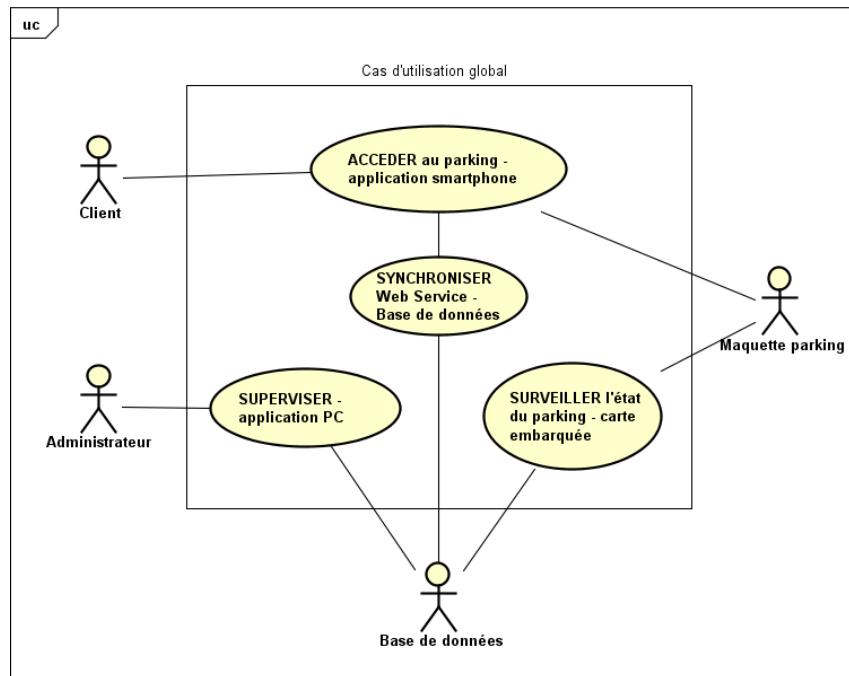


Figure 2: Cas d'utilisation global

- Cas d'utilisation de la partie ACCEDER au parking – application smartphone

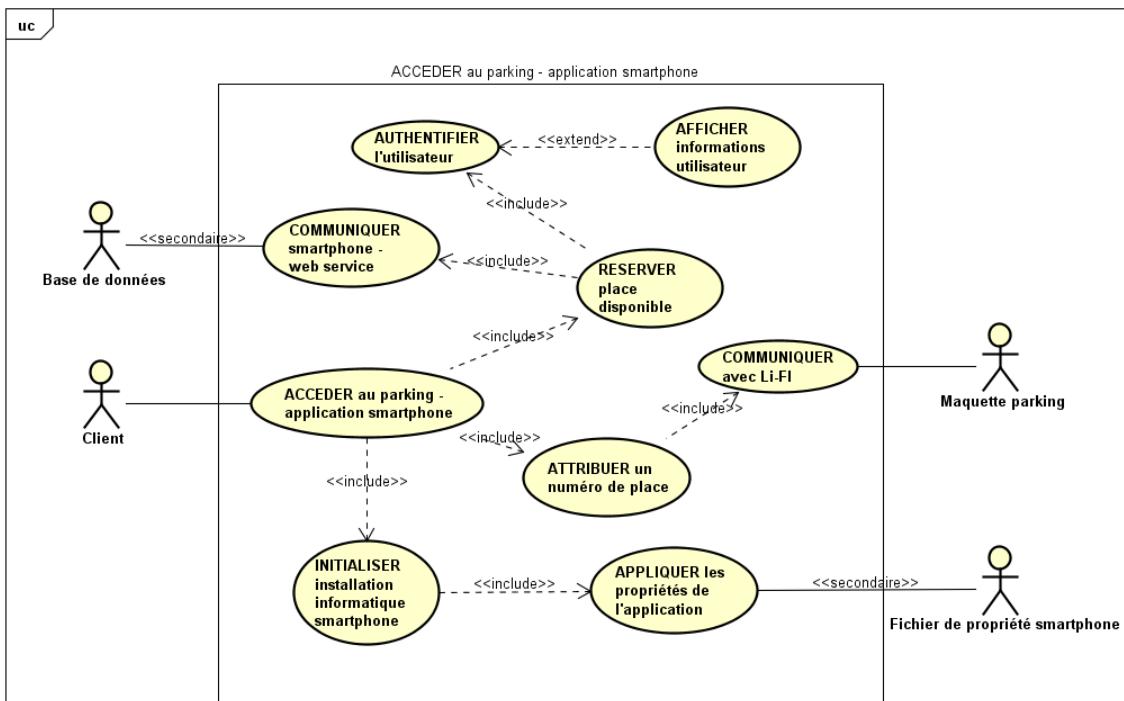


Figure 3: Cas d'utilisation ACCEDER parking

- Cas d'utilisation de la partie SURVEILLER l'état du parking – carte embarquée

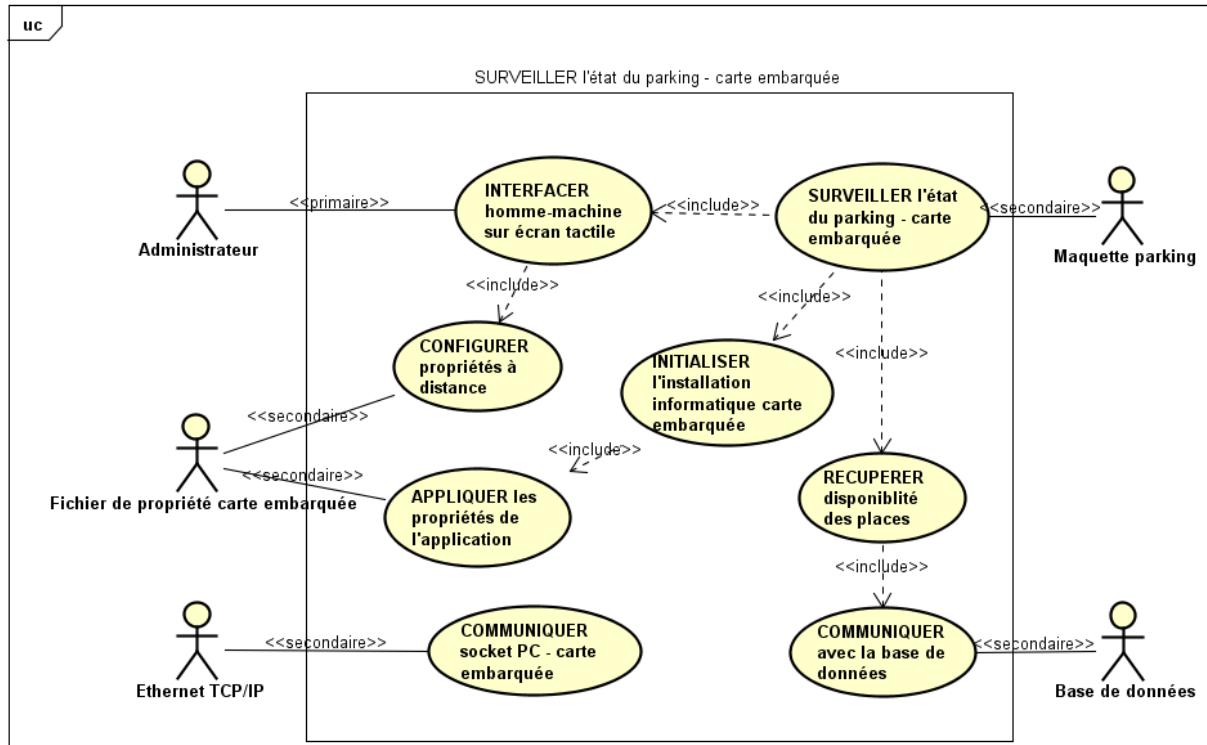


Figure 4: Cas d'utilisation SURVEILLER parking

- Cas d'utilisation de la partie SUPERVISER avec l'application sur PC

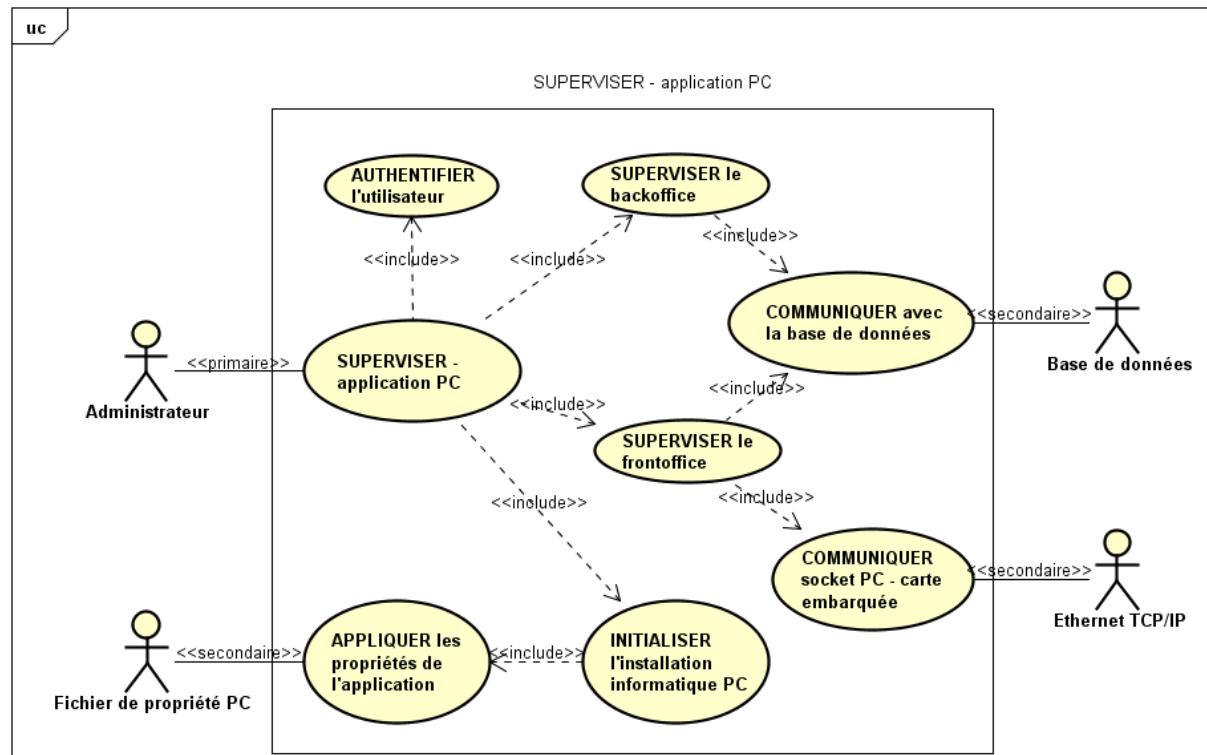


Figure 5: Cas d'utilisation SUPERVISER sur PC

Les cas d'utilisation de la partie SYNCHRONISER web service – base de données sont représentés dans le diagramme global ainsi que dans les diagrammes des différentes parties.

2.2 Modèle conceptuel de la base de données

Voici la représentation de la base de données utilisée dans le projet (MCD). On y retrouve huit tables :

- ❖ Admin : Ce sont les utilisateurs qui pourront se connecter sur l'application de supervision.
- ❖ Client : Ces comptes utilisateur permettent à chaque client de s'authentifier sur l'application smartphone, afin d'effectué une réservation en leur nom.
- ❖ Reservation : Les réservations futures et passées des clients, une place leur est attribuée dans un intervalle de temps.
- ❖ Place : Les places de parking, elles ont un numéro et on indique si elles sont disponibles.
- ❖ Type : Les types de place : normale, handicapée, et secours.
- ❖ Lane, Floor, Parking : Les places sont dans une allée lettrée, qui se trouvent à un étage numéroté, qui appartient à un parking.

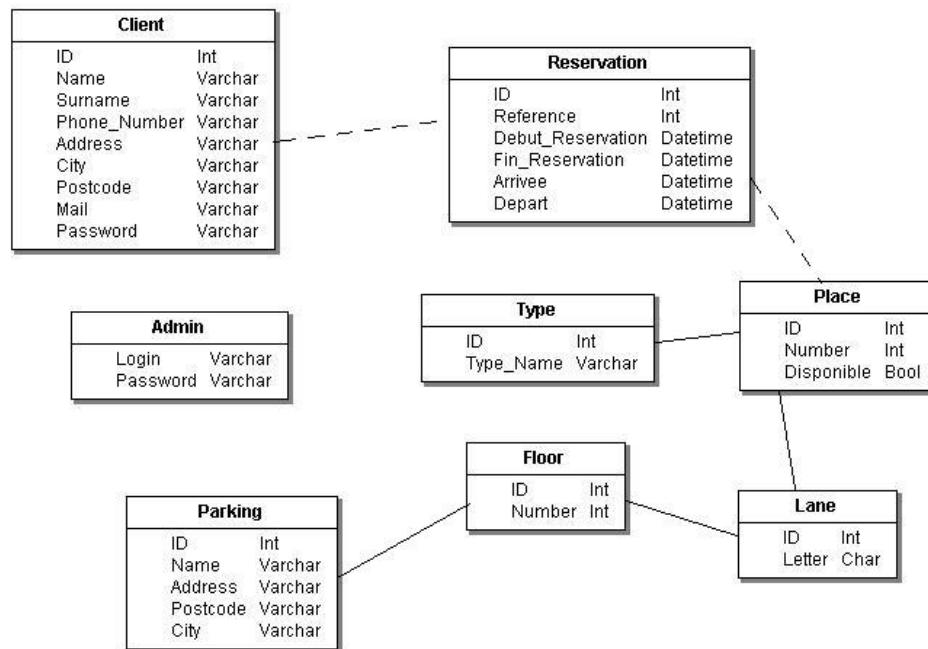
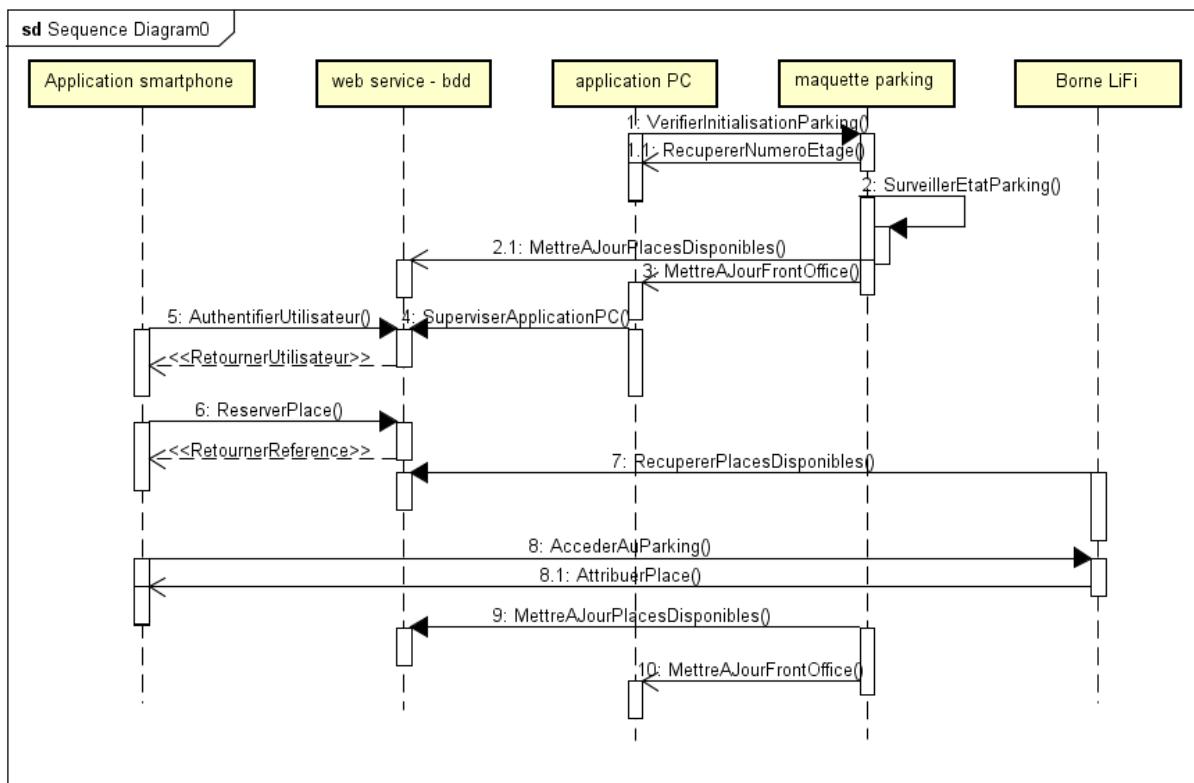


Figure 6: Modèle conceptuel de données

2.3 Diagramme de séquence général



2.4 Architecture réseau

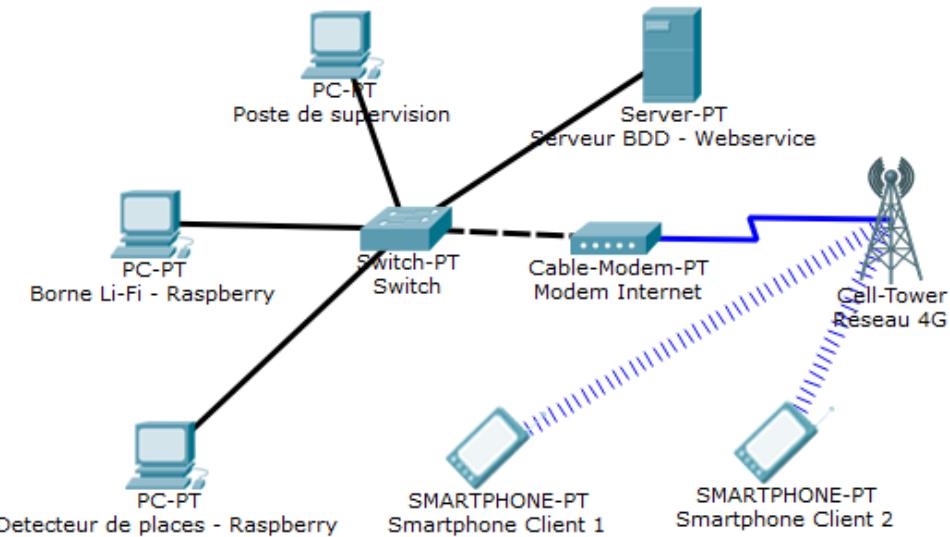


Figure 7: Architecture réseau

2.5 Gestion de projet

Le diagramme de Gantt nous permet de situer les tâches de chaque partie en fonction du temps. On peut distinguer l'analyse initiale réalisée au début du projet, les trois itérations avec les activités des parties (analyse, conception et test) que l'on verra plus en détail dans les parties personnelles. A la fin des itérations on retrouve les revues de projet correspondantes ainsi que le déploiement et les tests finaux avant la dernière revue.

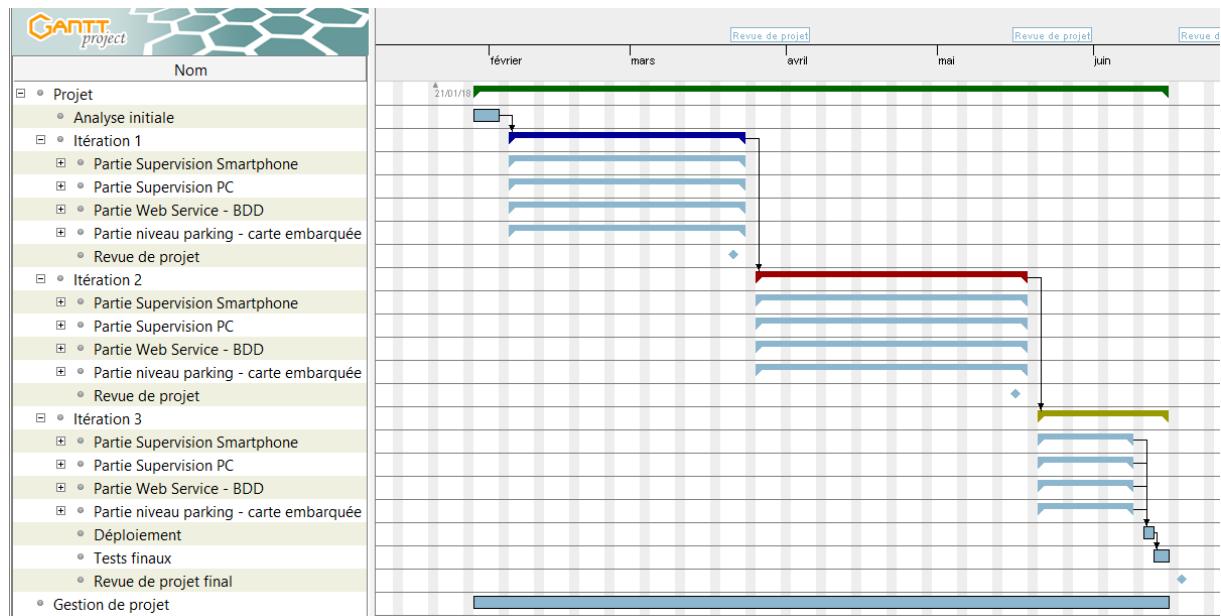


Figure 8: Diagramme de Gantt

2.6 Diagramme de déploiement

Le diagramme de déploiement est une vue statique qui sert à représenter l'utilisation de l'infrastructure physique par le système et la manière dont les composants du système sont répartis ainsi que leurs relations entre eux.

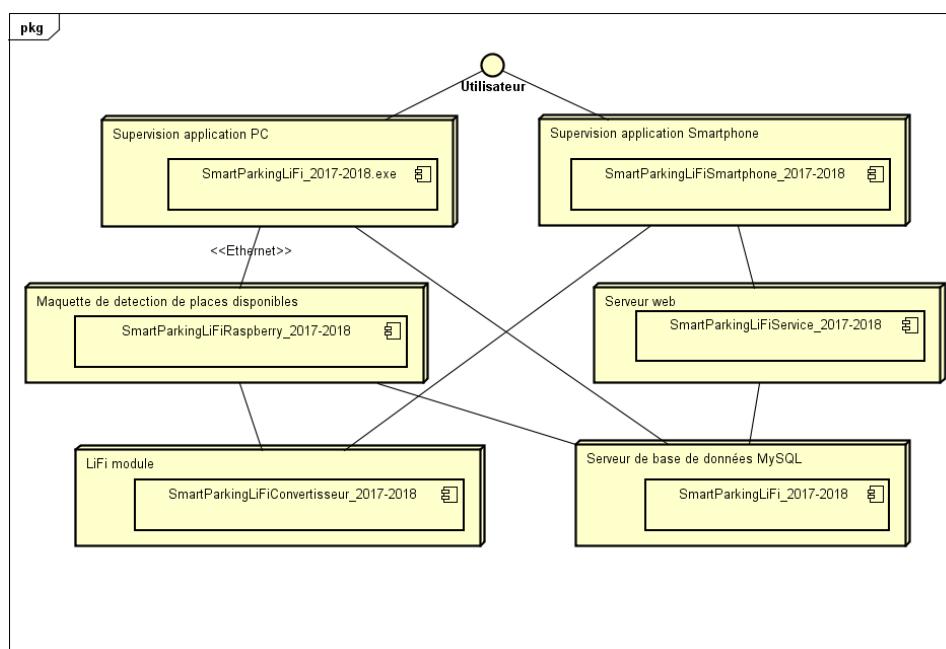


Figure 9: Diagramme de déploiement

3 Parties personnelles

Dans les parties suivantes nous verrons les travaux effectués par chaque membre de l'équipe du projet.

3.1 Partie ACCÉDER au parking – smartphone

Partie effectuée par Lancelot Poulin-Ponnelle

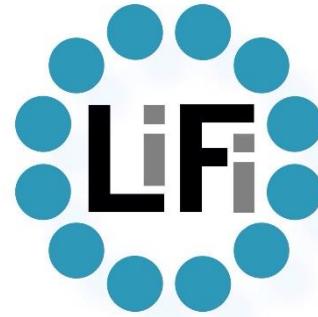
Cette partie consiste à la création d'une application mobile multiplateforme ainsi qu'à la programmation d'une Raspberry situé à l'entrée du parking permettant à l'utilisateur d'y accéder via la technologie Li-Fi.

3.1.1 Principe de fonctionnement

3.1.1.1 Le Li-Fi

La Li-Fi (ou Light Fidelity) est une technologie de communication sans fil basée sur l'utilisation de la lumière visible, de longueur d'onde comprise entre 480 nm (670 THz, bleu) et 650 nm (460 THz, rouge). Alors que le Wi-Fi utilise une partie radio du spectre électromagnétique hors du spectre visible, le Li-Fi utilise la partie visible (optique) du spectre électromagnétique. Le principe du Li-Fi repose sur le codage et l'envoi de données via la modulation d'amplitude des sources de lumière (scintillation imperceptible à l'œil), selon un protocole bien défini et standardisé.

Le Li-Fi est un type de système VLC (Visible Light Communication, transmission par la lumière visible). Il se différencie de la communication par laser, par fibre optique et de l'IrDa par ses couches protocolaires. Les couches protocolaires du Li-Fi sont adaptées à des communications sans fil jusqu'à une dizaine de mètres, soit légèrement plus que Bluetooth basse puissance, et moins que Bluetooth haute puissance ou Wi-Fi.



En 2012, il permettait une liaison entre un luminaire communiquant (lampe à LED) et un ordinateur, mais cette liaison était descendante et bas-débit. En 2015, il peut s'intégrer dans un réseau internet haut-débit en permettant une communication entre une lampe connectée au réseau via un câble Ethernet RJ45 avec capacité PoE (Power over Ethernet) et des ordinateurs distants de quelques mètres et dotés d'un capteur récepteur/émetteur spécial (qui pourrait bientôt ressembler à une simple clé USB). Associé à un système émetteur-récepteur infrarouge (pour le signal montant), il pourrait bientôt se connecter au Wi-Fi pour des liaisons bidirectionnelles à haut-débit, mais en restant limité en termes de distance à la source lumineuse (avec les techniques disponibles en 2015, au-delà de 10 à 15 m, le signal est dégradé).

3.1.1.2 Le système d'accès au parking

L'application est téléchargeable sur les stores correspondants, le smartphone devra être équipé d'un récepteur Li-Fi.

L'utilisateur pourra réserver à l'avance une place depuis l'application et, lors de son arrivée au parking, il émettra sa référence de réservation en dirigeant le flash de son téléphone vers la borne Li-Fi. En réponse, cette dernière enverra au smartphone le numéro de place attribué à l'utilisateur grâce aux lampes à LED intégrant un module Li-Fi. La barrière d'entrée du parking s'ouvrira alors. A sa sortie, l'utilisateur présentera son téléphone afin d'enregistrer son départ et ainsi ouvrir la barrière.

Une réservation peut se faire au maximum 24 heures à l'avance et une place lui sera garantie grâce aux places de secours, réservées aux adhérents du parking quand il est complet. Les invités (ainsi que les adhérents) qui ne réservent pas se verront attribuer une place selon leurs disponibilités dans les heures qui suivent.

Lors de la réservation, l'utilisateur saisit son heure d'arrivée, ainsi que son heure de départ. Un retard ou une avance de 15 minutes sont accordés. Le parking dispose de places handicapés, réservables en option sur l'application.

L'utilisateur pourra voir son historique de réservation ainsi que ses informations personnelles. Notre système ne gèrera pas les abonnements et paiements des clients du parking.

Pour 100 places de parking :

- 88 sont des places normales.
- 10 sont des places de secours, elles ne sont pas fixe : s'il il y en a une de prise, la prochaine place normale libérée deviendra une place de secours (celle prise deviendra normale), ce qui permettra d'en avoir toujours à disposition. Elles font donc l'objet d'un algorithme.
- 2 sont des places handicapées, conformément à la législation. Elles sont indépendantes des places de secours.
- Les places concernant les véhicules deux roues ne sont pas prises en compte par le système.

3.1.2 Synoptique de l'architecture matérielle et logicielle

La borne Li-Fi sera équipée d'une carte embarquée Raspberry PI 3 sous le système d'exploitation Raspbian (Debian). Cette carte permettra la réception et l'émission de données Li-Fi. L'application qui va gérer la borne sera programmée sous Code::Blocks en langage C++. Code::Blocks est un environnement de développement intégré libre et multiplateforme. Il est écrit en C++ et utilise la bibliothèque wxWidgets. Code::Blocks est orienté C et C++, mais il supporte d'autres langages comme FORTRAN ou le D. Code::Blocks existe pour Linux, Windows et Mac OS X.

L'application mobile sera développée sur l'environnement Visual Studio avec la bibliothèque Xamarin.Forms qui permet de créer des applications natives pour Android et iOS. A ses débuts, l'ambition de Xamarin était de proposer une implémentation open source de la plateforme .NET sous Unix, à la suite du succès de leur bibliothèque Mono, l'entreprise est rachetée par Microsoft.

Microsoft Visual Studio est un ensemble complet d'outils de développement permettant de générer des applications web ASP.NET, des services web XML, des applications bureautiques et des applications mobiles. Visual Basic, Visual C++, Visual C# utilisent tous le même environnement de développement intégré (IDE), qui leur permet de partager des outils et facilite la création de solutions faisant appel à plusieurs langages. Par ailleurs, ces langages permettent de mieux tirer parti des fonctionnalités du framework .NET, qui fournit un accès à des technologies clés simplifiant le développement d'applications web ASP et de services web XML grâce à Visual Web Developer.

Xamarin.Forms utilise une base de code unique : le C#. À la place de développer deux applications, on va pouvoir mettre les parties communes dans un seul projet comme les interfaces, les événements et les algorithmes. Et lorsque l'on veut utiliser un composant spécifique à une plateforme, il nous suffit d'utiliser les bibliothèques Xamarin.Android ou Xamarin.iOS, toujours dans le même projet : on indique à l'application le code qui s'exécutera sur Android ou iOS.

Pour l'authentification de l'utilisateur et l'enregistrement des réservations depuis l'application, je communiquerai avec le web service réalisé par mon coéquipier dans la partie webservice et base de données.

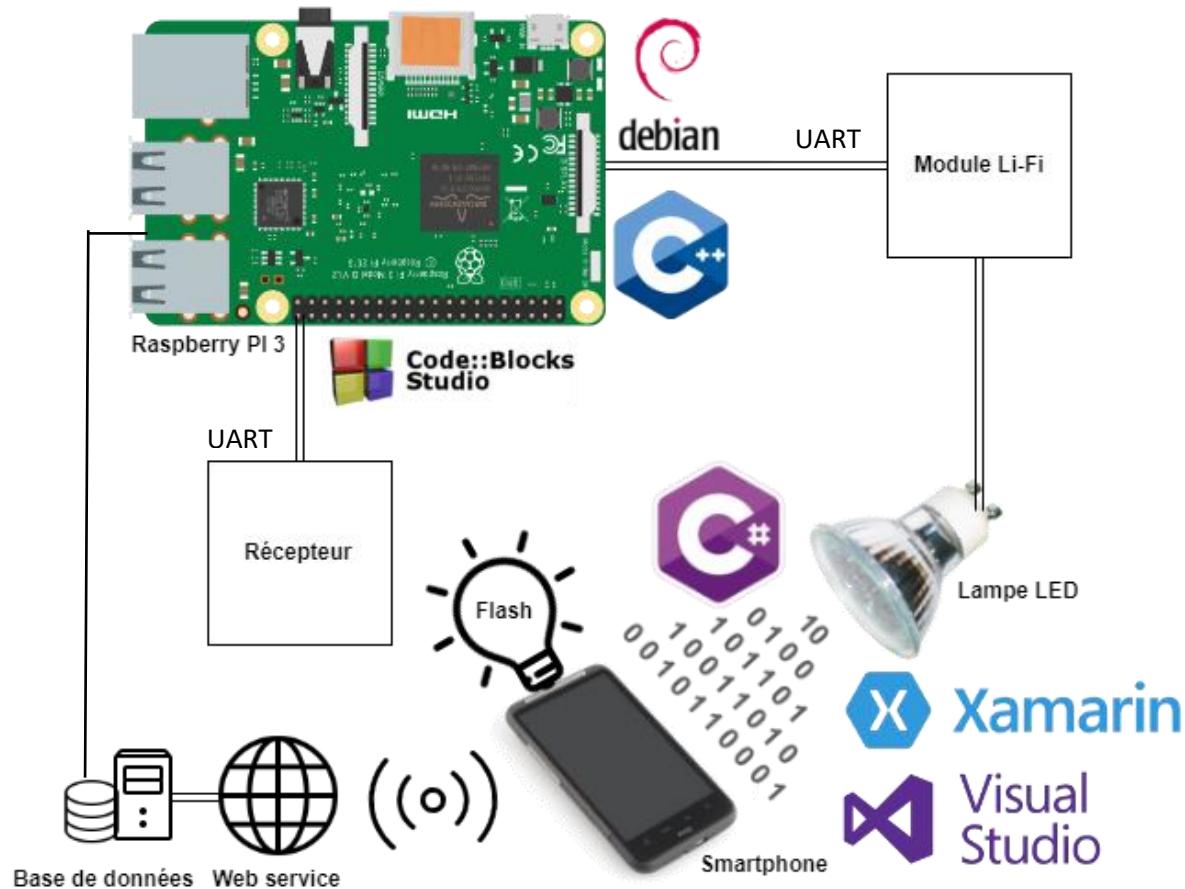


Figure 10: Structure de la partie ACCÉDER parking

J'aurais à disposition une Raspberry PI 3, ainsi qu'un module Li-Fi qui va convertir les données envoyées en série par la carte en trame Li-Fi que va émettre la lampe à LED.



Figure 13: Raspberry PI 3



Figure 12: Module Li-Fi



Figure 11: Lampe à LED

De plus, j'ai une tablette équipée d'un récepteur Li-Fi ainsi qu'un flash qui permettra de recevoir les informations des lampes et d'en envoyer au récepteur de la carte qui est un capteur d'intensité.



Figure 15: Tablette avec un récepteur Li-Fi Oledcomm (audio)



Figure 14: Photorésistance et flash de téléphone

Il faut savoir que plus tard nos smartphones capteront le Li-Fi grâce à l'appareil photo, et qu'un kit de développement spécifique aux plateformes rendra la tâche bien plus simple pour communiquer en Li-Fi. Aujourd'hui je possède une bibliothèque développée en Java Android par la société Oledcomm, conçue pour fonctionner avec le récepteur Li-Fi fourni. Nous verrons ultérieurement comment je l'ai adapté à une application mobile Xamarin C#.

3.1.3 Travail réalisé

3.1.3.1 Tâches détaillées

Pendant la première itération, je devais tout d'abord créer l'interface graphique de l'application, gérer l'authentification de l'utilisateur en local (le web service étant fini à la deuxième itération) et enregistrer ses informations de connexion dans un fichier de propriété. Il fallait que j'effectue des tests de réservation et d'affichage de celle-ci. Enfin, l'installation de la Raspberry et sa communication avec la base de données pour ainsi émettre les places disponibles en Li-Fi.

- • Création de l'interface graphique
 - Maquettage de l'interface
 - Analyse graphique et UML
 - Réalisation de l'interface
- • Communication Web Service - Fichier de propriétés
 - Méthodes de requête Web Service
 - Sauvegarde configuration utilisateur - web service
- • Gérer l'authentification d'un client/visiteur
 - Analyse algorithme et UML
 - Authentification
 - Test de connexion en local
 - Afficher les informations de l'utilisateur
- • Gérer les réservations
 - Analyse algorithme et UML
 - Ajout d'une réservation
 - Afficher réservations et places disponibles
 - Test d'ajout et d'affichage en local
- • Envoi des places en Li-Fi avec la Raspberry - Lampe
 - Analyse algorithme et UML
 - Installation de la raspberry - lampe Li-Fi
 - Récuperer réservations et places disponibles depuis la BDD
 - Réalisation d'un objet d'envoi Li-Fi avec lampes
 - Envoi d'un numéro de place par réservation
 - Test d'envoi

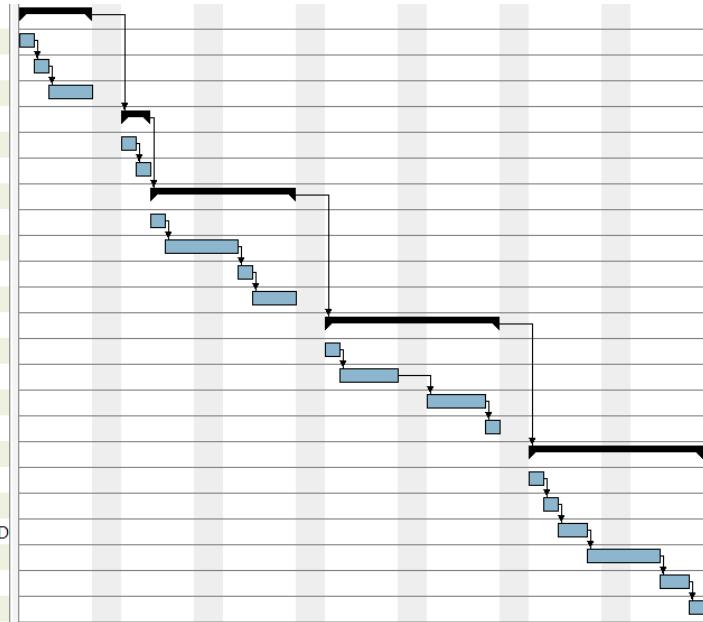


Figure 16: Diagramme de Gantt de la première itération de la partie ACCÉDER au parking

Pour la deuxième itération, j'avais planifié la réception des places depuis la borne Li-Fi sur le smartphone grâce au module Li-Fi et à la bibliothèque convertit. Je devais ainsi tester la réception des places ainsi que l'envoie d'une confirmation d'arrivée au parking (envoie de la référence) grâce à l'émission de données Li-Fi avec le flash de la caméra du smartphone vers la borne Li-Fi.

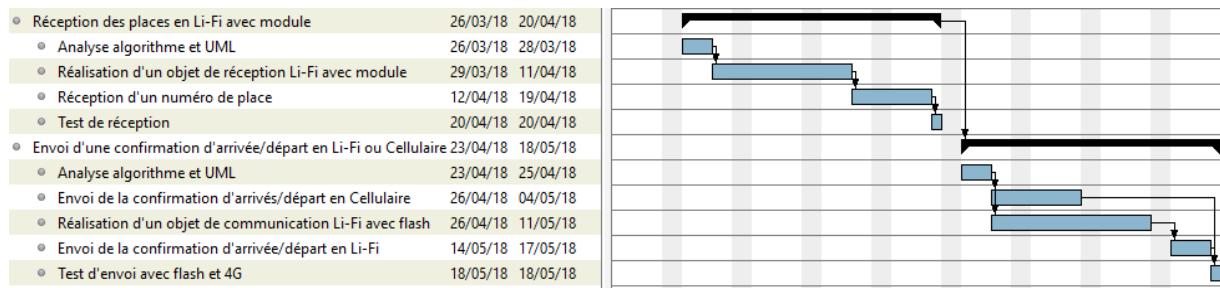


Figure 17: Diagramme de Gantt de la deuxième itération de la partie ACCÉDER au parking

3.1.3.2 Choix techniques

Premièrement, j'ai vérifié que la bibliothèque Li-Fi en Java pouvait être utilisé dans une application C#. Malheureusement, à cause de certaines dépendances à des composants Android, le fichier JAR (Java Archive) n'est pas intégrable à une application .NET. J'ai aussi essayé de la convertir en fichier DLL mais sans succès.

L'archive Java était extractible ce qui m'a permis de récupérer les classes de la bibliothèque. J'ai converti ces fichiers Class en fichier Java pour accéder au code grâce à un décompileur en ligne (<http://www.javadecompilers.com/>)

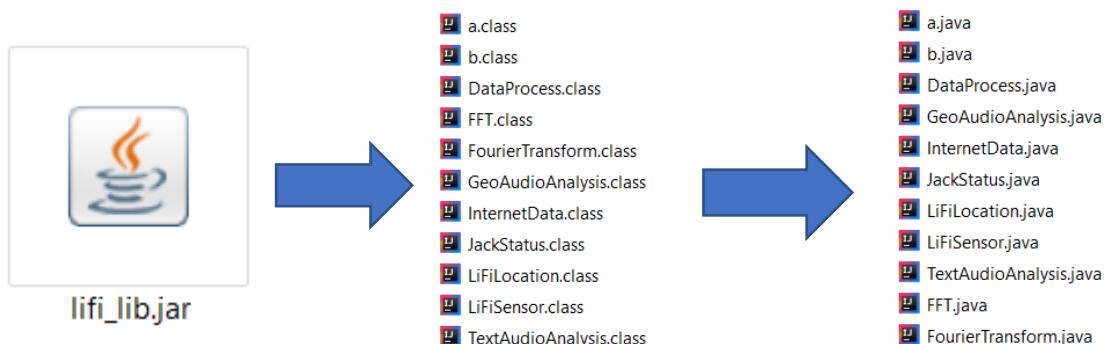


Figure 18: Extraction et décompilation de l'archive Java « lifi_lib.jar »

J'ai donc décidé de continuer sur cette voie et convertir cette bibliothèque en C# ce qui permettra son utilisation dans une application Xamarin multiplateforme, au lieu de le développé que sur Android avec le langage Java.

3.1.3.3 Conversion de la bibliothèque Java Li-Fi en C#

Malgré le nom des fichiers, des méthodes et des variables non explicites, j'ai réussi à comprendre le fonctionnement de la bibliothèque, et comment il fallait l'utiliser. J'ai commencé à convertir les méthodes de chaque classe en C# ligne par ligne et aussi grâce à un outil de conversion mais qui reste très limité.

Voici un extrait de la classe Java LiFiSensor qui permet de récupérer les données reçues par le capteur Li-Fi par la prise audio :

```
protected AudioRecord a = new AudioRecord(0, 44100, 16, 2, this.b);
protected int c;
protected short[] d;
protected int e;

protected short[] c()
{
    this.d = new short[this.c];
    this.e = this.a.read(this.d, 0, this.c);
    return this.d;
}
```

Figure 19: Méthode Java pour lire l'audio

On peut voir que le nom des variables ne m'indique rien de ce qu'il se passe dans cette fonction. Après la conversion en C# plutôt simple pour une ces quelques lignes, j'ai renommé ces variables et commenté la méthode.

```
protected AudioRecord SensorRecord;
protected int ReceivedBufferLength;

- références
public LiFiSensor()
{
    SensorRecord = new AudioRecord(0, 44100, ChannelIn.Mono, Encoding.Pcm16bit, MinBufferSize);
}

// Lit la sortie audio et retourne le buffer contenant l'information
- références
protected virtual short[] ReadSensor(int BufferLength)
{
    short[] ReceivedBuffer = new short[BufferLength];
    ReceivedBufferLength = SensorRecord.Read(ReceivedBuffer, 0, BufferLength);
    return ReceivedBuffer;
}
```

Figure 20: Nouvelle méthode C#

On peut remarquer que le composant Xamarin Android « `AudioRecord` » a le même nom que celui en Java, et la méthode « `read` » prend une majuscule (standard en .NET). Concernant le nom des variables, j'ai réussi à trouver à quoi elle correspondait : le buffer, sa taille, la taille du buffer récupéré, etc. De même pour la création de l'objet « `AudioRecord` » qui a comme paramètre le nombre de bits (16) par trame ainsi que le type de canal (Mono).

Avec le logiciel de conversion :

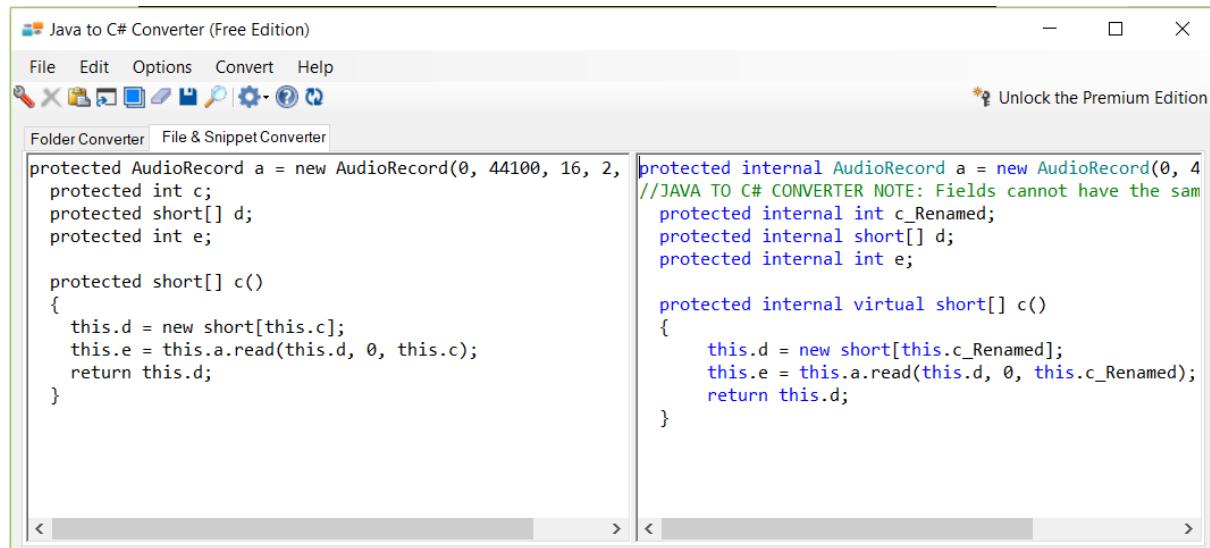


Figure 21: Logiciel de conversion Java en C#

On voit que ça fonctionne malgré le fait qu'il ne prend pas en compte les bibliothèques et donc je ne pouvais pas l'utiliser pour une classe ou des méthodes entières.

Après la conversion de toute l'archive j'ai pu dresser le diagramme de classes simplifiés ci-dessous.

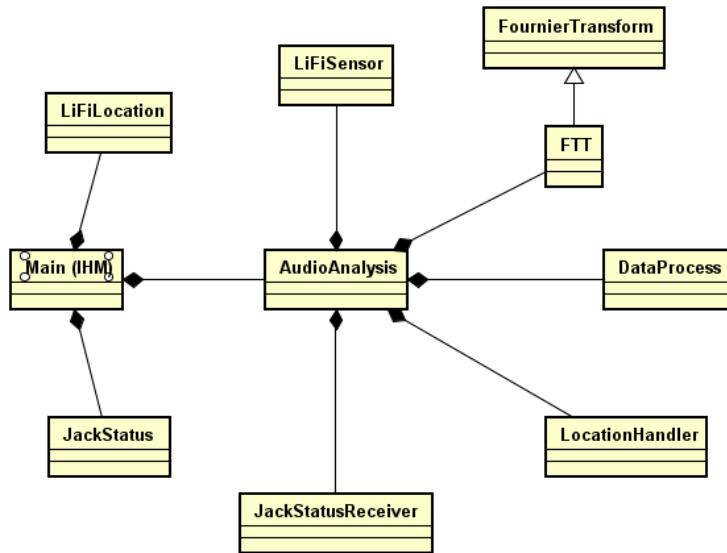


Figure 22: Diagramme de classe simplifié de la bibliothèque Li-Fi

L'interface homme-machine crée un objet de classe `AudioAnalysis` qui va utiliser le récepteur Li-Fi avec différents objets (`JackStatus` quand le récepteur est branché en jack, `DataProcess` pour convertir les bytes en données hexadécimal, la transformée de Fournier pour la fréquence d'émission...) pour recevoir les données émises depuis la lampe.

Cette partie de mon travail était une des plus importantes et elle m'a pris beaucoup de temps à réaliser, j'ai donc pris du retard sur l'interface graphique, mais de l'avance sur la communication Li-Fi. C'était une étape obligatoire pour l'avancement de l'application car elle m'a permis de savoir sur quelle technologie j'allais travailler : Xamarin ou Java Android.

3.1.3.4 Test de la communication Li-Fi

Afin de savoir si ma nouvelle bibliothèque C# fonctionnait, j'ai créé une application qui va pouvoir tester la réception de trames Li-Fi. J'ai simplement créé un objet `AudioAnalysis` qui est la classe principale qui permet de récupérer les données du capteur. J'ai affiché les résultats à l'écran :

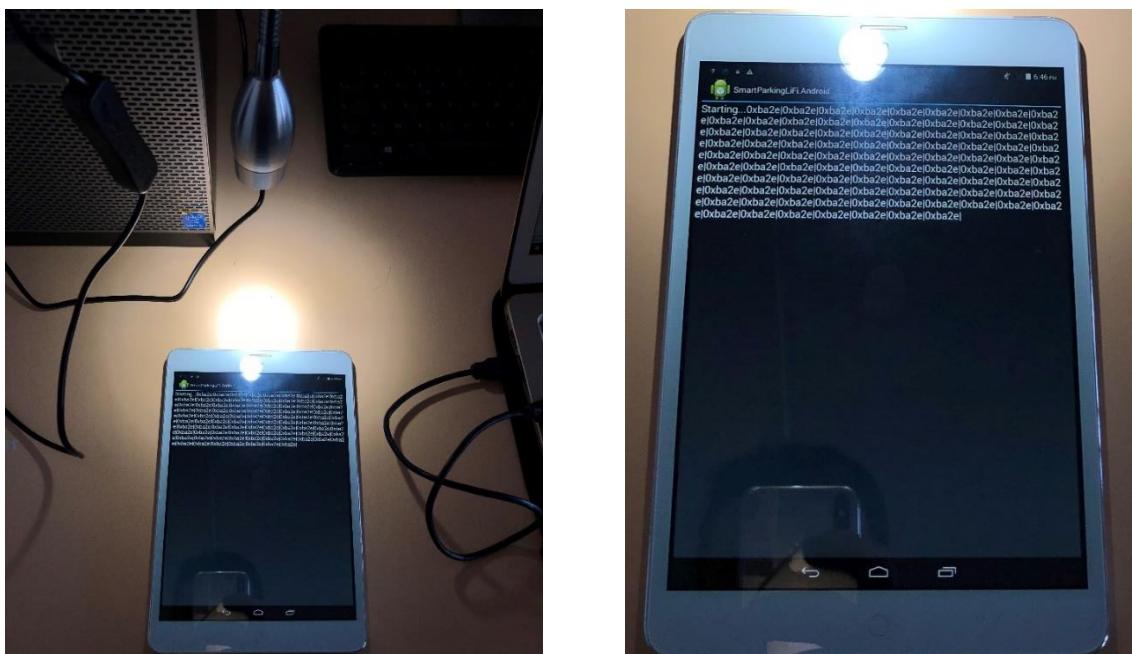


Figure 23: Résultat sur la tablette sous la lampe Li-Fi

Lorsque la tablette (équipée d'un récepteur Li-Fi) passe sous la lampe à LED, l'écran affiche le « tag » de la lampe, ce tag correspond à l'identifiant de la lampe. Ici, je n'envoie aucune information au boîtier d'émission. Chaque lampe possède un identifiant basé sur sa fréquence d'émission : le récepteur détecte la fréquence et le programme utilise la classe FFT qui est autre qu'un calcul utilisant la transformation de Fourier.

Après ce premier succès, j'ai tout de suite eu l'envie d'effectuer un test d'émission de la Raspberry à la lampe à LED. Après l'installation de Raspbian et de Code::blocks sur la carte (disponible en annexe), j'ai dû me documenter sur les liaisons UART utilisées par le module Li-Fi. Ce sont des liaisons séries monodirectionnelles : TX pour l'envoie, et RX pour la réception.

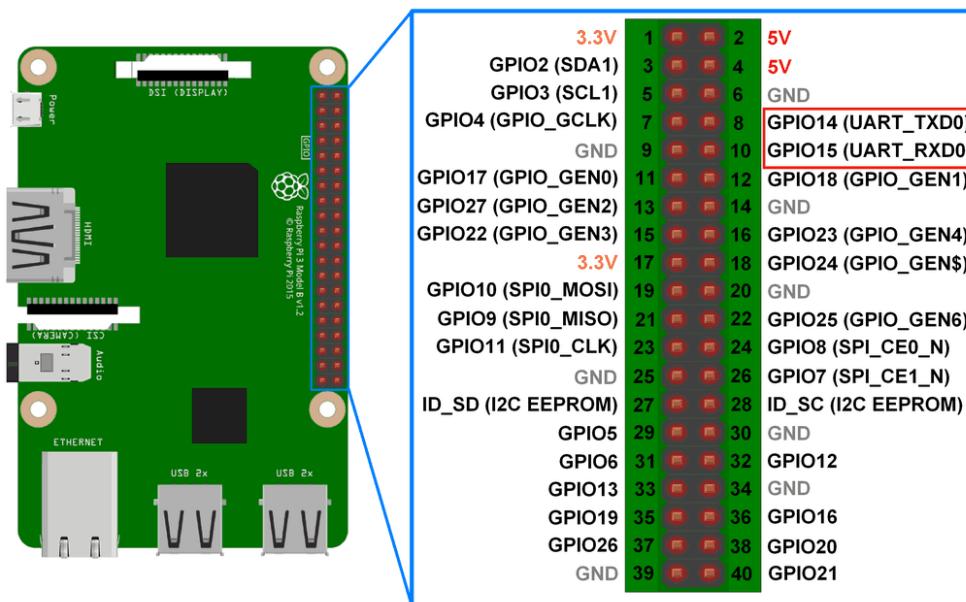


Figure 24: Câblage des broches Raspberry UART

Voici mon câblage :

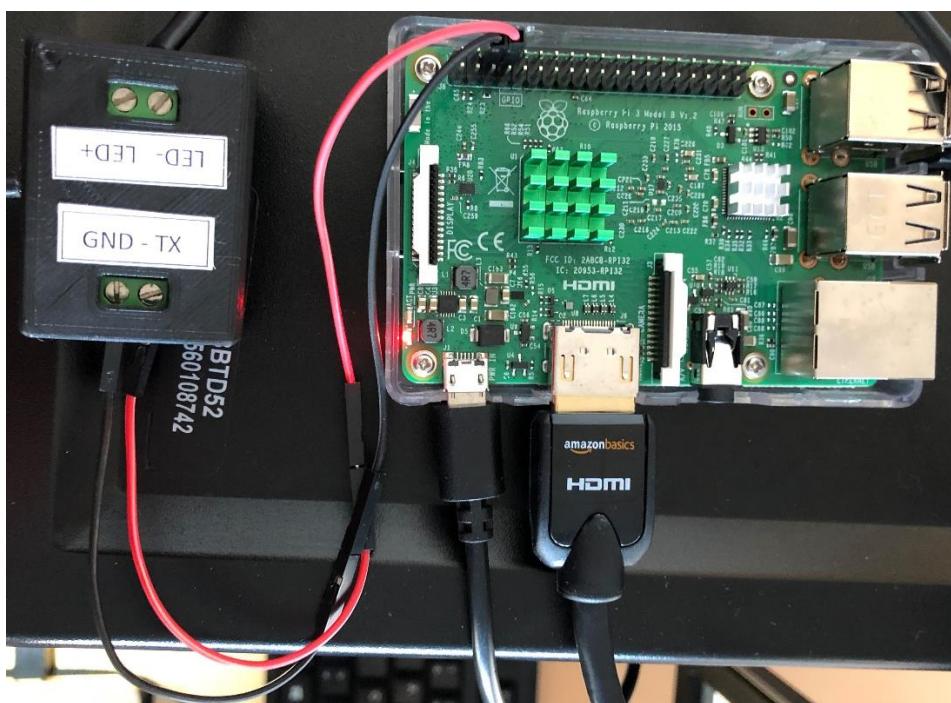


Figure 25: Câblage TX de la Raspberry vers l'émetteur Li-Fi

Il faut aussi installer le composant « ttyS0 » qui permet d'utiliser les port UART sur la Raspberry PI 3. J'ai développé une application procédurale qui envoyait « Hello World ! » sur la sortie TX de la Raspberry. (Installation et algorithme en annexe)

Pour vérifier le fonctionnement du système j'ai branché un récepteur Li-Fi vers USB (fourni par Oledcomm) sur mon ordinateur. A l'aide du logiciel RealTerm, j'ai configuré (comme sur la Raspberry : 115200baud, 8 bits et 1 bit de stop) et ouvert le port COM correspondant. En plaçant le récepteur sous la lampe je recevais bien le message « Hello World » dans mon terminal.



Figure 26: Lampe à LED dirigée vers le récepteur Li-Fi branché en USB

The screenshot shows the Code::Blocks IDE interface. On the left, the Project Manager displays a workspace named 'Test_LiFi'. The main window shows the code for 'main.cpp'. The code is a C program that sends the string 'Hello World!' over a serial connection. The terminal window titled 'Test_LiFi' shows the output of the program, which consists of repeated messages 'Writing data success'. At the bottom, the 'Logs & others' tab bar includes tabs for 'Code:Blocks', 'Search results', 'Build log', 'Build messages', and 'Debugger'.

```
33
34
35
36
37
38     //---- TX BYTES ----
39     unsigned char tx_buffer[20];
40     unsigned char *p_tx_buffer;
41
42     p_tx_buffer = &tx_buffer[0];
43     *p_tx_buffer++ = 'H';
44     *p_tx_buffer++ = 'e';
45     *p_tx_buffer++ = 'l';
46     *p_tx_buffer++ = 'l';
47     *p_tx_buffer++ = 'o';
48     *p_tx_buffer++ = ' ';
49     *p_tx_buffer++ = 'W';
50     *p_tx_buffer++ = 'o';
51     *p_tx_buffer++ = 'r';
52     *p_tx_buffer++ = 'l';
53     *p_tx_buffer++ = 'd';
54     *p_tx_buffer++ = '!';
55     *p_tx_buffer++ = '!';
56
57     if (uart0_filestream != -1)
58     {
59         int count = write(uart0_filestream, &tx_buffer[0], (p_tx_buffer - &tx_buffer[0]));           //Filestream, bytes to wr.
60         if (count < 0)
61         {
62             printf("UART TX error\n");
63         }
64         else
65         {
66             printf("Writing data success\n");
67         }
68     }
69 }
```

----- Run: Debug in Test_LiFi (compiler: GNU GCC Compiler)-----
Checking for existence: /home/pi/POULIN/Projet LiFi/Test_LiFi/bin/Debug/Test_LiFi
Executing: xterm -T Test_LiFi -e /usr/bin/cb_console_runner LD_LIBRARY_PATH=\$LD_LIBRARY_PATH:. /home/pi/POULIN/Projet\ LiFi/Test_LiFi/bin/Debug/Test_LiFi

Figure 27: Console du programme d'envoie UART

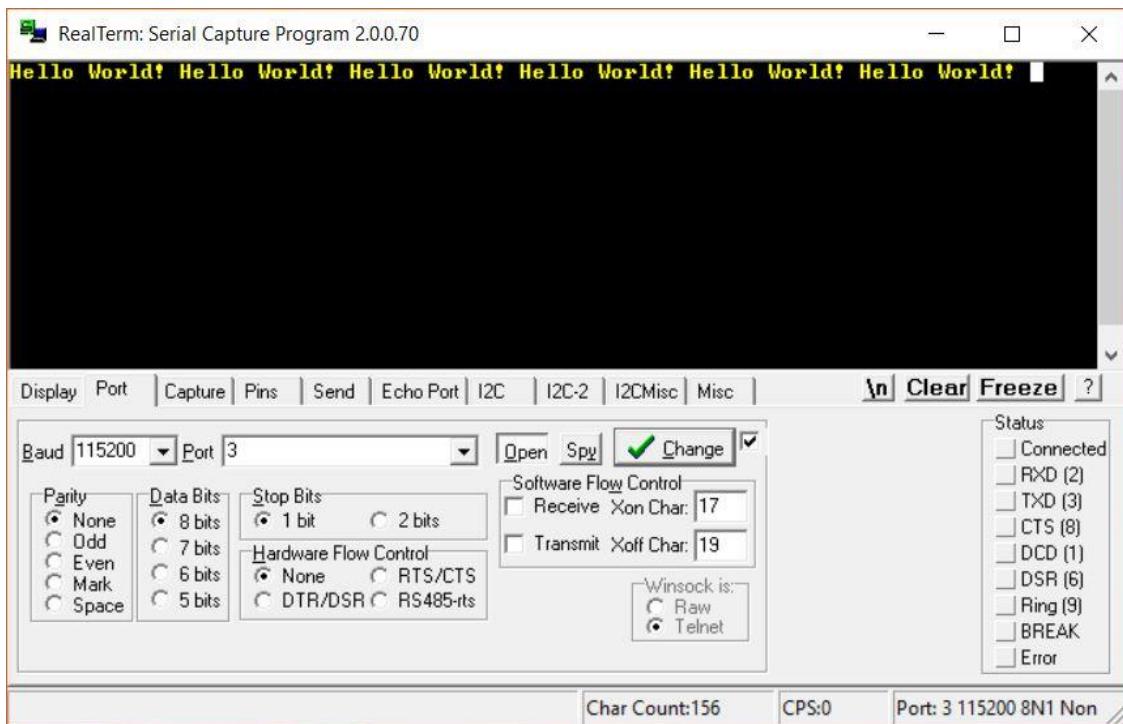


Figure 28: Console de réception du port COM correspondant au récepteur Li-Fi

Pour finir, j'ai essayé de réceptionner des données avec ce même récepteur mais cette fois ci branché sur l'entrée RX de la Raspberry (on peut voir une sortie UART avec des broches sur le récepteur). Avec le code pour recevoir disponible dans la même annexe, je recevais les informations parfaitement.

J'émettais et recevais des données avec la technologie Li-Fi.

3.1.3.5 Interface graphique de l'application

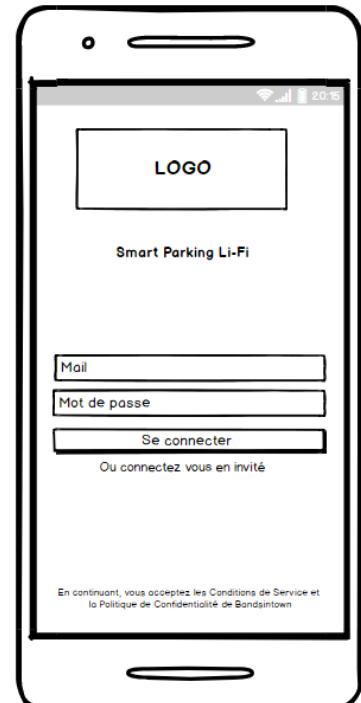
J'ai ensuite commencé le maquettage de mon application à l'aide d'un outil web gratuit : Balsamiq. J'ai réalisé les maquettes des différentes pages de mon application : page d'authentification, de réservation, d'émission/réception Li-Fi et d'affichage des informations utilisateurs.

Un fois l'application téléchargée, l'utilisateur entre ses informations pour se connecter. S'il n'est pas adhérent, il se connecte en invité.

Ensuite, il accèdera à une interface avec trois onglets. Le premier lui permettra de reserver une place, et voir ses dernières réservations. On pourra imaginer un système de réservation automatique s'il l'utilisateur vient au parking aux mêmes périodes successivement. Le deuxième fera fonctionner le flash de son téléphone et le récepteur Li-Fi lorsqu'il sera devant la borne d'entrée du parking : un étage, une allée et un numéro de place lui seront indiqués. Le dernier onglet lui affichera ses informations personnelles et les réglages de l'application.

L'invité aura seulement accès au deuxième onglet, le système lui créera automatiquement une réservation sur un compte provisoire et lui attribuera une place ainsi qu'un temps de stationnement.

Une notification peut être envoyée à l'utilisateur quand sa réservation va expirer.



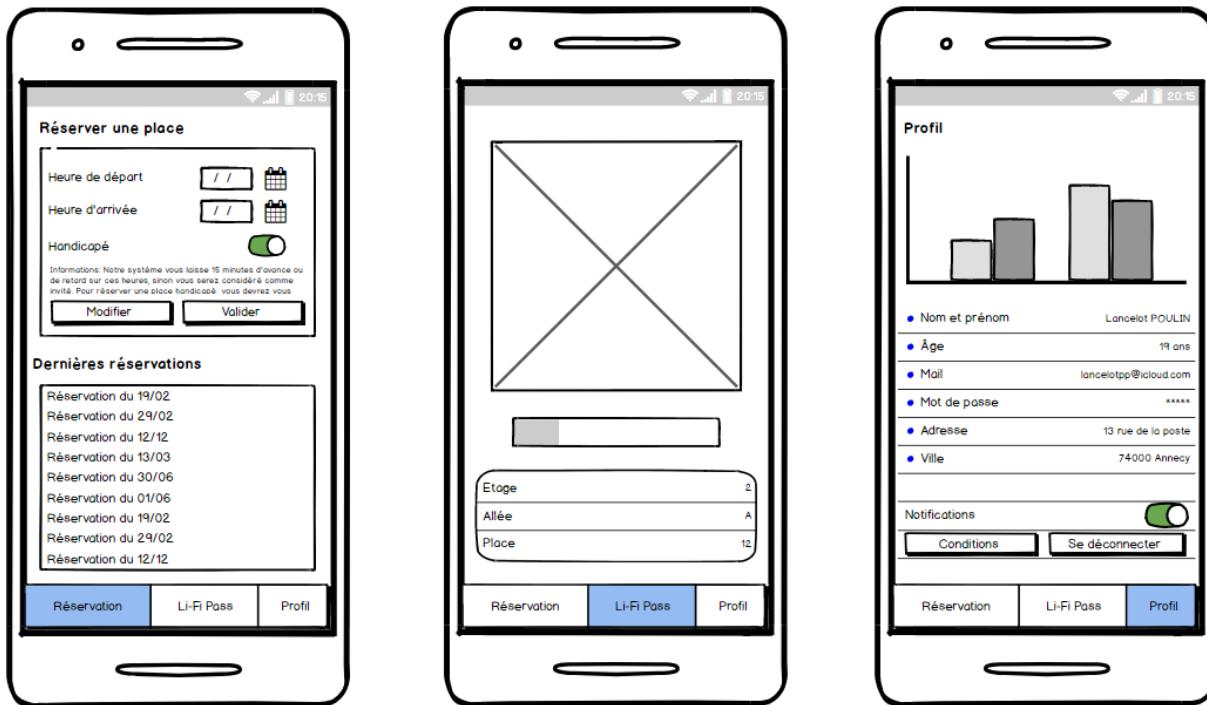
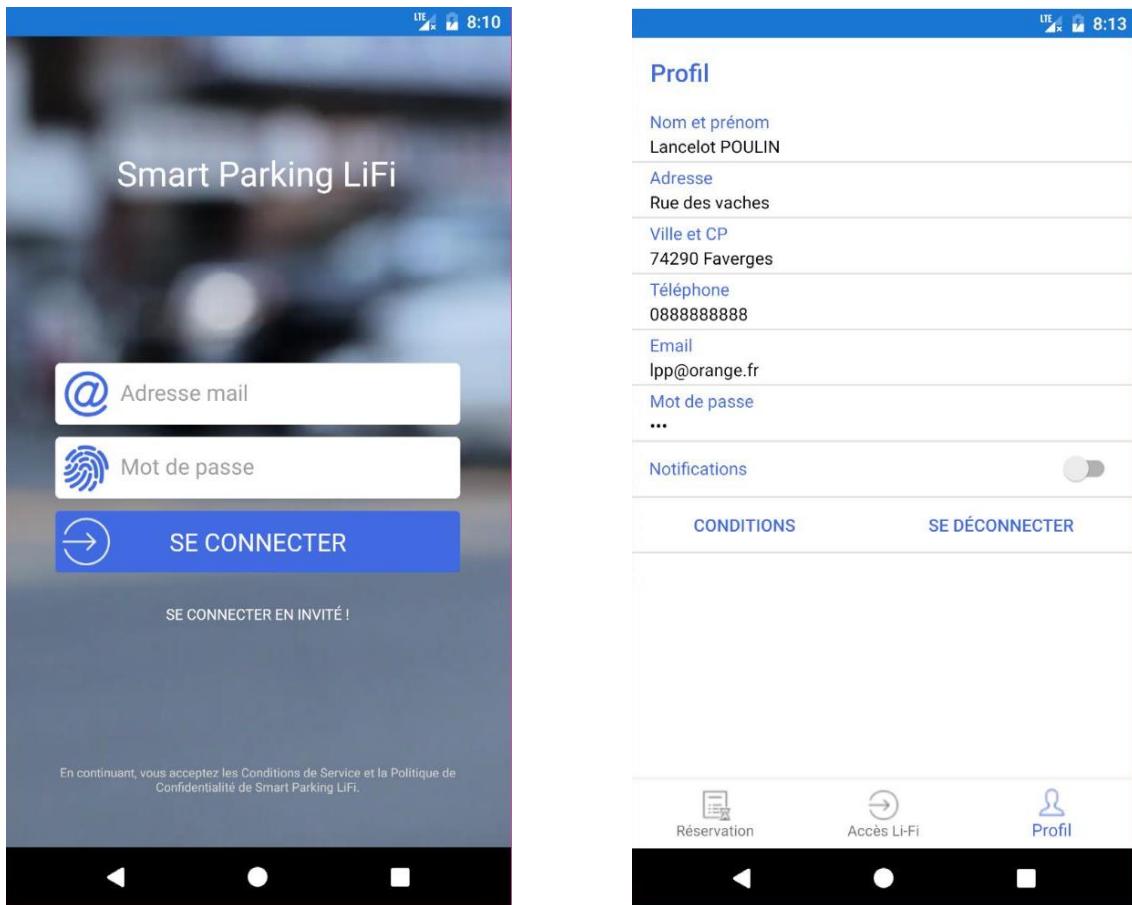


Figure 29: Maquette de l'application

Voici les pages de l'application réalisés en langage XAML avec Xamarin.Forms :



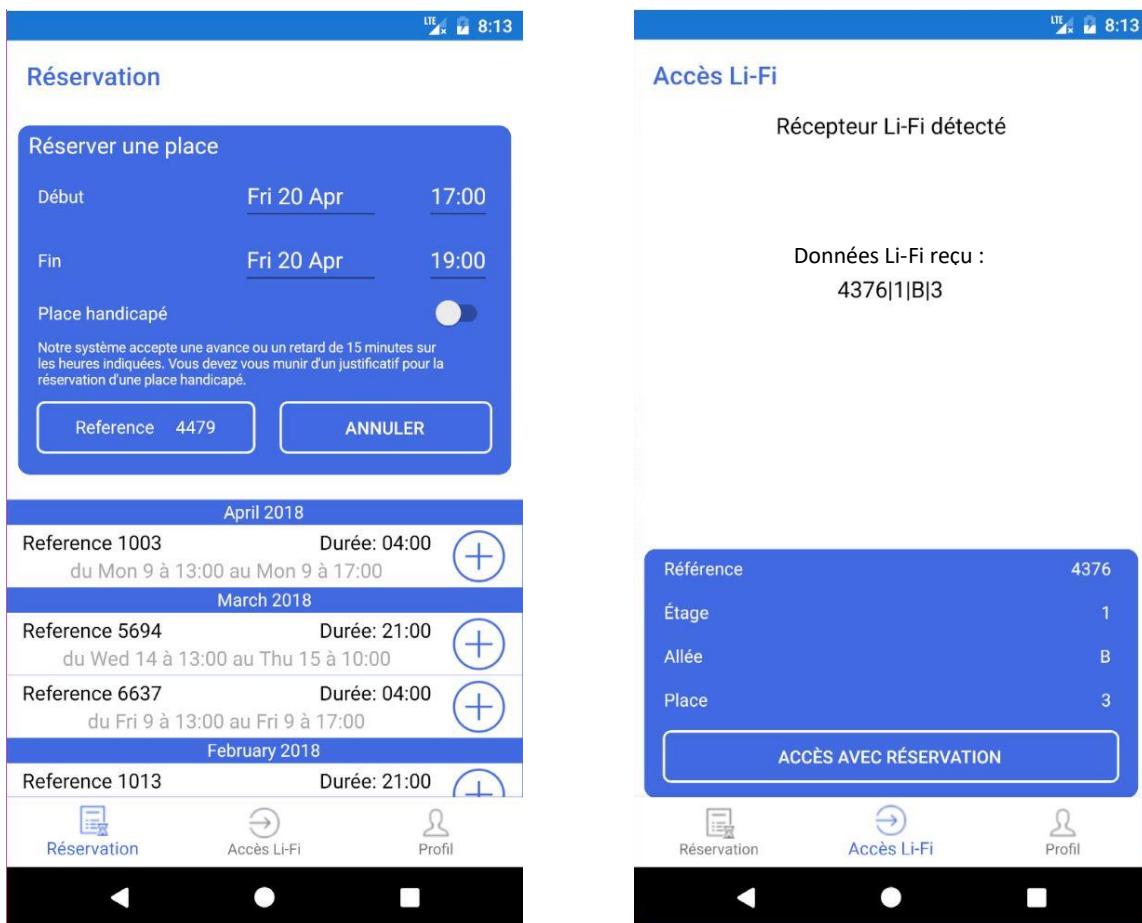


Figure 30: Interface graphique de l'application

L'adhérent peut aussi accéder au parking sans réservation comme un invité, en indiquant une durée de stationnement.

3.1.3.6 Ajout d'un fichier de configuration

Grâce au package disponible sur NuGet, on peut créer des fichiers dans le répertoire d'installation de l'application. Cette bibliothèque est PCL (Portable Class Library) c'est-à-dire qu'elle fonctionne sur toutes les plateformes .NET.

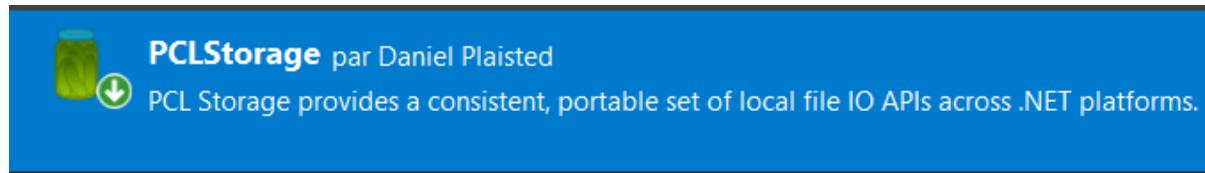


Figure 31: Package NuGet PCLStorage

Une fois le téléchargement et l'ajout de PCLStorage au projet, j'ai créé une classe qui va écrire dans un fichier de propriété les informations de l'utilisateur lorsqu'il va se connecter à l'application.

Ça permettra la sauvegarde de son mail et mot de passe quand l'application sera éteinte. Ces informations seront lues à chaque fois que l'application sera lancée pour une authentification automatique.

Voici la classe C# permettant d'écrire dans le fichier de configuration :

```

1  using PCLStorage;
2  using System.Threading.Tasks;
3
4  namespace PLSport.Data
5  {
6      // File to keep in memory user log when application is off
7
8      public static class Settings
9      {
10         public static async Task<string[]> Read() // Read informations when application start
11         {
12             return (await (await FileSystem.Current.LocalStorage.CreateFileAsync("Settings.txt",
13                 CreationCollisionOption.OpenIfExists)).ReadAllTextAsync()).Split('\n');
14         }
15
16         public static async Task Write(string Mail, string Password)
17         {
18             if (Mail == null && Password == null) // Delete informations when user disconnect
19             {
20                 await (await FileSystem.Current.LocalStorage.CreateFileAsync("Settings.txt",
21                     CreationCollisionOption.OpenIfExists)).WriteAllTextAsync("");
22             }
23             else // Write informations when user log in
24             {
25                 await (await FileSystem.Current.LocalStorage.CreateFileAsync("Settings.txt",
26                     CreationCollisionOption.OpenIfExists)).WriteAllTextAsync($"{Mail}\n{Password}");
27             }
28         }
29     }
30 }
```

3.1.3.7 Requête au web service

Pour communiquer avec une base de données, l'option la plus sécurisée reste le web service, qui va s'occuper d'identifier les personnes effectuant des requêtes à la base de données, celle-ci ne sera donc pas directement accessible depuis une machine extérieure.

Depuis une application Xamarin, on utilise le plus souvent des requêtes HTTP pour l'échange de données avec un web service. Voici le code pour récupérer les informations de l'utilisateur lorsqu'il se connecte :

```

try // Requête HTTP de type GET, on insère les informations de connexion dans l'URL
{
    HttpResponseMessage Response = await App.Client.GetAsync($"http://{App.ServerAddress}/WebService/Login.php?Mail={Mail}&Password={Password}");
    if (Response.IsSuccessStatusCode) // Succès: Le mail et mot de passe sont correct
    {
        string JsonData = await Response.Content.ReadAsStringAsync(); // On récupère les données Json
        var Users = JsonConvert.DeserializeObject<IEnumerable<Data.Model.User>>(JsonData); // On déserialise dans une classe modèle à la BDD
        foreach (var User in Users) // On accède à l'objet utilisateur
        {
            await Data.Settings.Write(User.Mail, User.Password); // On écrit les informations de l'utilisateur dans le fichier de configuration
            await Navigation.PushAsync(new Tabbed MainPage(User)); // On accède à la page d'onglet réservation/Li-Fi
        }
    }
    else { DisplayAlert("Erreur", "Mail ou mot de passe incorrect, veuillez réessayer.", "OK"); } // Message d'erreur
}
catch { DisplayAlert("Erreur", "Aucune réponse du serveur. Veuillez réessayer ultérieurement.", "OK"); } // Message d'erreur

```

Figure 32: Requête HTTP en C#

On effectue une requête GET sur le fichier LogIn.php du web service avec comme argument le mail et le mot de passe de l'utilisateur. Si le serveur nous répond avec un statut succès alors il a trouvé l'utilisateur correspondant, sinon on affiche un message d'erreur. Ensuite on déserialise les données encodées en Json dans un objet Utilisateur, qui est une classe modèle de la table Client de la base de

données (disponible en annexe). Enfin, on écrit les informations de l'utilisateur dans le fichier de propriété et on accède aux pages à onglets vus plus haut.

Toutes les requêtes HTTP vers le web service requises au fonctionnement de l'application sont disponible en annexe ainsi que les classes C# modèle correspondant à la base de données.

Voici un exemple :

```

1  // Reservation database model
2
3  using System;
4
5  namespace SmartParkingLiFi.Data.Model
6  {
7      public class Reservation
8      {
9          public int ID { get; set; }
10         public string Reference { get; set; }
11         public DateTime Debut_Reservation { get; set; }
12         public DateTime Fin_Reservation { get; set; }
13         public DateTime Arrivee { get; set; }
14         public DateTime Depart { get; set; }
15         public int Place_ID { get; set; }
16         public int Client_ID { get; set; }
17     }
18 }
```

3.1.3.8 Activité principale de l'application

L'application smartphone doit gérer les différents statuts de l'utilisateur, j'ai donc réalisé un algorithme utilisant quatre statuts possibles : (en cours signifie que le client est dans le parking)

- « None » où l'utilisateur n'a pas de réservation validée ou en cours
- « Validate » où l'utilisateur a une réservation validée mais pas en cours
- « InProgress » l'utilisateur a une réservation en cours
- « Guest » où l'utilisateur est un invité (sa réservation est en cours ou non)

Cela est traduit par une énumération C# `enum ReservationStatus { None, Validate, InProgress, Guest };`

Ces statuts vont permettre à l'utilisateur l'accès ou la restriction de certaine fonctionnalité de l'application (valider/annuler/modifier une réservation, accès avec/sans réservation...). Nous savons si l'utilisateur est un invité dès lors qu'il se connecte en tant qu'inviter. Pour l'adhérent, j'ai créé un algorithme pour savoir son statut actuel : c'est l'activité principale de l'application (en thread). Il va ainsi pourvoir réguler l'utilisateur lors de l'utilisation de l'application lorsque le temps passe (car ici le temps est très important avec les réservations). C'est pourquoi l'algorithme s'exécute toute les 5 secondes dans l'application pour la mettre à jour des derniers événements de la BDD et de l'heure.

On traite de cette manière les différents cas possibles : Par exemple, l'utilisateur ne peut pas annuler sa réservation lorsqu'il est dans le parking. Plus il y a de fonctionnalité, plus nous devons vérifier si cela n'interfère pas avec le bon fonctionnement du parking. Le programme est disponible en annexe.

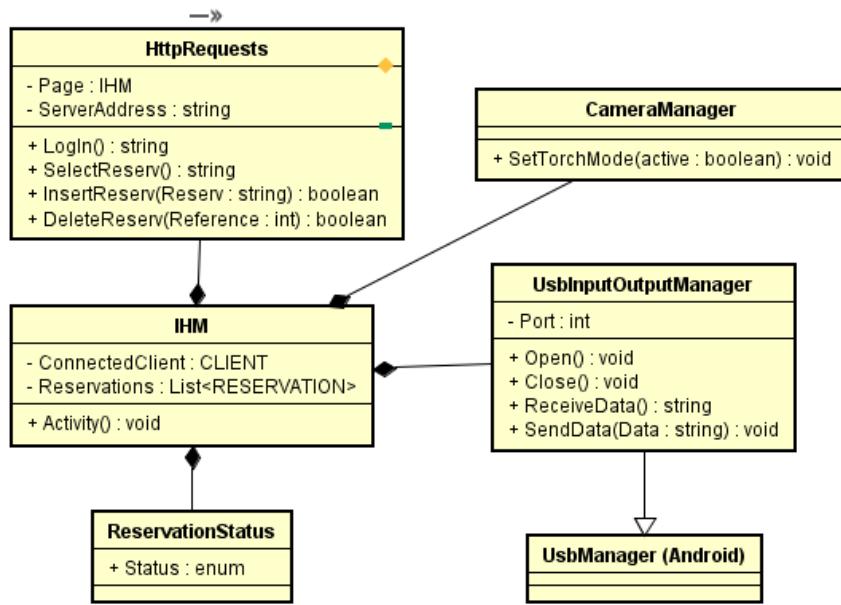


Figure 33: Diagramme de classe simplifié de l'IHM de l'application smartphone

3.1.3.9 Envoi de données Li-Fi depuis le smartphone

Afin d'envoyer la référence de réservation depuis le smartphone du client à la borne Li-Fi lors de l'accès au parking, j'ai utilisé le flash de la caméra grâce aux bibliothèques natives des plateformes Android et iOS. Comme le montre le code ci-dessous, j'ai créé un algorithme permettant d'allumer le flash pendant des durées définies correspondant aux chiffres de la référence à envoyés. Par exemple, pour envoyer 2688, le flash s'allume pendant 200ms, puis 600ms et enfin deux fois 800ms (Chiffre multiplié par 100ms, 1000ms pour le chiffre 0). Ces données dont encadrées par un temps de start et de stop de 50ms durant lesquels le flash est allumé.

Ici nous utilisons l'objet CameraManager qui permet l'accès aux fonctionnalités de la caméra. La méthode SetTorchMode(...) change l'état du flash (ON/OFF). J'ai dû choisir des délais assez longs pour remédier au manque de précision du flash de la caméra de certain appareil. Les tests sont réussis et la réception des données côté Raspberry pourra être vérifiée lors de la dernière tâche.

```

// Envoie de données via le flash de la caméra de smartphone
0 références
private void SendFlashLiFiData(string Data)
{
    CameraManager CameraFlashManager = AppCompatActivity.ApplicationContext.GetSystemService(Context.CAMERA_SERVICE) as CameraManager;

    CameraFlashManager.SetTorchMode(CameraFlashManager.GetCameraIdList()[0], true); // Start bit
    Thread.Sleep(50);
    CameraFlashManager.SetTorchMode(CameraFlashManager.GetCameraIdList()[0], false);
    Thread.Sleep(50);
    for (int i = 0; i < Data.Length; i++)
    {
        int NumberFlashDelay = Convert.ToInt32(Data.Substring(i, 1)) * 100; // 1 -> 100ms | 2 -> 200ms | ...
        if (NumberFlashDelay == 0) // 0 = 1000ms
            NumberFlashDelay = 1000;

        CameraFlashManager.SetTorchMode(CameraFlashManager.GetCameraIdList()[0], true); // On allume le flash pendant le temps calculé
        Thread.Sleep(NumberFlashDelay);
        CameraFlashManager.SetTorchMode(CameraFlashManager.GetCameraIdList()[0], false);

        Thread.Sleep(50); // Délai entre chaque nombre envoyé
    }
    CameraFlashManager.SetTorchMode(CameraFlashManager.GetCameraIdList()[0], true); // Stop bit
    Thread.Sleep(50);
    CameraFlashManager.SetTorchMode(CameraFlashManager.GetCameraIdList()[0], false);
}
  
```

Figure 34: Algorithme d'envoi de données avec le flash

3.1.3.10 Réception de données Li-Fi sur le smartphone

J'ai utilisé le récepteur Li-Fi micro-USB que l'on peut brancher sur smartphone pour recevoir des données Li-Fi. Il me fallait une bibliothèque de communication série comme lorsque l'on utilise le récepteur sur un ordinateur, vu précédemment. Je me suis servi d'un package NuGet « SerialInputOutputManager » qui hérite de la classe Android « UsbManager ». Elle permet de configurer et ouvrir un port COM et donc de récupérer les données reçues par le récepteur Li-Fi du smartphone.

Tout d'abord, nous vérifions que l'on trouve bien le récepteur Li-Fi dans les drivers branchés au téléphone, puis nous configurons le port avec la même configuration que le port UART de la Raspberry. Ensuite on indique les actions à effectuer lors de la réception de données ou erreurs rencontrés, et enfin on ouvre le port COM.

L'utilisateur doit nous donner sa permission pour accéder au récepteur (port USB de l'appareil). Le code commenté est à retrouver en annexe.

Lorsque l'on place le smartphone sous la lampe Li-Fi envoyant la référence, le numéro, l'allée et l'étage de la place, nous recevons bien les données émises.

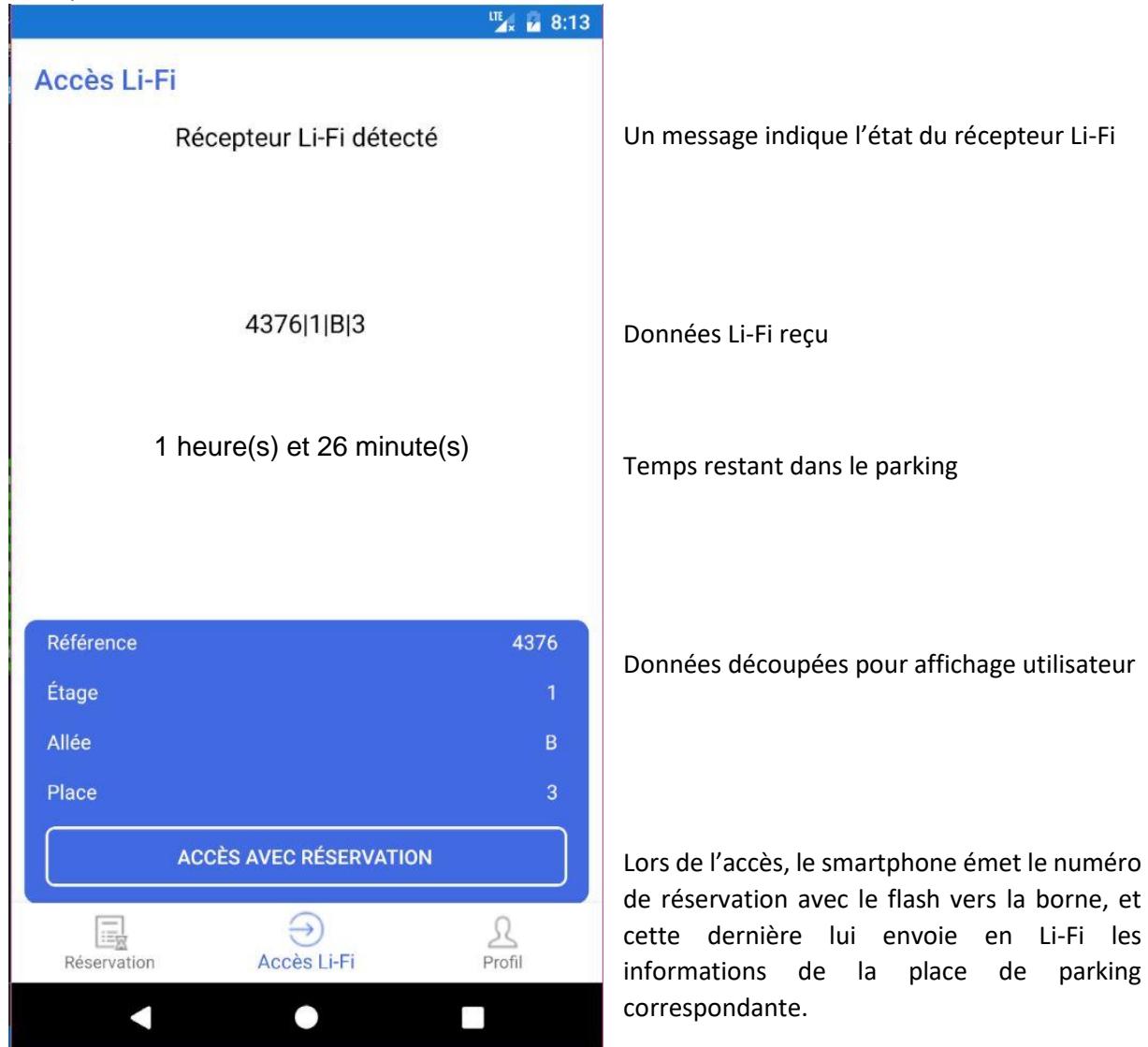


Figure 35: Données Li-Fi reçus sur le smartphone

3.1.3.11 Réception de donnée Li-Fi sur la borne Li-Fi

Pour cette dernière sous-partie il me restait à faire la réception des données émises par le smartphone, soit la référence de réservation, sur la borne Li-Fi. J'ai décidé d'utiliser une photorésistance (LDR) afin de détecter la lumière et donc le flash de la caméra du smartphone. Puis il me suffisait de calculer les temps où le flash est allumé.

Pour le fonctionnement de la LDR, je n'utiliserais pas de convertisseur analogique-numérique (la Raspberry n'est pas équipée d'entrée/sortie analogique) pour récupérer la valeur de la photorésistance mais d'un condensateur : Ce dernier permet de se charger et de se décharger pendant une durée dépendant de la résistance R en ohms (ici, la photorésistance) et de sa capacité C en farads.

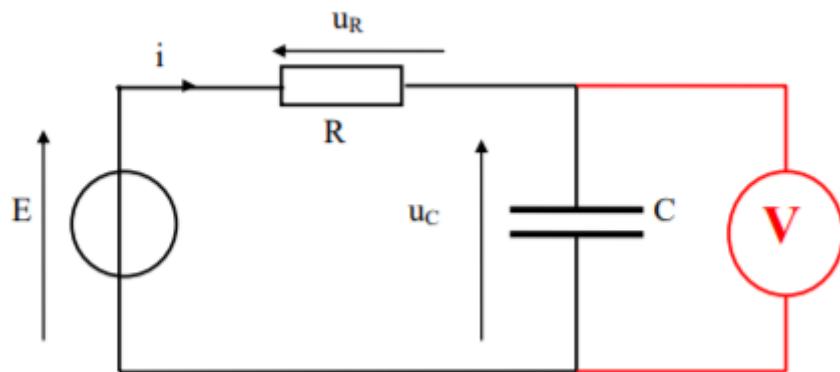


Figure 36: Montage pour le relevé de la courbe de charge d'un condensateur

La courbe de charge d'un condensateur est une exponentielle.

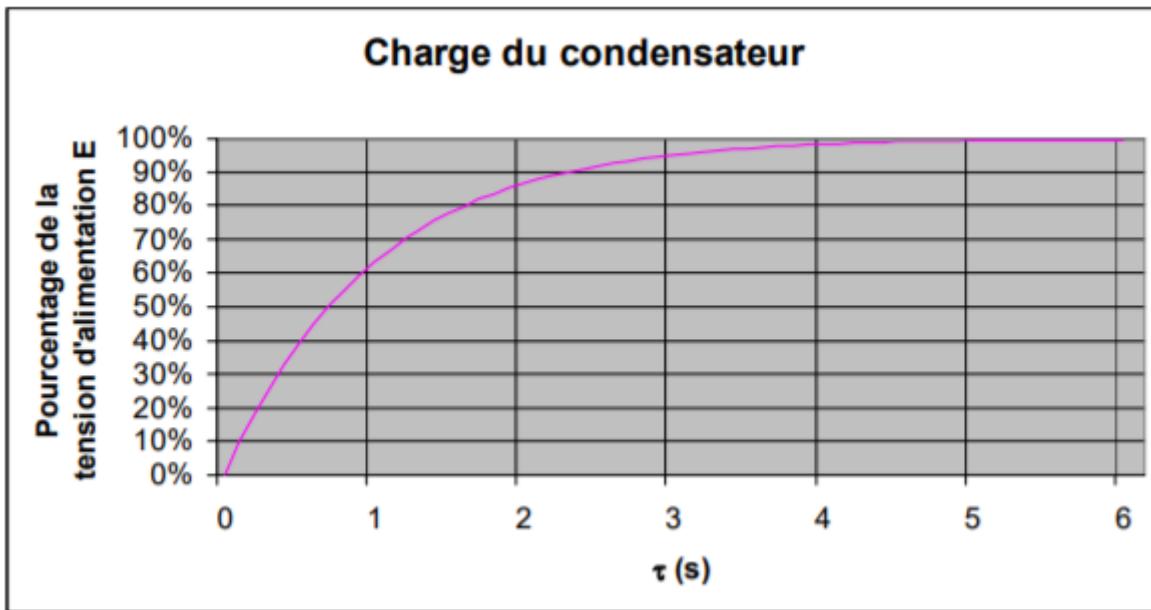


Figure 37: Courbe de charge d'un condensateur

Cette durée de charge est la fonction du produit $R \cdot C$ appelé constante de temps du circuit : $\tau = R \cdot C$
Par exemple pour $R = 10 \text{ k}\Omega$ et $C = 1000 \mu\text{F}$ on a : $\tau = Rn \cdot C = (10 \cdot 10^3) \cdot (1000 \cdot 10^{-6})$ soit $\tau = 10 \text{ s}$.

Plus cette constante de temps τ est grande, plus la charge du condensateur est lente, donc si la résistance est faible, alors le temps de charge sera faible. Voici le rapport entre la résistance et la luminosité de la LDR.

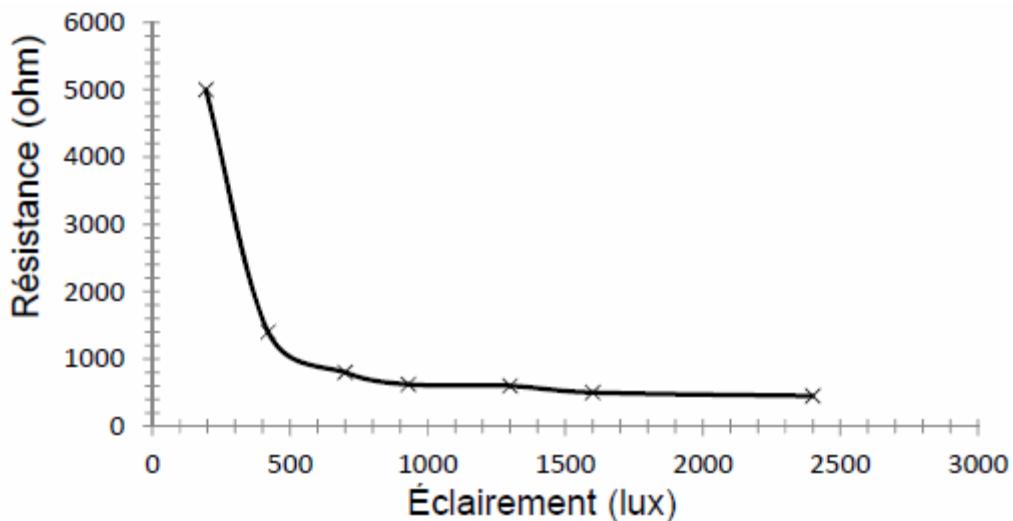


Figure 38: Courbe du rapport résistance/éclairement de la LDR

On peut en déduire que si τ est faible, alors la luminosité L est grande, et si τ est grand, alors L est faible. Grâce aux câblage effectué sur la Raspberry ci-dessous ainsi que quelques lignes de code, je vais pouvoir obtenir la constante de temps du circuit. J'utilise un condensateur de 10 millifarads.

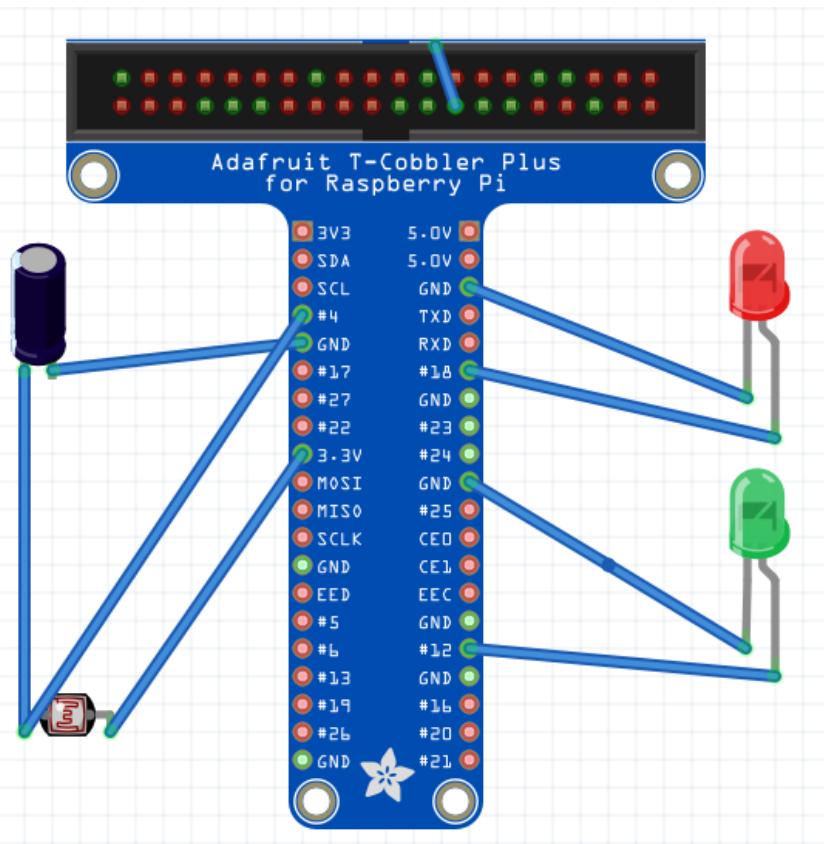


Figure 39: Câblage de la photorésistance sur la Raspberry

Le câblage a été effectué sur Fritzing qui est un logiciel gratuit permettant de faire des schémas de câblage et proposant beaucoup de composants qui sont créés par des utilisateurs sous forme de plug-in



Pour le code, on utilise l'horloge du processeur de la Raspberry : On sauvegarde l'heure lorsque l'on met à l'état 0 la broche reliée au condensateur en la mettant en sortie, puis on attend que cette broche soit à l'état 1 en la changer en entrée. Ce temps entre l'état 0 et l'état 1 calculé avec la soustraction de la nouvelle heure et l'heure sauvegardé correspond au temps de charge du condensateur, c'est la première partie du programme.

```

284     string Data = "";
285     clock_t LDRClock = clock();
286     clock_t LightClock = clock();
287     bool IsLightDetected = false;
288
289     while (Data.length() < 4) // Référence à 4 chiffres = on attend 4 valeurs
290     {
291         pinMode(LDR_PIN, OUTPUT); // Mise à l'état LOW
292         digitalWrite(LDR_PIN, LOW);
293         delayMicroseconds(80);
294
295         pinMode(LDR_PIN, INPUT); // On remet la PIN en entrée
296         delayMicroseconds(20);
297
298         LDRClock = clock();
299         while (digitalRead(LDR_PIN) == LOW) { } // On attend que la PIN soit HIGH = condensateur déchargé
300         double LDRDuration = (clock() - LDRClock); // Temps de charge
301         // printf("%f \n", LDRDuration); // Pour voir les temps de charge
302
303         if (LDRDuration < LIGHT_DETECTION_CONST) // Si flash détecté, on lance le chrono ou on le continue
304         {
305             if (!IsLightDetected)
306             {
307                 LightClock = clock();
308                 IsLightDetected = true;
309             }
310         }
311         else // Si plus de flash détecté, on calcul la valeur avec la durée du chrono
312         {
313             if (IsLightDetected) // ms -> valeur (100ms = 1, 200ms = 2)
314             {
315                 int LightDuration = (int)((clock() - LightClock) / (double) CLOCKS_PER_SEC) * 10;
316                 if (LightDuration == 10) { LightDuration = 0; } // 100ms --> Valeur 0
317                 if (!(Data.length() == 0 && LightDuration == 0)) // Ajoute la valeur à la référence
318                 {
319                     Data += NumberToString(LightDuration);
320                 }
321                 IsLightDetected = false;
322             }
323         }
324     }
325     return Data;
326 }
```

Figure 40: Programme pour la détection du flash du smartphone

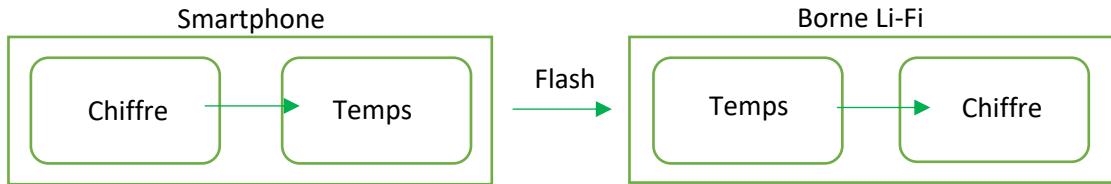
Voici les valeurs de temps récupérées lors de l'exécution du programme :

Test_LiFi	Test_LiFi
5942.000000	1598.000000
5140.000000	1487.000000
5454.000000	1489.000000
5499.000000	1522.000000
5218.000000	1677.000000
5383.000000	1474.000000
5432.000000	1501.000000
5666.000000	1394.000000
5360.000000	1618.000000
4639.000000	1337.000000
5255.000000	1579.000000
6109.000000	1527.000000
5241.000000	1663.000000
5242.000000	1426.000000
5677.000000	1507.000000
5430.000000	1542.000000
5548.000000	1709.000000
5232.000000	1405.000000
5795.000000	1609.000000
5185.000000	2077.000000
5710.000000	1603.000000
5581.000000	1442.000000
4698.000000	1591.000000
	1591.000000

Figure 41: Console affichant les temps de charge du condensateur

On obtient un temps entre 4000 et 5000 microsecondes à luminosité ambiante et proche de 1500 microseconde lorsque l'on pointe le flash du téléphone vers la photorésistance. J'en ai conclu que

lorsque l'on passe en dessous de 3000, on détecte un flash de smartphone. La seconde partie de programme calcul le temps d'allumage du flash : Lors de la détection du flash, on lance un chrono en sauvegardant l'horloge de la Raspberry, puis on fait la différence de temps T1 - TO comme précédemment pour connaître la durée d'émission du flash. On convertit ce temps (ms) en valeur (chiffre) quatre fois pour obtenir le nombre de la référence à quatre chiffres.



3.1.3.12 Communication à la base de données depuis la borne Li-Fi

Pour la communication avec la base de données, j'utilise la bibliothèque C++ « `cppconn` » qui est un package installable avec « `apt-get` ». La fonction permettant la connexion avec les informations de notre serveur :

```

#include <cppconn/driver.h>
#include <cppconn/exception.h>
#include <cppconn/resultset.h>
#include <cppconn/statement.h>

// Connection à la BDD
bool ConnectToDatabase()
{
    try
    {
        // Etape 1 : créer une connexion à la BDD
        driver = get_driver_instance();
        con = driver->connect("mysql-projetlifi.alwaysdata.net", "156739", "projetlifi");

        // Etape 2 : connexion à la base choisie, ici olivier_db
        con->setSchema("projetlifi_bdd");
        stmt = con->createStatement();
        return true;
    }
    catch (sql::SQLException &e)
    {
        // Gestion des exceptions pour débogage
        cout << "# ERR: " << e.what();
        cout << " (code erreur MySQL: " << e.getErrorCode();
        cout << ", EtatSQL: " << e.getSQLState() << " ) " << endl;
    }
    return false;
}

```

Si la connexion est un succès, alors on peut effectuer des requêtes avec l'objet « `statement` » du connecteur MySQL. Pour insérer une référence invitée :

```

stmt->execute("INSERT INTO reservation VALUES (NULL, " + Reference + ", " + BeginDate + ", " + EndDate + ", NULL, NULL, " + PlaceID + ", 1)");

```

J'obtiens la date et l'heure actuelle de la Raspberry grâce à la fonction `clock()` de la bibliothèque `clock.h`. Voici comment je récupère la date de début de réservation ainsi que la date de fin (soit la date actuelle + 2 heures) pour la création d'une réservation invitée :

```

// Génère les dates de réservation invitée lors de la création (Date actuelle jusqu'à Date actuelle + 2 heures) (Refactorisation)
void GenerateReservationDateTimes(string &BeginDate, string &EndDate)
{
    time_t current; struct tm datetime; char format[32];
    time(&current); datetime = *localtime(&current); strftime(format, 32, "%Y-%m-%d %H:%M:%S", &datetime);
    string Convert1(format); BeginDate = Convert1;
    current += 60 * 60 * 2; datetime = *localtime(&current); strftime(format, 32, "%Y-%m-%d %H:%M:%S", &datetime); // DateTime + 2 heures
    string Convert2(format); EndDate = Convert2;
}

```

J'ajoute 60 secondes multipliées par 60 minutes multipliées par 2 heures. Je les convertis en string (chaîne de caractère) pour la requête.

3.1.3.13 Boucle principale de l'application de la borne Li-Fi

Il ne me reste plus qu'à créer l'algorithme qui va permettre à la borne Li-Fi d'accueillir les clients lors de l'accès. Voici les étapes qui se répèteront à l'arrivée d'une personne au parking :

1. Initialisation/configuration port UART et connexion à la base de données (non répété)
2. Allume la LED rouge et éteint la LED verte
3. Attente d'un client : détection d'un flash de téléphone
4. Réception de la référence du client
5. On récupère une place disponible dans le parking
6. Si c'est un invité (Référence 1000)
 - a. On crée un référence unique non existante dans la BDD
 - b. On instancie un intervalle de temps de 2 heures (début/fin de la réservation)
 - c. On crée la réservation (insertion BDD) avec les heures et la place attribuée
 - d. On envoie la référence du client ainsi que sa place de parking en Li-Fi
 - e. Le client peut aller se garer et sa référence lui permet de voir son temps restant
7. Si c'est un adhérent (Référence 1001 à 9999 données lors de la validation sur l'application)
 - a. On récupère les informations de sa réservation depuis la BDD
 - b. On met à jour sa réservation dans la BDD avec la place attribuée
 - c. On envoie la place de parking attribuée en Li-Fi
8. Si les étapes 6 et 7 se passe correctement, on allume la LED verte et on éteint la LED rouge
9. Sinon on affiche l'erreur (exemple : plus de place disponible pour les invités)
10. On reboucle à l'étape 2

Je rappelle qu'il y a toujours une place pour un adhérent qui a réservé mais pas pour un invité. Un invité obtient une réservation de deux heures. C'est l'application qui gère si l'utilisateur est en avance sur son début de réservation et donc s'il peut accéder au parking où s'il doit partir en cas de retard sur l'heure de fin de réservation. Le programme commenté est disponible en annexe.

Lorsque l'on envoie la référence de réservation avec le flash du téléphone, on obtient les différentes possibilités sur la console du programme :

```
Reference: 7130
Erreur: Reference inexistante

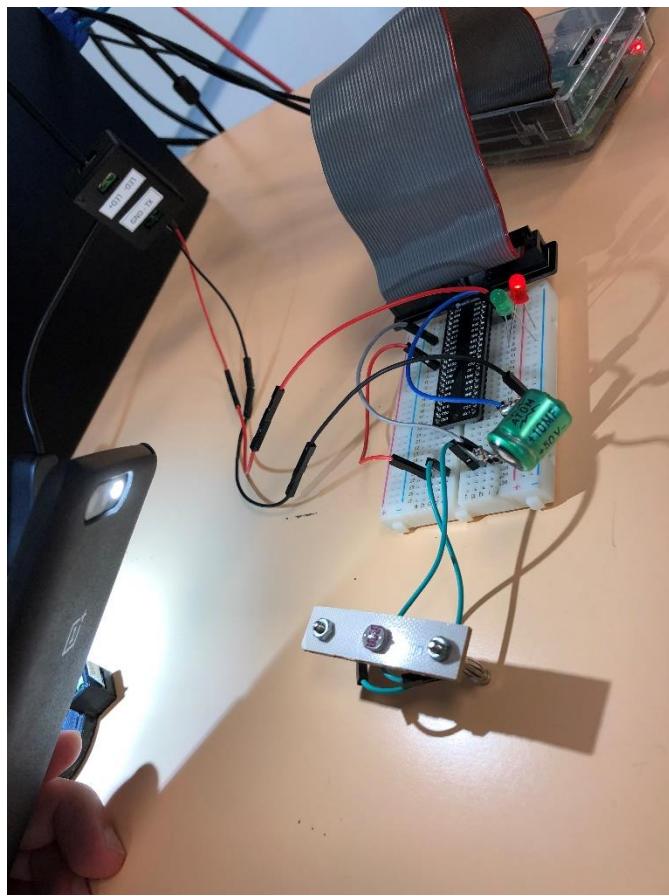
Reference: 1000
Envoi LiFi en cours: 2426|1|B|2
Fin Envoi LiFi
```

Quand la référence envoyée est inexistence ou erronée on peut voir qu'il y a une erreur. La référence 1000 est un accès invité, la borne renvoie donc la référence ainsi que la place attribuée au téléphone via Li-Fi.

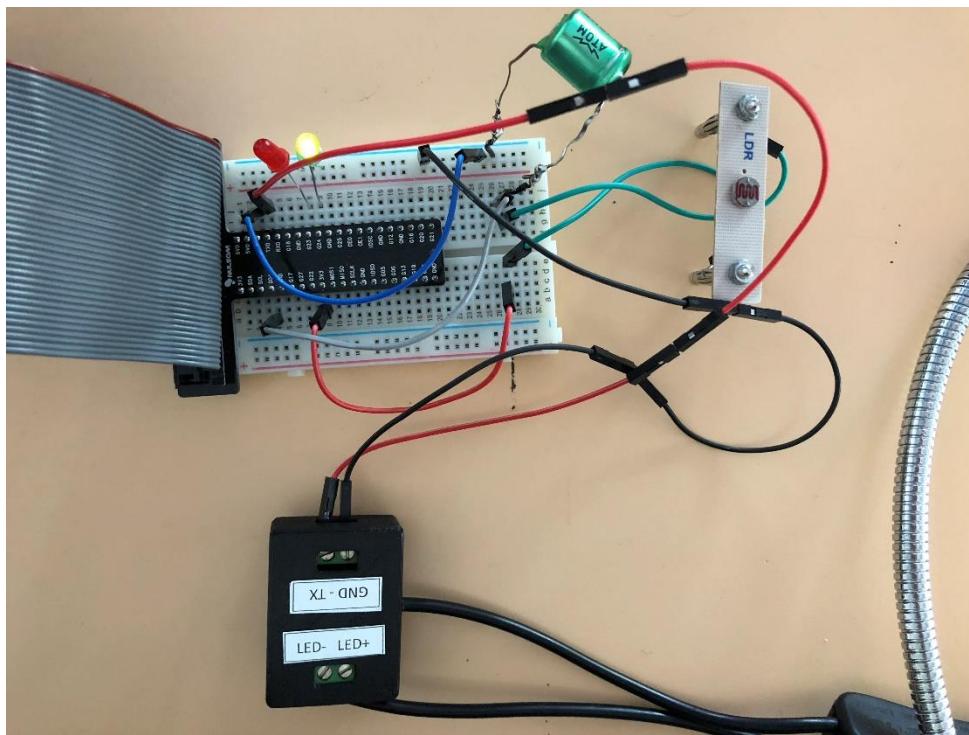
```
Reference: 1010
Envoi LiFi en cours: 1010|1|B|2
Fin Envoi LiFi
```

La référence 1010 étant une réservation validée par un adhérent, la borne renvoie donc la même référence et un numéro de place.

On dirige le flash



La LED passe au vert



3.1.4 Conclusion

Malgré les quelques difficultés que j'ai eues pour comprendre et convertir la bibliothèque Li-Fi, ce que je n'avais jamais fait auparavant, je n'ai pas pris de retard grâce à mon expérience avec Xamarin et Raspberry. Mes tâches présentées dans le diagramme de Gantt ont été bien respectées et l'application smartphone et le programme de la borne Li-Fi sont terminés.

Les tests de réservation avec le web service sont validés. La réception et l'émission de données Li-Fi entre la borne et le smartphone étant fonctionnel, l'utilisateur peut réserver et accéder au parking avec son smartphone, en tant qu'adhérent et invité.

La partie qu'on m'a donnée était assez vague et j'ai dû prendre des décisions pour utiliser les données par la lumière en bidirectionnelles avec le smartphone et la Raspberry. Je me suis beaucoup aidé de ressources web pour l'envoi de référence par le smartphone avec le flash (qu'on ne peut pas formellement appeler « Li-Fi »)

On peut opter pour des améliorations comme l'intégration d'une barrière piloté par un servomoteur à l'entrée du parking ou encore la récurrence de réservation pour un utilisateur (réservation automatique chaque jour).

Travailler sur une nouvelle technologie comme le Li-Fi est très intéressant, avec une évolution ces dernières années et une démocratisation dans les prochaines, j'ai eu la chance d'avoir effectué une réelle approche dans un contexte professionnel.

3.2 Partie SURVEILLER l'état du parking – carte embarquée

Partie effectuée par Quentin Mermaz

3.2.1 Spécifications

Ma partie consiste à surveiller chaque place d'un parking à l'aide de capteur à ultrasons et de sauvegarder ces informations sur une base de données, mais je dois également communiquer en socket berkeley avec le poste de supervision pour qu'il puisse vérifier le bon fonctionnement de l'installation.

Ci-dessous un schéma représentant ma partie ainsi que celles avec qui je communique :

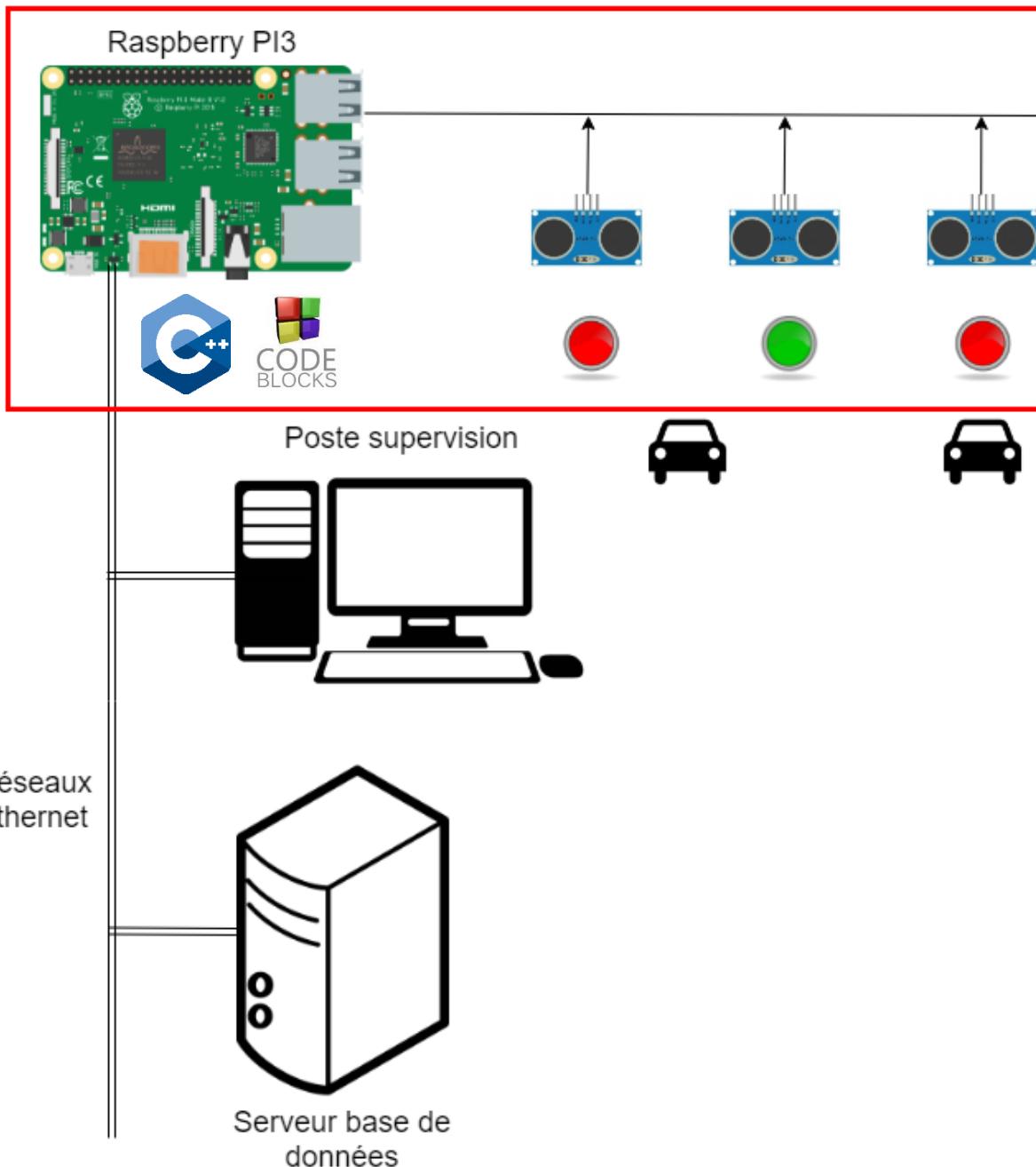


Figure 42: Structure de la partie SURVEILLER parking

3.2.2 Synoptique de l'architecture matérielle et logicielle

Pour réaliser ma partie j'ai comme matériel à disposition :

- Une Raspberry PI 3, c'est un mini-ordinateur de la taille d'une carte de crédit qui permet de connecter des appareils (ici, des capteurs d'ultrasons) et ainsi automatiser des relevés de mesure par exemple. Elle peut aussi servir de serveur pour héberger une base de données.



Figure 43: Une Raspberry PI 3

- Une carte microSD SanDisk de 32Go



Figure 44: Carte microSD SanDisk

- Des capteurs ultrasons, ils vont permettre de connaître la distance entre un objet et lui-même ce qui avec du code peut le transformer en capteur de présence.



Figure 45: Capteur à ultrasons HC-SR04

- Des LEDS qui indiqueront si la place est disponible



Figure 46: LEDS

Les logiciels et le système d'exploitation utilisés :

- Raspbian, c'est le système d'exploitation basé sur Debian qui a été créé spécialement pour Raspberry.

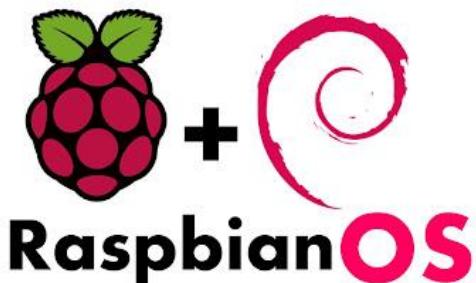


Figure 47: RaspbianOS

- Win32DiskImager, c'est un logiciel qui va permettre d'installer un système d'exploitation sur une carte SD pour ensuite la mettre dans une Raspberry.



Figure 48: Win32Diskimager

- Code::blocks est un environnement de développement libre qui permet de développer en C et C++

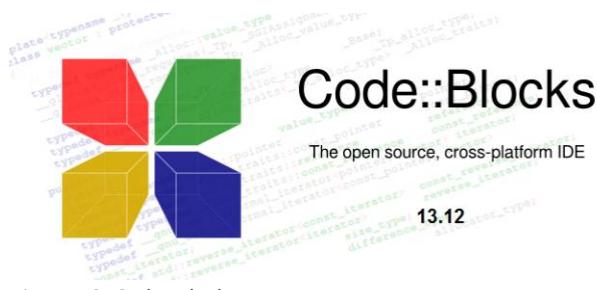


Figure 49: Code::Blocks

J'ai donc installé Raspbian sur la carte microSD, puis j'ai pu démarrer la Raspberry en ayant préalablement inséré la carte SD dedans et j'ai finalement configuré la Raspberry pour avoir un espace de travail opérationnel (Tutorial de l'installation de Raspbian jusqu'à l'installation de Code::Blocks en annexe).

3.2.3 Travail réalisé

De manière à réaliser un câblage électrique fonctionnel reliant le capteur à la Raspberry, j'ai dû étudier la documentation constructeur du capteur que vous pouvez consulter en annexe. Après cette étude j'ai réalisé le câblage que voici.

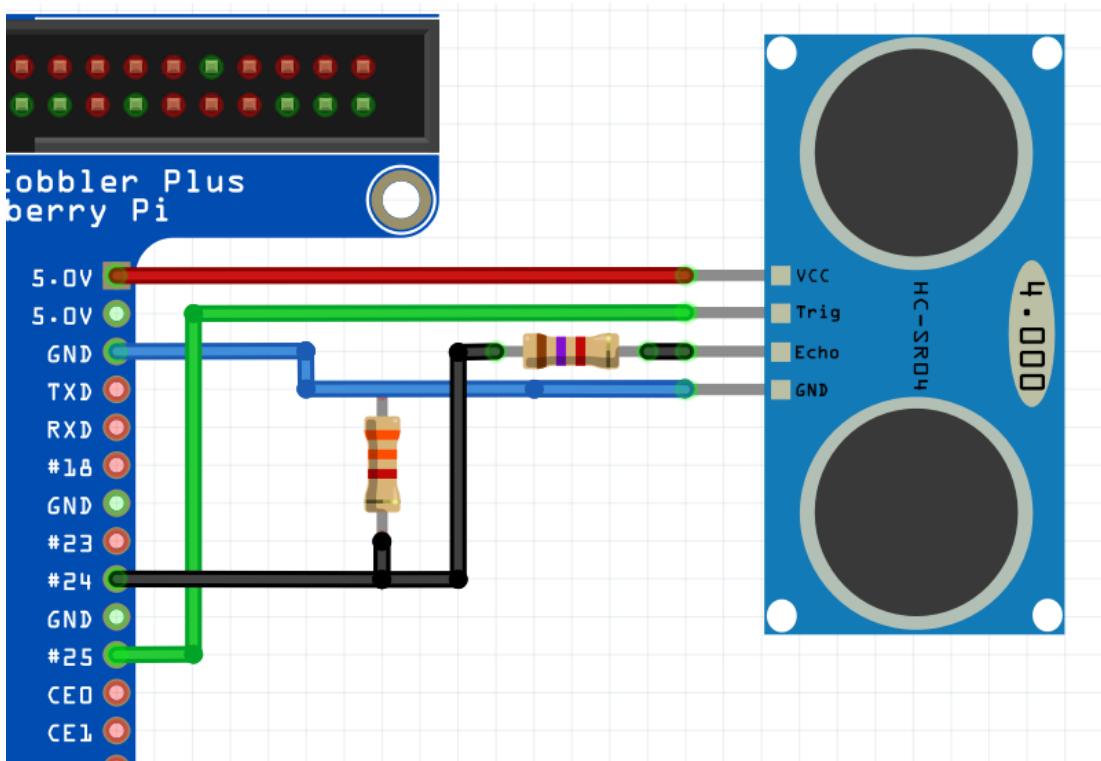


Figure 50: Câblage du capteur à ultrasons

Comme vous pouvez le constater sur le schéma ci-dessus, le capteur est alimenté en 5V par conséquent il envoie en retour du 5V dans la Raspberry, mais cette dernière accepte seulement du 3.3V, pour remédier à cela j'ai donc mis un pont diviseur en sortie du capteur qui me permet d'envoyer 3.3V dans la Raspberry au lieu de 5V (voir annexe).

Une fois mon câblage fait j'ai commencé à réfléchir à un programme qui pourrai le tester, pour cela je suis retourné dans la documentation constructeur pour étudier le fonctionnement du capteur. J'ai donc constaté qu'il fallait envoyer une impulsion de minimum 10 μ s puis le capteur va envoyer 8 ondes ultrasoniques à une fréquence de 40KHz, dès lors qu'il en reçoit une en retour, il coupe la mesure et envoi dans la Raspberry une impulsion proportionnelle à la distance qu'il a mesurée.

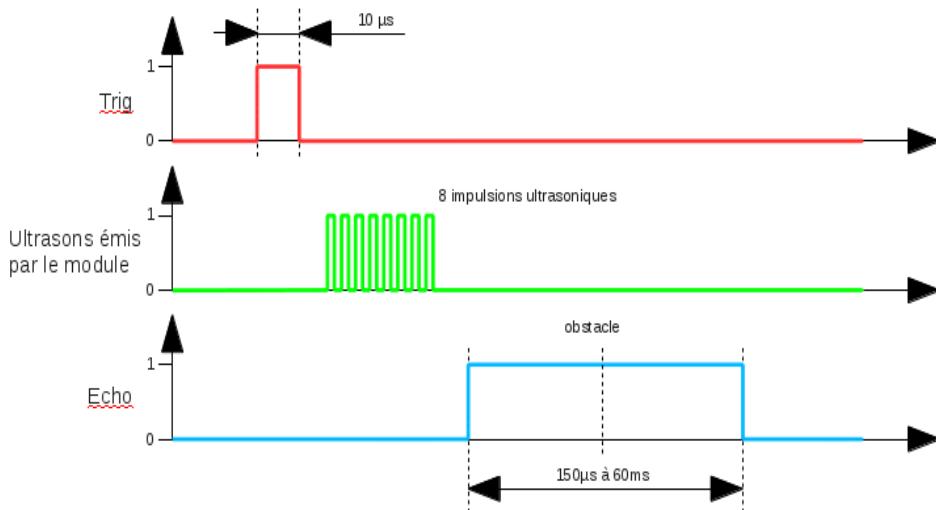


Figure 51: Ultrasons émis

Pour vérifier que le capteur fonctionne bien j'ai créé un programme qui envoie une impulsion dans le capteur, puis j'ai branché un oscilloscope sur l'entrée et la sortie du capteur, et j'ai réalisé plusieurs mesures à différentes distances d'un mur (des captures d'écran de l'oscilloscope sont disponibles en annexe).

Un exemple de trame commenté est présenté ci-dessous :

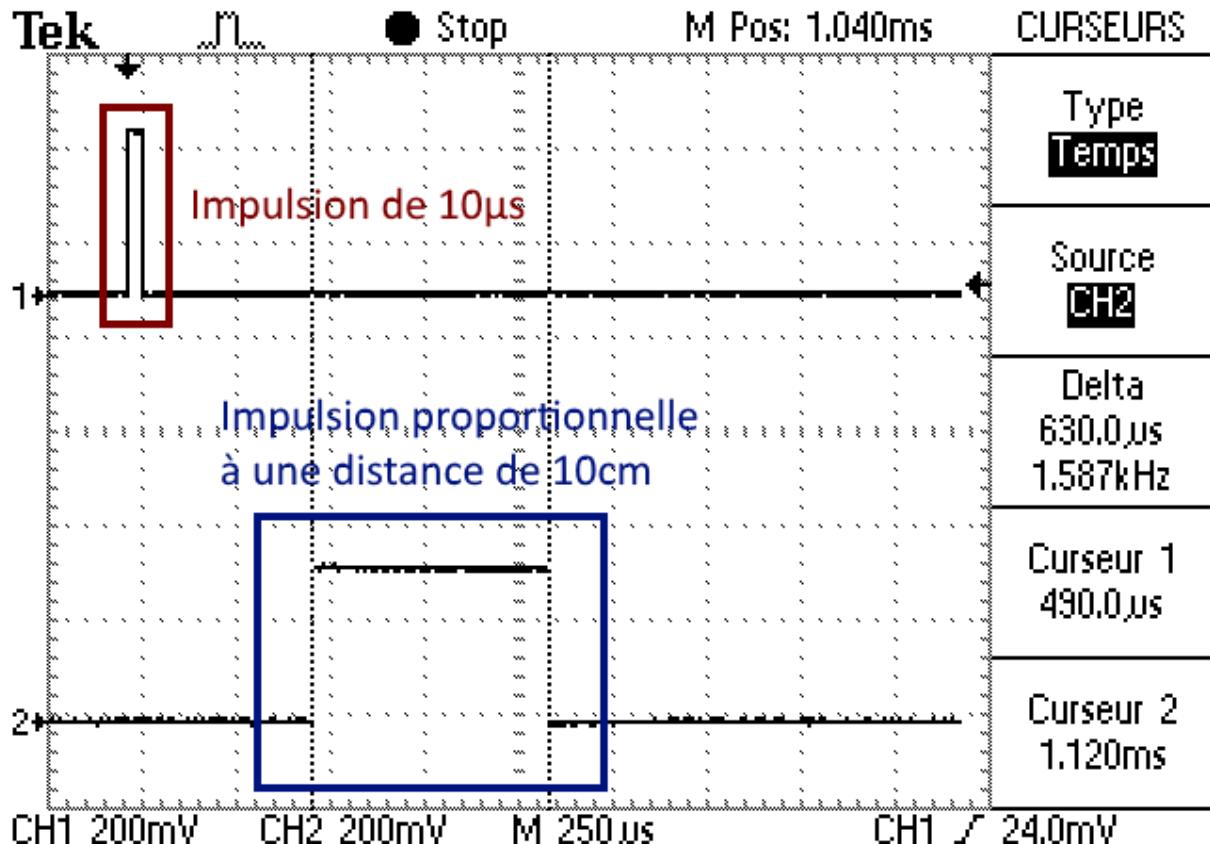


Figure 52: Trame envoyée par le capteur

On peut bien voir l'impulsion de 10 μ s en haut qui correspond à l'entrée du capteur et l'impulsion de 1,120ms en dessous qui correspond à la sortie du capteur, j'avais placé le capteur à 10cm d'un mur donc cette impulsion en sortie correspond à une distance de 10cm mais en réalité elle correspond à une distance de 20cm car l'onde effectue un aller et un retour, soit deux fois 10cm.

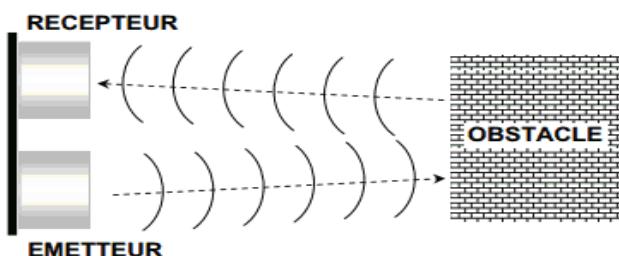


Figure 53: Principe du capteur d'obstacle

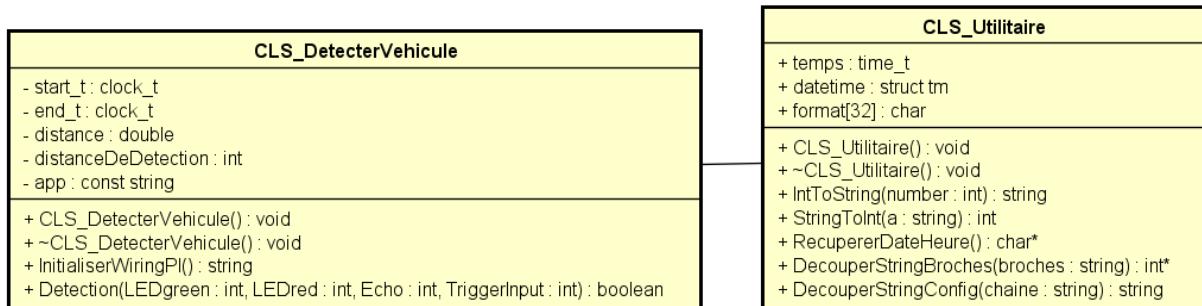
Une fois que le capteur a été testé et approuvé, il ne restait plus qu'à coder un programme qui calculait la distance entre le capteur et l'obstacle. Pour cela je me suis donc demandé comment je pourrai procéder, après réflexion je me suis servi de la formule « distance = vitesse du son * temps ». Sachant que la vitesse du son est de 340m/s dans l'air, il ne me restait plus qu'à connaître le temps que l'onde a mis pour parcourir la distance, pour cela il me suffisait de savoir combien de temps dur l'impulsion

que me renvoie le capteur et j'avais mon temps. J'ai donc créé un programme qui envoie une impulsion de $10\mu s$ sur l'entrée du capteur, puis tout de suite après il surveille la sortie du capteur et tant que le capteur n'envoie rien, le programme ne fait rien mais dès lors que le capteur commence à envoyer une impulsion le programme lance un chronomètre. A la fin de l'impulsion le programme arrête le chronomètre, applique la formule énoncée précédemment et enfin, il affiche le résultat comme ci-dessous. Le code est disponible en annexe.

```
Test_capteur
0.18513m
0.22066m
0.26384m
0.28628m
0.30447m
0.32198m
0.32691m
0.35105m
0.34986m
0.37349m
0.37944m
```

Figure 54: Résultat des distances calculées

Une fois tous les tests réalisés avec succès j'ai pu mettre au point le diagramme de classe pour détecter les véhicules, j'ai donc réfléchi et réalisé le diagramme de classe `CLS_DetecterVehicule` que voici :



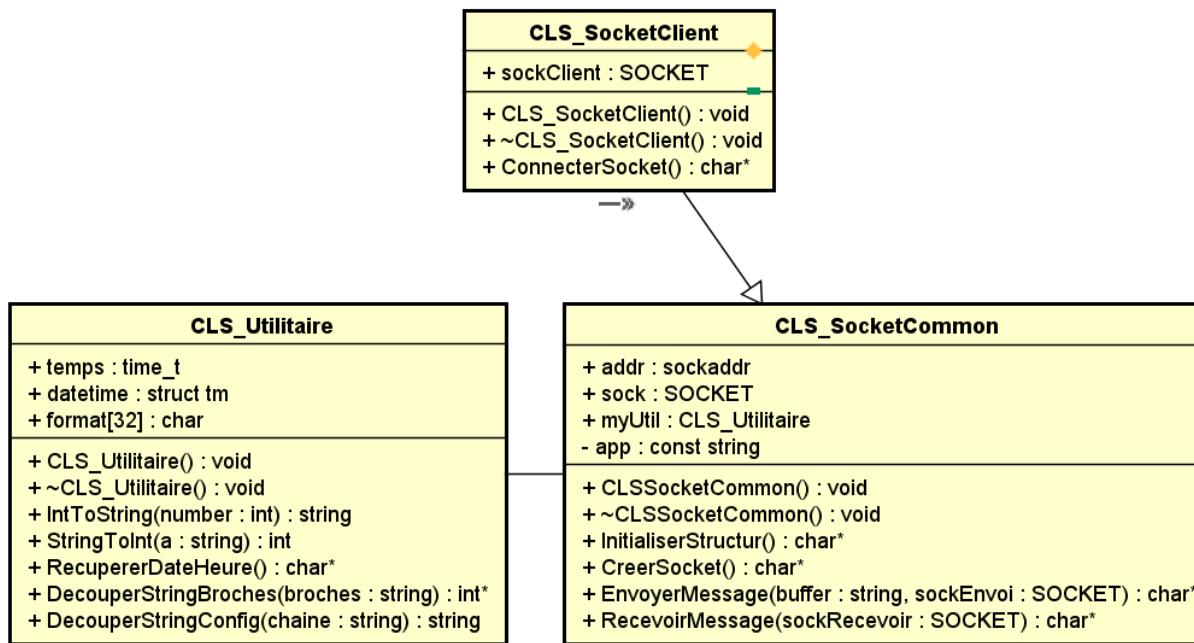
- La méthode `InitialiserWiringPI` sert à initialiser la bibliothèque `WiringPI` et de récupérer les paramètres dans le fichier de config qui va permettre de se servir des entrées sorties de la Raspberry (voir annexe).
- La méthode `Detection` va permettre de détecter s'il y a un véhicule sur la place de parking en renvoyant true ou false, comme vous pouvez le voir la fonction possède quatre paramètres qui sont les numéros des broches d'entrée et sortie sur lesquelles le capteur et les LED sont branchés à la raspberry (voir annexe).
- La classe `CLS_DetecterVehicule` et `CLS_Utilitaire` sont reliées par une association car la classe `CLS_DetecterVehicule` va avoir besoin de mettre au bon format les données récupérées dans le fichier de config.
- On peut remarquer les variables `start_t` et `end_t` elles servent à calculer la durée de l'impulsion renvoyé par le capteur, la variable `distance` qui permet de connaître la distance entre le capteur et un

obstacle et la variable distanceDeDetection est initialisé à partir du fichier de config situé à l'adresse que contient la variable app.

Suite à cela avec la partie SUPERVISER sur l'application PC nous avons décidé de réaliser la connexion socket berkeley entre nos deux parties. Il faut savoir que dans une communication socket il y a un ou plusieurs clients(ordinateur) qui vont pouvoir se connecter à un serveur(ordinateur) et ainsi ce dernier peut envoyer mais aussi recevoir des messages des clients et chaque client peut également envoyer et recevoir des messages du serveur. Etant donnée qu'il y aura une raspberry par étage et qu'un seul poste de supervision il est évident que le poste de supervision doit être le serveur et que chaque raspberry soit un client, ce qui permettra donc de vérifier le bon fonctionnement de chaque étage en fonction de si oui ou non il est connecté au serveur et s'il envoi le bon numéro d'étage qu'il va chercher sur la base de données.

La librairie à utiliser pour développer des sockets est la librairie socket.h qui est intégré à l'OS de linux.

Après cette analyse j'ai donc dû mettre au point une classe qui permettait de communiquer en socket mais vu que qu'un client et un serveur ont des méthodes en commun j'ai décidé de créer une classe CLS_SocketCommon qui peut être utilisé pour un serveur ou bien un client, et ensuite j'ai créé une classe spécialisée en client, la classe CLS_SocketClient. Voici le diagramme de classe de ces dernières :



Comme vous pouvez le voir la classe CLS_SocketClient hérite de la classe CLS_SocketCommon, ce qui lui donne accès à toute les méthodes et attribut de la classe CLS_SocketCommon.

Pour la classe CLS_SocketCommon :

- La méthode `InitialiserStructur` permet d'initialiser la structur qui contient l'adresse IP et le port de connexion du serveur, récupéré dans le fichier de config.
- La méthode `CreerSocket` qui va donc se servir de la structure précédemment initialisée pour créer le socket.
- La méthode `EnvoyerMessage` permet d'envoyer des messages au serveur, cette dernière possède deux paramètres le premier étant le buffer qui va contenir le message à envoyer) puis le deuxième est le `sockEnvoi` qui correspond au socket précédemment créé.

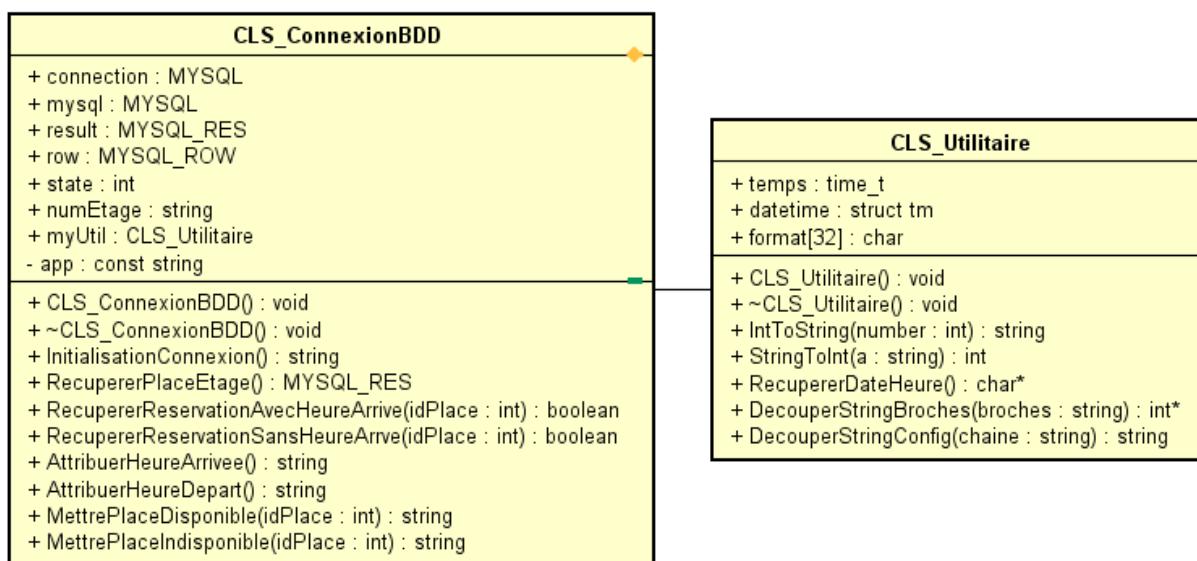
- La méthode recevoirMessage qui permet de recevoir les messages envoyés par le serveur comme vous pouvez le voir elle a un paramètre qui correspond au socket créé précédemment.

Dans la classe CLS_SocketClient :

- La méthode ConnecterSocket de la classe CLS_SocketClient, cette dernière va donc permettre de connecter le client et le serveur pour qu'ils puissent échanger.

On peut également constater que les classes CLS_Utilitaire et CLS_SocketCommon sont reliées par association car la classe CLS_SocketCommon a besoin de mettre au bon format les données récupérées dans le fichier de config à l'aide de la méthode DecouperStringConfig puis de convertir la chaîne string du port en int.

Une fois tout cela fait je devais communiquer avec la base de données pour changer la disponibilité des places ou ajouter une heure d'arrivée et une heure de départ, pour cela j'ai regardé comment communiquer d'une Raspberry en C++ à une base de données MySql et j'ai trouvé la librairie mysql.h. j'ai donc créé la classe CLS_ConexionBDD :



Tout d'abord la méthode InitialiserConnexion permet de récupérer les identifiants du serveur dans le fichier de config puis d'établir une connexion au serveur à l'aide ses identifiants (voir annexe), puis la méthode RecupererPlaceEtage va servir à récupérer toute les informations des places d'un étage et elle va retourner l'intégralité de ces données, les méthodes RecupererReservationAvecHeureArrivee et RecupererReservationSansHeureArrivee vont permet de savoir s'il existe ou non une réservation ayant pour place l'idPlace que l'on passe par paramètre. Les méthodes AttribuerHeureArrivee et AttribuerHeureDepart permettent donc d'attribuer une heure d'Arrivée ou de Départ à la réservation trouvé et enfin les deux dernières méthodes MettrePlaceDisponible et MettrePlaceIndisponible. Les variables connection et mysql servent à initialiser et stocker la connexion à la base de données, result va contenir le résultat du requête sql puis row va permettre de traiter ce que contient result (voir annexe), la variable state va permettre de détecter les erreurs des requêtes, la variable numEtage indiquera le numéro d'étage auquel la raspberry correspond et app contient le chemin pour aller au fichier de config.

Ayant besoin de méthode (utilitaire) dans certaine classe, j'ai décidé de créer une classe CLS_Utilitaire :

CLS_Utilitaire
+ temps : time_t
+ datetime : struct tm
+ format[32] : char
+ CLS_Utilitaire() : void
+ ~CLS_Utilitaire() : void
+ IntToString(number : int) : string
+ StringToInt(a : string) : int
+ RecupererDateHeure() : char*
+ DecouperStringBroches(broches : string) : int*
+ DecouperStringConfig(chaine : string) : string

-La méthode IntToString va permettre de convertir un type int en type string et donc la méthode StringToInt permet de faire l'inverse.

-La méthode RecupererDateHeure va permettre de récupérer la date et l'heure du moment présent au même format que celui de la base de données (exemple : Année-Mois-Jour Heure:Minute:Seconde).

-La méthode DecouperStringBroches sert à découper en quatre parties la chaîne qui contient les quatre broches de la place correspondante que l'on récupère sur la base de données (exemple : "1,3,26,2" est la chaîne d'entrée et en sortie on obtient les quatres valeurs séparées "1" "3" "26" "2")

-Et enfin la méthode DecouperStringConfig permet de découper un string à partir d'un caractère spécifique (exemple : "IP serveur :127.35.254.123" après le passage dans la méthode il nous reste plus que "127.35.254.123")

-Les trois variables servent à récupérer et mettre la date et l'heure au même format que la base de données

J'ai donc dû réaliser le fichier de configuration de l'application, pour pouvoir modifier aisément la chaîne de connexion à la base de données ou bien l'adresse IP et le port du serveur socket ou encore modifier la distance de détection des véhicules. Vous pouvez voir à quoi ressemble le fichier de configuration ci-dessous :



```

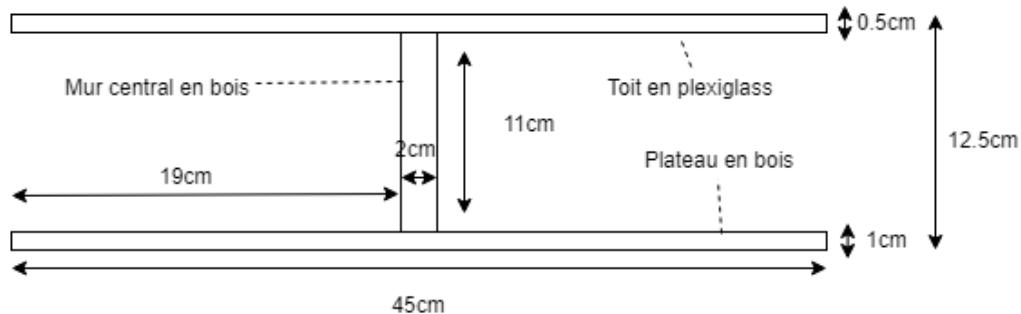
app.txt

Fichier Édition Rechercher Options Aide
Numéro étage :1
IP du serveur :mysql-projetlifi.alwaysdata.net
user :156739
mot de passe :projetlifi
nom de la base de données :projetlifi_bdd

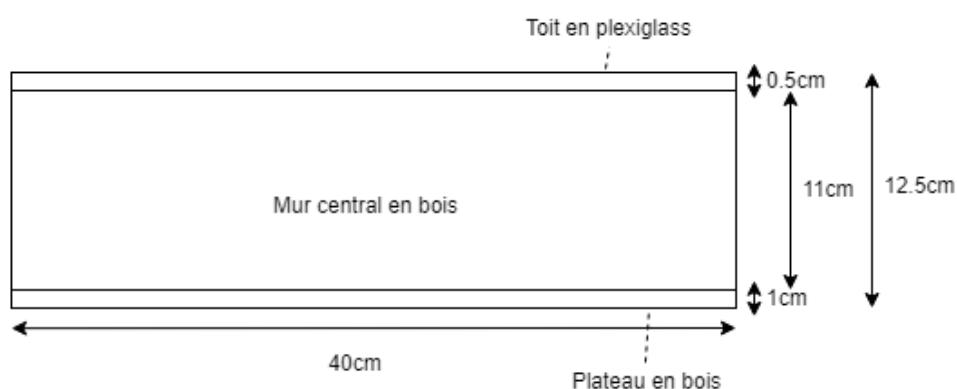
IP Supervision :172.31.254.73
Port Supervision :13000
|
Distance detecter voiture en centimetre :7

```

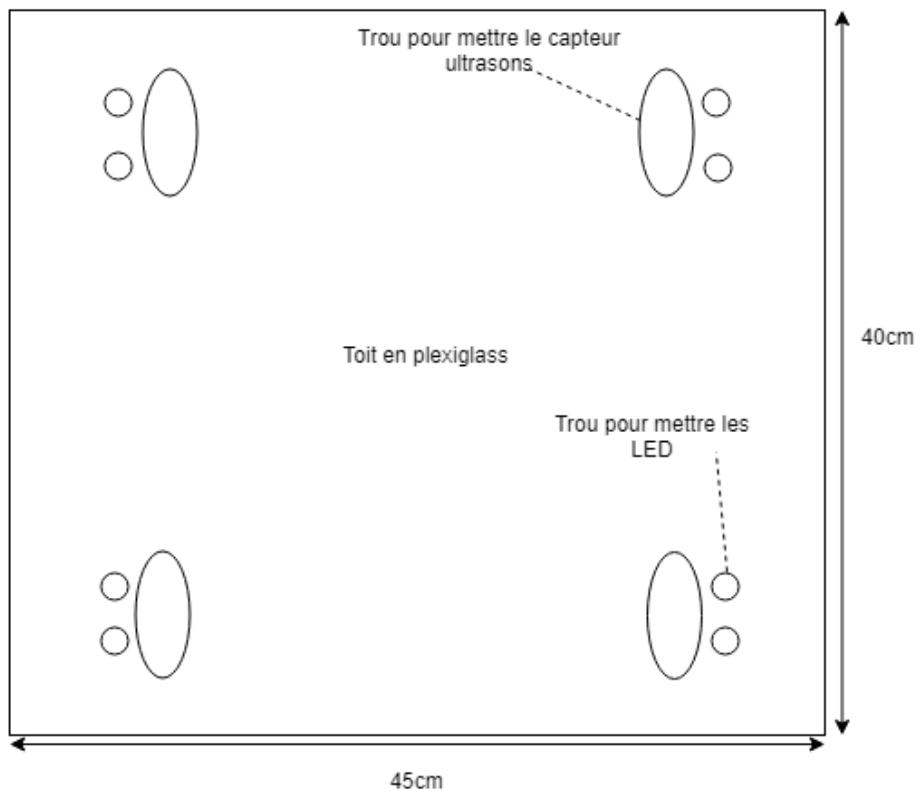
D'autre part j'ai décidé de réalisé la maquette pour pouvoir commencer à faire des tests. J'ai donc réalisé plusieurs plans et les ai présentés à mon équipe pour qu'ils me donnent leur avis et c'est donc le plan de maquette suivant que l'on a préféré et que j'ai construit :



Vue de profile

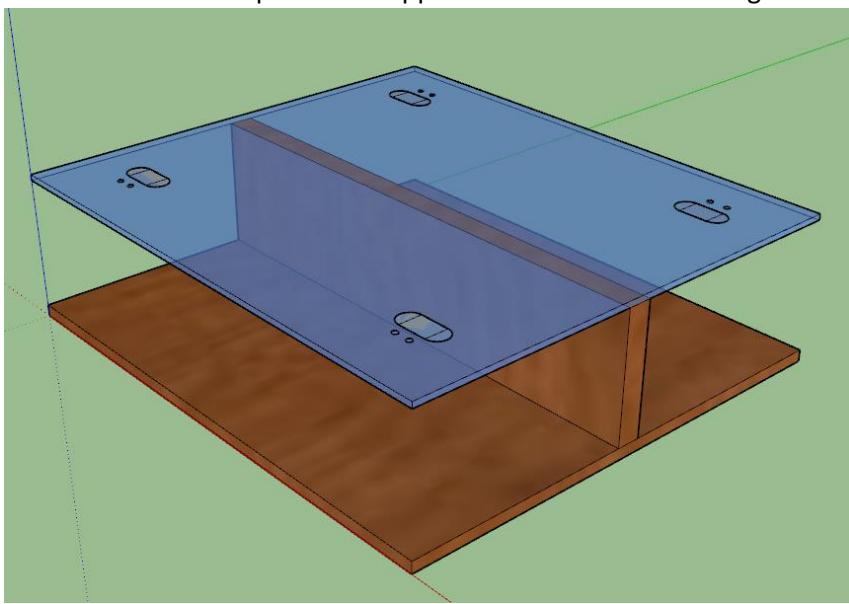


Vue de face



Vue du dessus

Suite à ce plan j'ai fait le model 3D dans le logiciel Sketchup2018 pour avoir un meilleur aperçu et mieux se rendre compte de son apparence final. Voici une image du model 3D :



Puis une fois le modèle créé et validé je l'ai réalisé. Pour cela j'ai acheté un plateau, une planche de bois et une plaque de plexiglass que j'ai découpé aux bonnes dimensions, voici le matériel utilisé :



Et voici le résultat final de la maquette avec les capteurs et les LED :

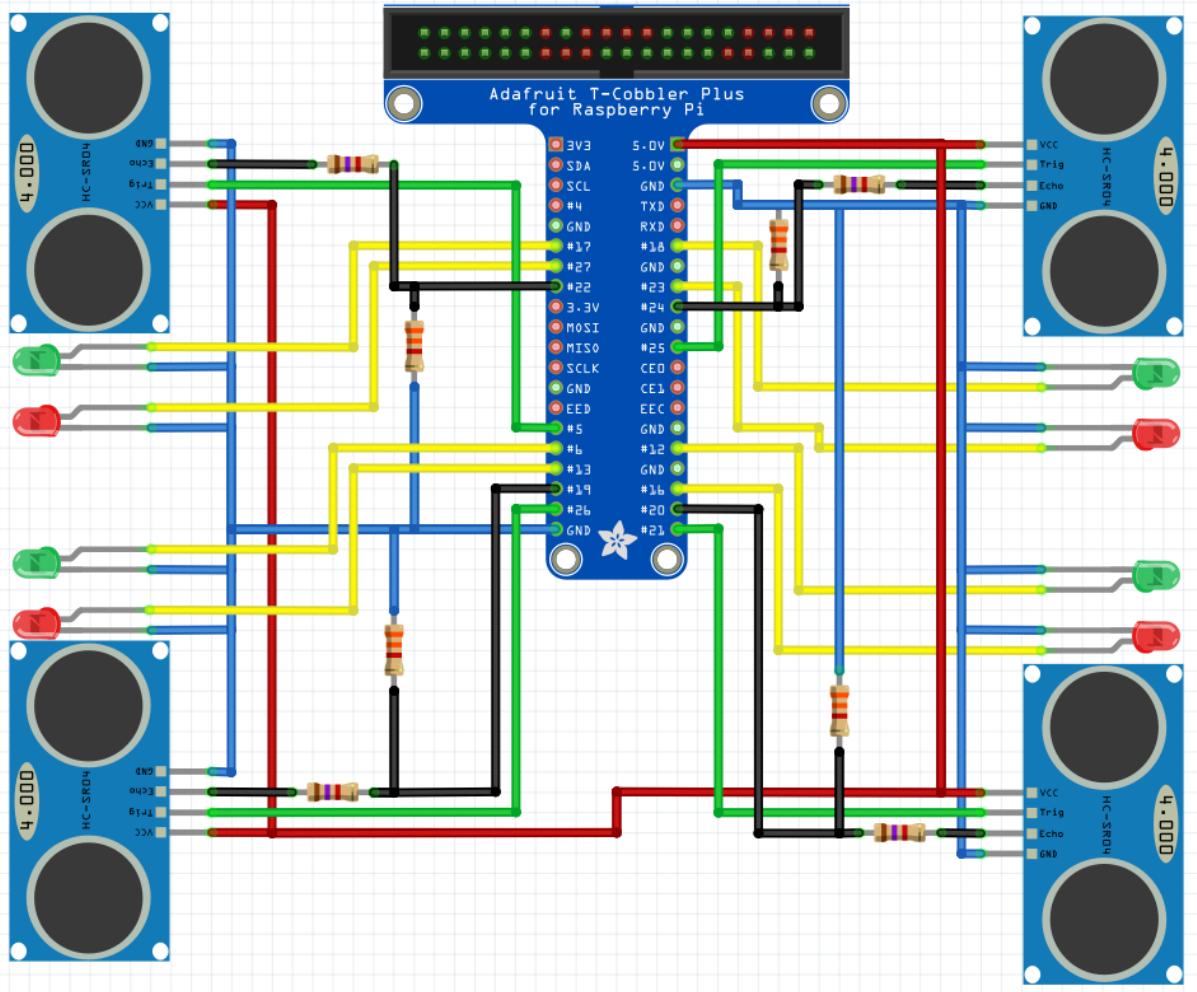


On peut constater que les équerres présentes sur l'image du matériel utilisé n'ont pas été utilisé car le plexiglass est plus résistant que prévu et donc il n'a pas besoin de renfort. Puis j'ai peint la maquette en noir et blanc pour marquer chaque place :



Une fois cela fait j'ai donc dû réaliser le câblage des capteurs et des LED, pour se faire j'ai d'abord réalisé le schéma du câblage sur le logiciel Fritzing qui est un logiciel gratuit auquel on peut ajouter des packs téléchargeables sur internet, ces packs servent à ajouter des composants/matériels électriques au logiciel. Pour ma part j'ai eu besoin de télécharger le pack adafruitRaspberry qui m'a permis d'utiliser les composants pour contrôler les entrées sorties de la raspberry.

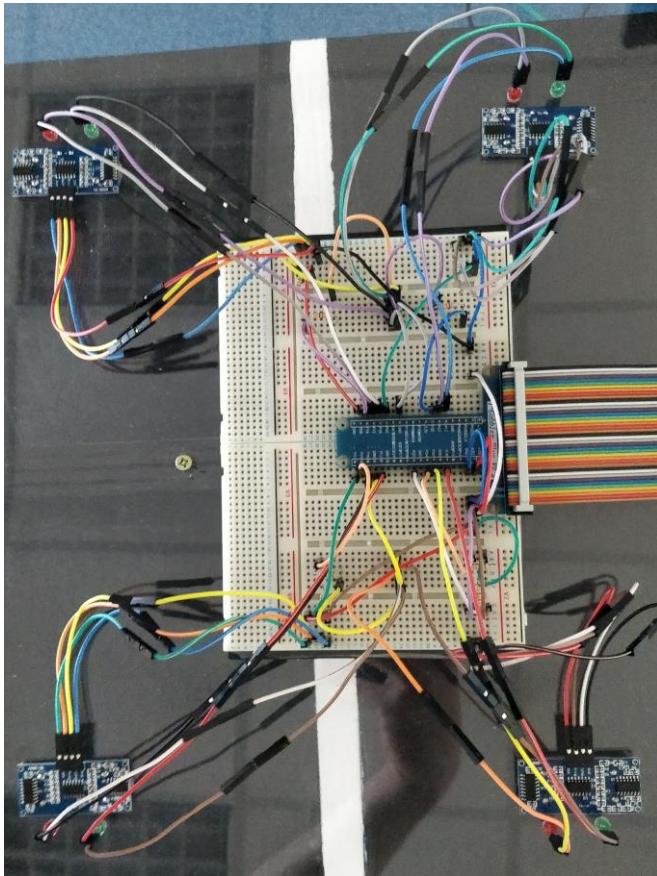
Voici le schéma réalisé :



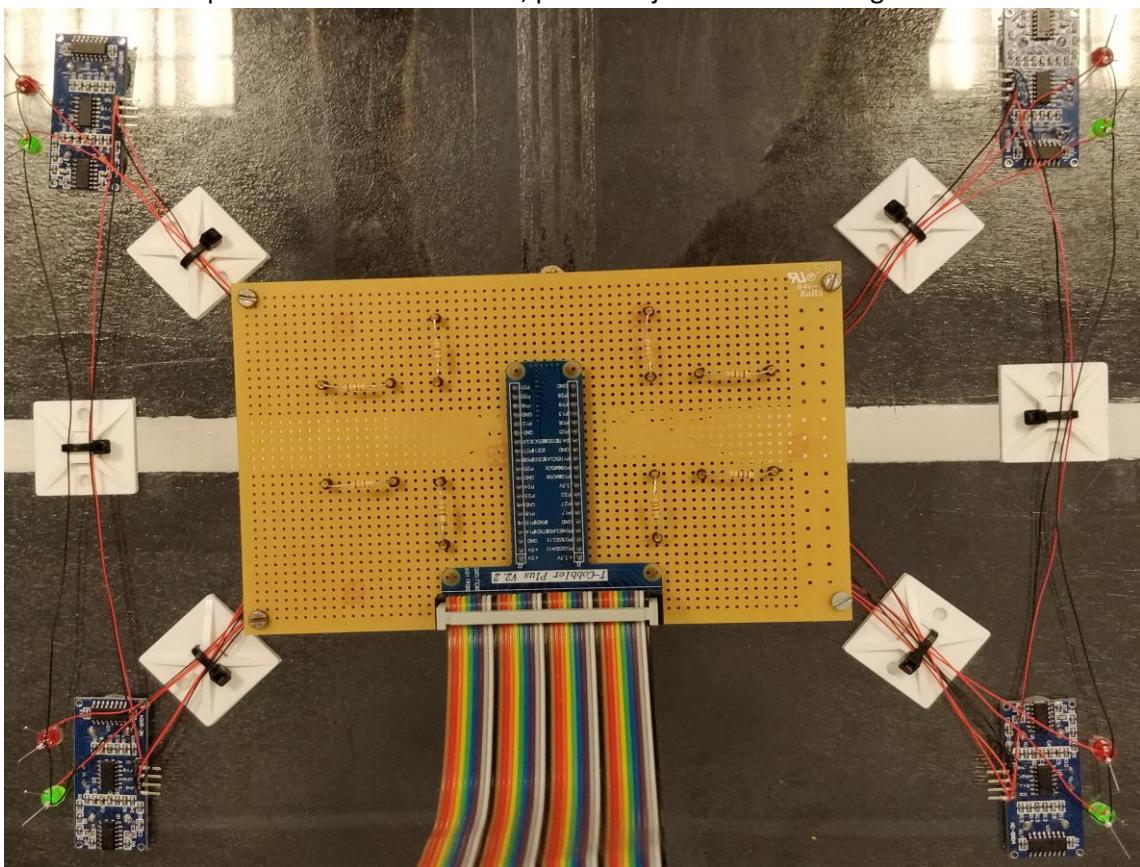
Code couleur :

- Rouge : 5V pour les capteurs ultra-sous
- Bleu : GND général
- Noire : Sorties des capteurs et entrées raspberry
- Vert : Entrées capteur et Sorties raspberry
- Jaune : Positif des LED 3,3V

Puis j'ai donc réalisé le câblage de la maquette que voici :

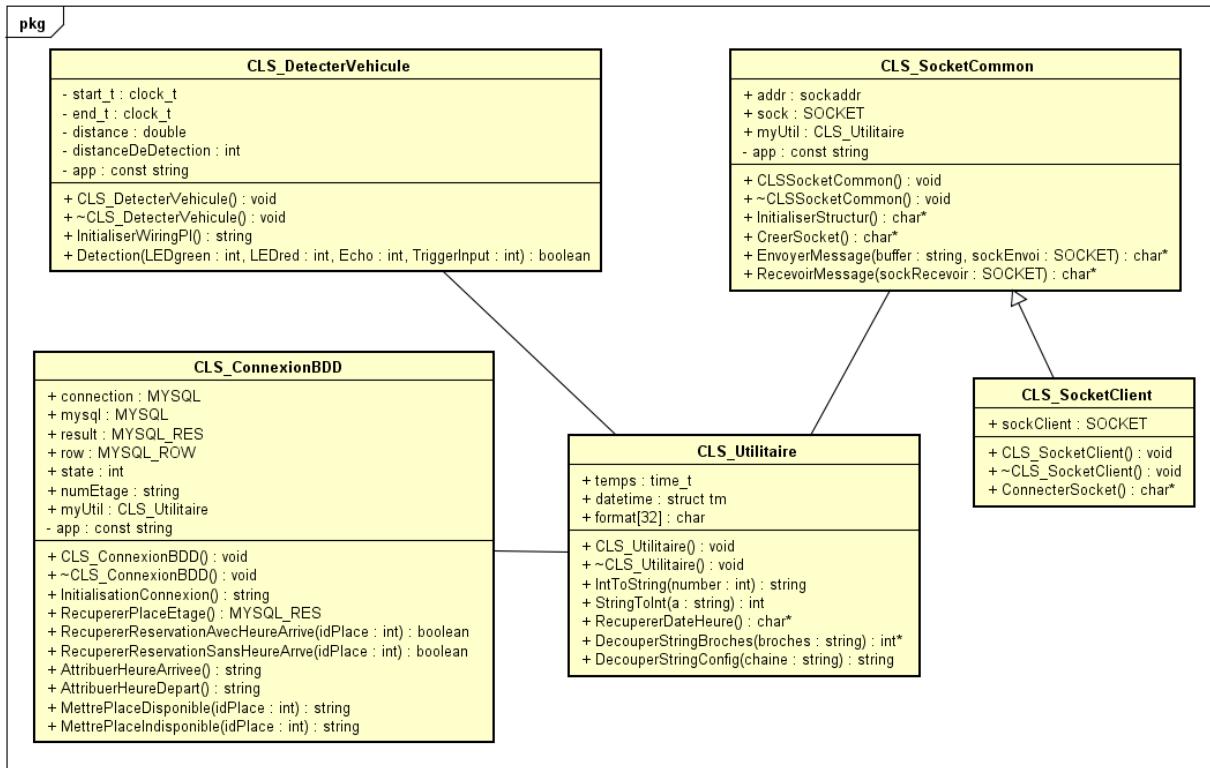


Par la suite j'ai décidé de refaire le câblage plus proprement et aussi pour éviter qu'un câble ne se débranche sans qu'on sache ou le remettre, pour cela j'ai réalisé le câblage si dessous :



Après cela j'ai donc réalisé diagramme de classe de tout le projet pour ensuite pouvoir le développer.

Voici le diagramme :



J'ai donc développé le main du programme qui instancie les classes CLS_ConexionBDD, CLS_DetecterVehicule, CLS_Utilitaire et CLS_SocketClient est composé des trois fonctions suivante :

- La fonction Initialisation permet d'initialiser tous les objets ainsi que toutes les librairies nécessaires.
- La fonction ConnexionSocket va donc être lancé par un thread pour toujours être en exécution et elle va se connecter au serveur puis quand elle reçoit le message "Quel étage ?" elle va envoyer au serveur son numéro d'étage qu'elle est allée récupérer sur la base de données.
- La fonction SurveillerParking est lancée dans une boucle while et va donc surveiller le parking en continu jusqu'à ce que le programme soit arrêté de force.

- Voici le main en code :

```
int main()
{
    Initialisation(); //Initialisation de l'application
    pthread_t ThreadSocket;
    pthread_attr_t attr;
    pthread_attr_init(&attr);

    pthread_create(&ThreadSocket, NULL, &ConnexionSocket, NULL );

    while(true){
        SurveillerParking();
    }

    return 0;
}
```

Voici le contenu de la fonction Initialisation :

```
//Initialisation de l'application
void Initialisation(){

    //Création des objets
    try{
        myConn = new CLS_ConexionBDD();
        myDetect = new CLS_DetecterVehicule();
        myUtil = new CLS_Utilitaire();
        myClient = new CLS_SocketClient();
    }catch(int e){
        cout << "Erreur dans la creation des objets !" << endl;
    }

    //Initialisation des librairies
    cout << myDetect->InitialiserWiringPI() << endl;
    cout << myConn->InitialiserConnexion() << endl;

    //Initialisation et création du socket
    cout << myClient->InitialiserStructur() << endl;
    cout << "\n" << endl;
}
```

Voici le contenu de la fonction ConnexionSocket :

```
//Communiquer en socket avec le poste de supervision
void* ConnexionSocket(void*){
    string connexion;
    while(true){
        if (connexion != "Echec de connection !"){
            string messageReçu = myClient->RecevoirMessage(myClient->sockClient);
            if (messageReçu == "Quel étage ?"){
                myClient->EnvoyerMessage(myConn->numEtage, myClient->sockClient);
                cout << myConn->numEtage << endl;
                cout << "ok" << endl;
            }else{
                cout << "not ok" << endl;
                cout << myClient->CreerSocket() << endl;
                connexion = myClient->ConnecterSocket();
                cout << connexion << endl;
            }
        }else{
            cout << "Serveur introuvable !" << endl;
        }
        delay(100);
    }
}
```

Voici le contenu de la fonction SurveillerParking :

```
//Surveiller le parking et update la base de données
void SurveillerParking(){
    MYSQL_RES res = myConn->RecupererPlaceEtage(); //Variable qui contient toutes les places
    MYSQL_ROW row; //Variable pour manipuler les données reçus
    //tant qu'il y a des données dans res continuer
    while((row = mysql_fetch_row(&res)) != NULL){
        cout << "ID de la place : " << row[0] << endl;
        cout << "Place : " << row[1] << "\nAllée : " << row[3] << endl;
        cout << "Broches de la place : " << row[5] << endl;
        cout << "Disponibilité : " << row[2] << endl;
        string broches = row[5];
        int* tabBroches = myUtil->DecouperStringBroches(broches); //Permet de recuperer les broches
        //Si véhicule détecté mettre la place en disponible
        if(myDetect->Detection(tabBroches[0], tabBroches[1], tabBroches[2], tabBroches[3]) == 1){
            //Si une réservation correspond à cette place alors mettre l'heure d'arrivée dans la rés
            if(myConn->RecupererReservationSansHeureArrive(myUtil->StringToInt(row[0])) == 1){
                myConn->AttribuerHeureArrive();
            }
            cout << myConn->MettrePlaceIndisponible(myUtil->StringToInt(row[0])) << endl;
        }else{//Sinon mettre la place en disponible
            //Si une réservation correspond à cette place alors mettre l'heure de départ dans la rés
            if(myConn->RecupererReservationAvecHeureArrive(myUtil->StringToInt(row[0])) == 1){
                myConn->AttribuerHeureDepart();
            }
            cout << myConn->MettrePlaceDisponible(myUtil->StringToInt(row[0])) << endl;
        }
        delay(70);
        cout << "\n" << endl;
    }
}
```

3.2.4 Conclusion

Pour conclure le câblage et le test du capteur est un succès ainsi que la création des classes CLS_DetecterVehicule, CLS_SocketCommon, CLS_SocketClient, CLS_ConexionBDD et CLS_Utilitaire, elles ont chacune été testé individuellement avec succès. La réalisation de la maquette est terminée ainsi que le câblage de tous les composants. Le programme final est terminé et est fonctionnel, chacune des classes fonctionnent correctement ensemble.

J'ai respecté les temps annoncés dans le diagramme de Gantt.

3.3 Partie SUPERVISER sur l'application PC

Partie effectuée par Vincent Brunaud

3.3.1 Spécifications

Ma partie consiste à réaliser une application de supervision sur PC. J'ai donc créé une application permettant de gérer les différentes parties de l'application telle que les utilisateurs, les réservations les historiques... Cette application contient aussi une partie configuration permettant de modifier les paramètres de configuration à la base de données ainsi que la configuration de communication socket. Je travaille avec le langage C# en WPF et son Framework **MahApps** (Installation en annexe). Pour la connexion à la base de données j'utilise le Framework **Entites** (Installation en annexe).

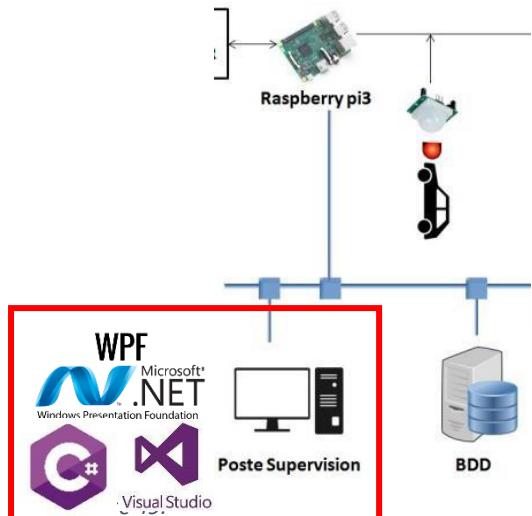
3.3.2 Présentation des technologies utilisées

MahApps.Metro est un projet de Paul Jenkins commencé en 2011 comme un moyen simple d'introduire une interface utilisateur de style Metro dans votre application WPF. Depuis lors, il a évolué et a pris des contributions. C'est un Framework pour le langage C# tel que Bootstrap pour le langage Web.

WPF (Windows Presentation Foundation) est un système de présentation nouvelle génération qui génère des applications clientes Windows avec des expériences utilisateurs visuellement surprenantes. Le cœur de WPF est un moteur de rendu vectoriel et indépendant de toute résolution, créé pour tirer parti du matériel graphique moderne. WPF étend le cœur avec un jeu complet de fonctionnalités de développement d'applications qui incluent Extensible Application Markup Language (XAML), des contrôles, la liaison de données et la disposition ainsi que des graphiques 2-D et 3-D, l'animation, les styles, les modèles, les documents, les médias, le texte et la typographie. WPF est inclus dans le Microsoft .NET Framework, vous pouvez donc générer des applications intégrant d'autres éléments de la bibliothèque de classes .NET Framework.

Entity Framework est un outil permettant de créer une couche d'accès aux données (DAL pour Data Access Layer) liée à une base de données relationnelle. Il propose la création d'un schéma conceptuel composé d'entités qui permettent la manipulation d'une source de données, sans écrire une seule ligne de SQL, grâce à LinQ To Entities. Comparé à d'autres solutions de mapping objet-relationnel (ORM), Entity Framework assure l'indépendance du schéma conceptuel (entités ou objets) du schéma logique de la base de données, c'est-à-dire des tables. Ainsi, le code produit et le modèle conceptuel ne sont pas couplés à une base de données spécifique.

3.3.3 Synoptique de l'architecture matérielle et logicielle



Structure de la partie SUPERVISER avec l'application

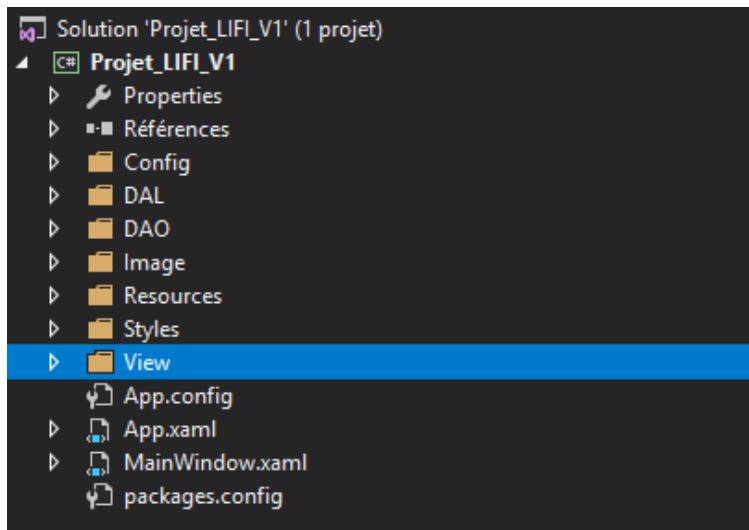
Ma partie consiste à gérer les réservations et les utilisateurs via une application PC. Pour cela je dois communiquer avec la base de données en me connectant à un Web service et je dois être en contact avec la carte embarquée pour vérifier que la connexion est bien établie. J'utilise la technologie Socket Berkley pour effectuer cela.

En termes de logiciel j'utilise uniquement le logiciel Visual Studio pour développer mon application. Cet environnement de développement est un choix réfléchi. En effet, j'ai décidé de développer sur ce logiciel car c'est un logiciel que je maîtrise avec bientôt 2 ans d'expérience dessus désormais mais aussi pour sa possibilité d'extension avec les différents Framework. J'ai pu ajouter MahApps Metro me permettant d'ajouter des styles sur mon application.

Pour ce qui est des langages de programmation, je développe en C# avec WPF en utilisant les requêtes LINQ pour effectuer mon CRUD (Create Read Update Delete). Une opération de requête LINQ se compose de trois actions : obtenir la ou les sources de données, créer la requête, puis exécuter cette dernière.

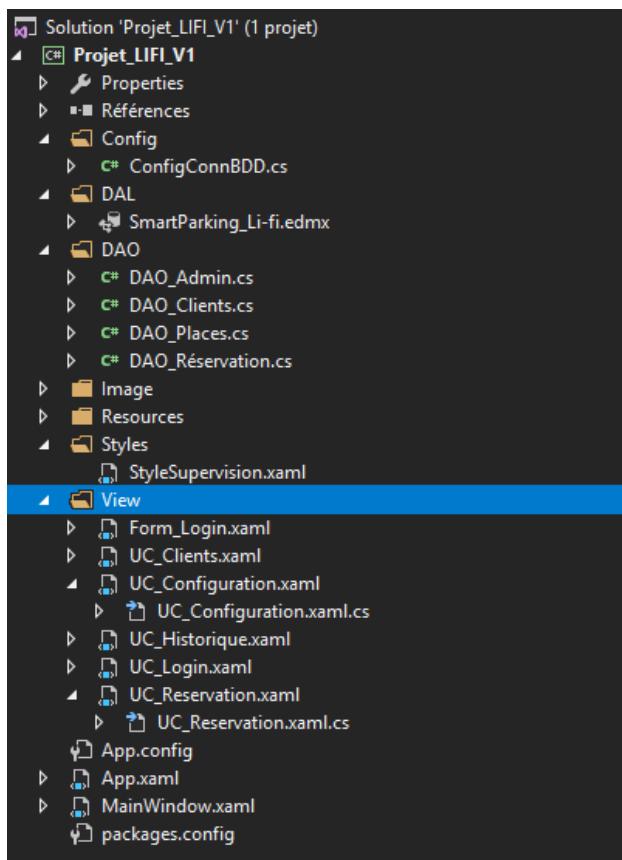
3.3.4 Travail effectué

J'ai réalisé le design de l'application via le logiciel Visual Studio en langage C# WPF. Cette application permet de gérer les réservations et les utilisateurs. Pour commencer, je vais vous présenter l'arborescence de mon projet :

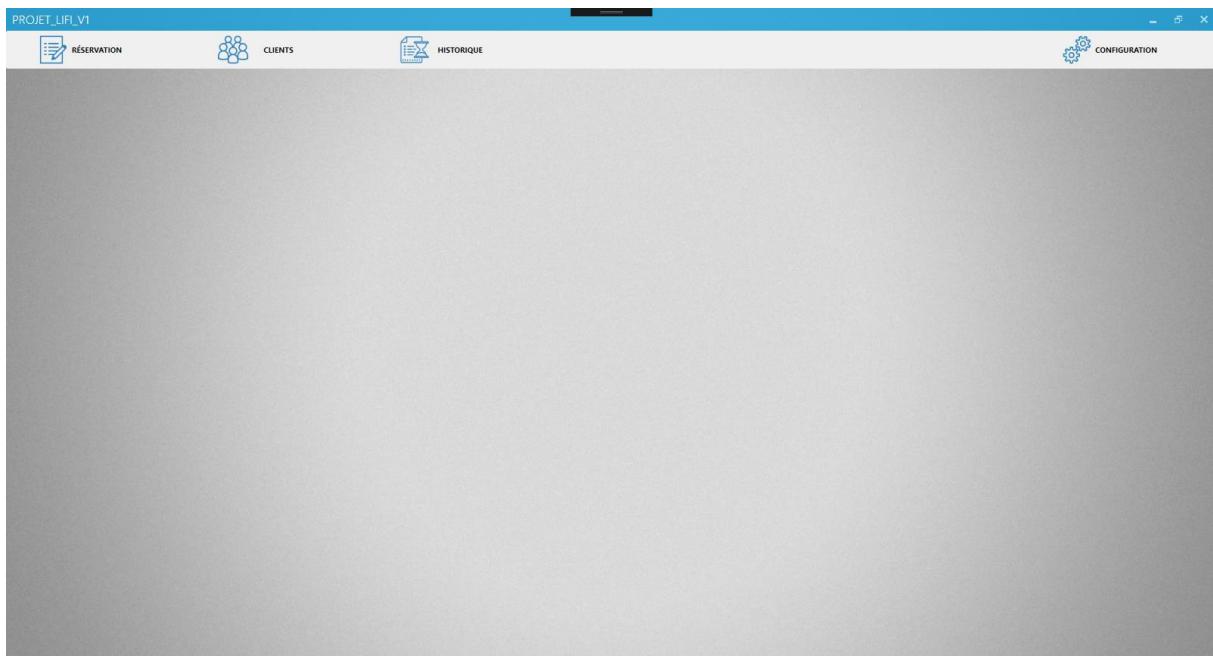


Le dossier **Config** contient ma classe de connexion à la base de données, Le dossier **DAL** (Data Access Layer) contient mon modèle EntityFramework, le dossier **DAO** (Data Access Object) contient la totalité de mes DAOs, dans le dossier **Image** il y a mes différentes images dont j'ai besoin sur mon application. Le dossier **Style** contient le fichier de Style que j'applique sur mon application et dans le dossier **View** j'ai tous mes UserControls.

Voilà désormais un aperçu avec l'arborescence détaillé :



Voici un premier aperçu dans l'application au lancement du programme :



Les différentes icônes sur le haut de la page correspondent à mon menu, le texte à côté est là pour donner des indications sur les onglets du menu.

La grande partie en dessus en gris correspond à mon **main container**. C'est ici que s'affiche les informations de chacun de mes onglets. Mon menu reste fixe sur la barre du haut tout en changeant de page en fonction de l'onglet cliqué auparavant.

Voici désormais un aperçu de l'onglet utilisateur :

 A screenshot of the "CLIENTS" tab from the application. The top navigation bar shows "PROJET_LIFI_V1" and the "CLIENTS" tab is selected. Below the bar is a section labeled "Liste des clients :" containing a "DataGridView" with columns: ID, Nom, Prenom, NumTel, Adresse, Ville, CodePostal, Mail, and Motde_Passe. The grid displays 15 rows of client data. To the left of the grid is an "Ajouter un client" form with fields for Nom, Prénom, Email, Téléphone, and Code postal, along with an "Ajouter" button. To the right is a "Rechercher un client :" form with a search input field and a "Rechercher" button. There are also three circular icons with edit, delete, and add symbols.

ID	Nom	Prenom	NumTel	Adresse	Ville	CodePostal	Mail	Motde_Passe
1	INVITE	INVITE	INVITE	INVITE	INVITE	INVIT	INVITE	fgr54e4g
2	MERMAZ	Quentin	0685169715	Rue Principale	Anney	74000	mermaaz.quentin@outlook.com	qm
3	POULIN	Lancelot	0657920136	Rue Des Jonguilles	Faverges	74290	poulin.lancelot@outlook.com	lpp
4	BRUNAUD	Vincent	0651238744	Rue De Loverchy	Bons	74500	brunaud.vincent@outlook.com	vb
5	POIRIER	Colin	0698201478	Impasse Des Violettes	Gruffy	74260	poirier.colin@outlook.com	cp
6	MESTRE	Quentin	0650035782	Rue Des Balmettes	Anney	74000	mestre.quentin@outlook.com	qm1
7	NIGON	Antoine	0688016730	Rue Des Rosiers	Sillingy	74950	nigon.antoine@outlook.com	an
8	HUMBLOT	Julien	0698014346	Impasse Des Fins	Brenthonne	74500	humblot.julien@outlook.com	jh
9	FAVRE	Robin	0666044752	Rue De La Cité	Meyhet	74000	favre.robin@outlook.com	rf
10	JEANDET	Steve	0670172747	Rue Du Carrefour	Aix Les Bains	73000	jeandet.steve@outlook.com	sj
11	DA SILVA	Thomas	0615456952	Rue De La Gare	La Roche	74800	da.silva.thomas@outlook.com	tds
12	TELLIER	Julien	0620221955	Rue De La Tour	Metz	57000	tellier.julien@outlook.com	jt
13	DEBALME	Mickael	0667278156	Rue De Lachat	Anney	74000	debalme.mickael@outlook.com	md
14	JUPITER	Nicolas	0615985234	Impasse Des Mines	Annemasse	74100	jupiter.nicolas@outlook.com	nj
15	DA COSTA	Paolo	0652314566	Les Champs	Mesigny	74090	dacosta.paolo@outlook.com	pdc

La partie en haut à gauche correspond à la **DataGrid**, c'est ici que sont afficher les données de la base de données. Les différentes **TextBoxs**, permettent d'ajouter un client en insérant les données adéquates. Les 3 boutons permettent d'effectuer le **CRUD** (Create Read Delete)

Update) un pour effectuer un ajout, un autre pour une modification et le dernier pour supprimer un utilisateur. J'ai aussi ajouté dans mon design une barre de recherche permettant par la suite d'afficher uniquement les données d'un client spécifique.

À présent, nous allons voir l'aperçu de l'onglet réservation :

ID	Reference	DebutReservation	FinReservation	Arrivée	Depart	mail
81	5057	5/28/2018 3:00:00 PM	5/28/2018 10:00:00 PM			favre.robin@outlook.com
82	8332	5/28/2018 3:00:00 PM	5/28/2018 10:00:00 PM			favre.robin@outlook.com
83	7467	5/28/2018 3:00:00 PM	5/28/2018 10:00:00 PM			favre.robin@outlook.com
84	3512	5/28/2018 3:00:00 PM	5/28/2018 10:00:00 PM			favre.robin@outlook.com
85	6246	5/28/2018 3:00:00 PM	5/28/2018 10:00:00 PM			favre.robin@outlook.com
86	6278	5/28/2018 3:00:00 PM	5/28/2018 10:00:00 PM			favre.robin@outlook.com
87	3452	5/28/2018 3:00:00 PM	5/28/2018 10:00:00 PM			favre.robin@outlook.com
88	8941	5/28/2018 3:00:00 PM	5/28/2018 10:00:00 PM			favre.robin@outlook.com
89	1229	5/28/2018 3:00:00 PM	5/28/2018 10:00:00 PM			favre.robin@outlook.com

J'ai effectué un principe similaire pour la DataGridView, placée en haut à gauche avec les informations de la base de données à l'intérieur. Comme pour la partie utilisateur, sur le poste de Supervision je peux directement effectuer une réservation avec un CRUD me permettant de réaliser des ajouts, des modifications ainsi que des suppressions. Une barre de recherche pour effectuer une réservation en fonction du nom du client. Pour les informations sur les places de parkings j'ai réalisé un tableau dynamique que j'explique plus en détail juste en dessous.

Réalisation d'un tableau dynamique en C# :

Pour la mise en forme des places de parkings j'ai créé un tableau dynamique me permettant d'ajout automatique le nombre de colonnes et de lignes que je souhaite. Pour cela, j'ai travaillé avec l'algorithme suivant :

Calculer le nombre d'allée

Calculer le nombre de places

Calculer le nombre de place par allée

Ajouter N colonnes en fonction du nombre de places par allée

Ajouter M lignes en fonction du nombre d'allées.

Les 4 places correspondent aux carrés verts, qui signifie que ces places sont disponibles. Pour faire le tableau dynamique j'ai tout d'abord créé manuellement une grille.

```
<Grid x:Name="grid" Loaded="grid_Loaded_1">
```

En donnant un nom à cette grille je peux m'en servir dans mon code en C#. Une fois cette grille créée, je n'ai pas qu'à ajouter automatiquement mes lignes et mes colonnes en fonction du nombre

de place par allées et du nombre d'allées. C'est pour ça que j'ai eu besoin d'établir un algorithme pour savoir combien de lignes et de colonnes ont besoin d'être générées.

```
grid.ColumnDefinitions.Add(new ColumnDefinition() { Width = GridLength.Auto });
```

Voilà comment je crée mes lignes et mes colonnes. Pour en créer le nombre nécessaire je fais une boucle qui crée autant de colonnes que de places par allées et autant de lignes que d'allées. Une fois que j'ai le bon nombre de ligne et de colonnes j'ajoute à l'intérieur de ces colonnes et de ces lignes des labels qui ont un arrière-plan de couleur verte pour l'initialisation. En lisant les données de la base de données je sais quand la place est disponible ou non. Je n'ai plus qu'à vérifier cette disponibilité et si elle ne l'a pas, je change la couleur d'arrière-plan en rouge ce qui signifie que la place est occupée.

```
if (places[IDPlace].Disponible == true)
// Vérification si la place est disponible
{
    // Passage de couleur du background en vert
    label1.Background = new SolidColorBrush(Colors.Green);
}
else // La place est indisponible
{
    // Passage de la couleur du background en rouge
    label1.Background = new SolidColorBrush(Colors.Red);
};
```

Pour terminer sur le tableau dynamique, je récupère la lettre de l'allée et le numéro de la place stocker dans la base de données pour les afficher dans mon label et savoir quel carré correspond à quelle place. Cette action se déroule dans la création de mon label, en lui attribuant des spécificités. Je lui défini une taille, et récupère la lettre de l'allée et le numéro de la place qui sont le contenu de mon label (**Content**).

```
Label label1 = new Label()
{
    Width = 50,
    Height = 50,
    Content = allees[IDAAllee].Lettre +
    places[IDPlace].Numero
};
```

Le troisième onglet est l'onglet des historiques :

The screenshot shows a software interface titled 'PROJET_LIFI_V1'. The 'HISTORIQUE' tab is selected. A sidebar on the left allows choosing a duration: Journée, Semaine, or Mois. The main area displays a table of reservation history:

ID	Reference	DebutReservation	FinReservation	Arrivée	Départ	mail
1	1000	4/23/2018 1:00:00 PM	4/23/2018 5:00:00 PM	4/23/2018 1:08:54 PM	4/23/2018 4:47:12 PM	INVITE
2	1002	4/23/2018 3:00:00 PM	4/23/2018 7:00:00 PM	4/23/2018 3:08:54 PM	4/23/2018 6:47:12 PM	mermaz.quentin@outlook.com
3	1003	4/24/2018 1:00:00 PM	4/24/2018 5:00:00 PM	4/24/2018 1:08:54 PM	4/24/2018 4:47:12 PM	poulin.lancelot@outlook.com
4	1004	4/24/2018 3:00:00 PM	4/24/2018 5:00:00 PM	4/24/2018 3:08:54 PM	4/24/2018 6:47:12 PM	brunaud.vincent@outlook.com
5	1005	4/25/2018 1:00:00 PM	4/25/2018 5:00:00 PM	4/25/2018 1:08:54 PM	4/25/2018 4:47:12 PM	poinier.colin@outlook.com
6	1006	4/25/2018 3:00:00 PM	4/25/2018 7:00:00 PM	4/25/2018 3:08:54 PM	4/25/2018 6:47:12 PM	mestre.quentin@outlook.com
7	1007	4/26/2018 1:00:00 PM	4/26/2018 5:00:00 PM	4/26/2018 1:08:54 PM	4/26/2018 4:47:12 PM	nigond.antoine@outlook.com
8	1008	4/26/2018 3:00:00 PM	4/26/2018 7:00:00 PM	4/26/2018 3:08:54 PM	4/26/2018 6:47:12 PM	humbolt.julien@outlook.com
9	1009	4/27/2018 1:00:00 PM	4/27/2018 5:00:00 PM	4/27/2018 1:08:54 PM	4/27/2018 4:47:12 PM	favre.robin@outlook.com
10	1010	4/27/2018 3:00:00 PM	4/27/2018 7:00:00 PM	4/27/2018 3:08:54 PM	4/27/2018 6:47:12 PM	jeandet.steve@outlook.com
11	1011	4/28/2018 1:00:00 PM	4/28/2018 5:00:00 PM	4/28/2018 1:08:54 PM	4/28/2018 4:47:12 PM	da.silva.thomas@outlook.com
12	1012	4/28/2018 3:00:00 PM	4/28/2018 7:00:00 PM	4/28/2018 3:08:54 PM	4/28/2018 6:47:12 PM	teller.julien@outlook.com
13	1013	4/29/2018 1:00:00 PM	4/29/2018 10:00:00 AM	4/29/2018 1:08:54 PM	4/29/2018 4:47:12 PM	debalme.mickael@outlook.com
14	1014	4/29/2018 3:00:00 PM	4/29/2018 7:00:00 PM	4/29/2018 3:08:54 PM	4/29/2018 6:47:12 PM	jupiter.nicolas@outlook.com
15	1015	4/30/2018 1:00:00 PM	4/30/2018 5:00:00 PM	4/30/2018 1:08:54 PM	4/30/2018 4:47:12 PM	dacosta.pao@outlook.com
16	1016	4/30/2018 3:00:00 PM	4/30/2018 7:00:00 PM	4/30/2018 3:08:54 PM	4/30/2018 6:47:12 PM	lecoq.leopold@outlook.com
17	1017	5/1/2018 1:00:00 PM	5/1/2018 5:00:00 PM	5/1/2018 1:08:54 PM	5/1/2018 4:08:54 PM	duparc.bastien@outlook.com
18	1018	5/1/2018 3:00:00 PM	4/1/2018 7:00:00 PM	5/1/2018 3:08:54 PM	5/1/2018 6:08:54 PM	paccot.alan@outlook.com
19	1019	5/2/2018 1:00:00 PM	5/2/2018 5:00:00 PM	5/2/2018 1:08:54 PM	5/2/2018 3:38:54 PM	perroud.theo@outlook.com
20	1020	5/2/2018 3:00:00 PM	5/2/2018 7:00:00 PM	5/2/2018 3:08:54 PM	5/2/2018 6:26:54 PM	rteuwski.valentin@outlook.com
21	4681	5/9/2018 4:00:00 PM	5/9/2018 11:00:00 PM	5/9/2018 4:07:18 PM	5/9/2018 4:07:21 PM	jeandet.steve@outlook.com
22	8495	5/9/2018 4:00:00 PM	5/9/2018 7:00:00 PM	5/9/2018 4:07:26 PM	5/9/2018 4:07:29 PM	da.silva.thomas@outlook.com
23	4851	5/9/2018 4:00:00 PM	5/9/2018 5:00:00 PM	5/9/2018 4:07:30 PM	5/9/2018 4:07:33 PM	teller.julien@outlook.com
24	7856	5/9/2018 4:00:00 PM	5/9/2018 9:00:00 PM	5/9/2018 4:07:22 PM	5/9/2018 4:07:29 PM	debaime.mickael@outlook.com
48	9581	5/22/2018 2:00:00 PM	5/22/2018 10:00:00 PM	5/22/2018 2:53:12 PM	5/22/2018 2:58:12 PM	jeandet.steve@outlook.com
58	6695	5/23/2018 7:00:00 PM	5/23/2018 8:00:00 PM	5/23/2018 7:00:00 PM	5/23/2018 4:16:27 PM	poulin.lancelot@outlook.com
63	9186	5/24/2018 2:00:00 PM	5/24/2018 2:45:00 PM	5/24/2018 2:30:00 PM	5/24/2018 2:45:00 PM	poulin.lancelot@outlook.com
75	1527	5/24/2018 3:00:00 PM	5/24/2018 10:00:00 PM			nigond.antoine@outlook.com
76	7871	5/24/2018 3:00:00 PM	5/24/2018 10:00:00 PM			nigond.antoine@outlook.com
77	3952	5/24/2018 3:00:00 PM	5/24/2018 10:00:00 PM			nigond.antoine@outlook.com
78	9252	5/24/2018 3:00:00 PM	5/24/2018 10:00:00 PM			nigond.antoine@outlook.com
79	2945	5/24/2018 3:00:00 PM	5/24/2018 10:00:00 PM			nigond.antoine@outlook.com
80	9463	5/27/2018 2:30:00 PM	5/27/2018 6:00:00 PM			poinier.colin@outlook.com

Dans celui-ci je peux visualiser la totalité des réservations en fonction de la date. Les radios boutons correspondent à différents temps pour la réservation (une journée, une semaine, un mois) en sélectionnant l'un d'entre eux nous verrons afficher les réservations en fonction de la date souhaitée. Cette fonction permet de visualiser plus facilement les réservations dans des durées plus courtes.

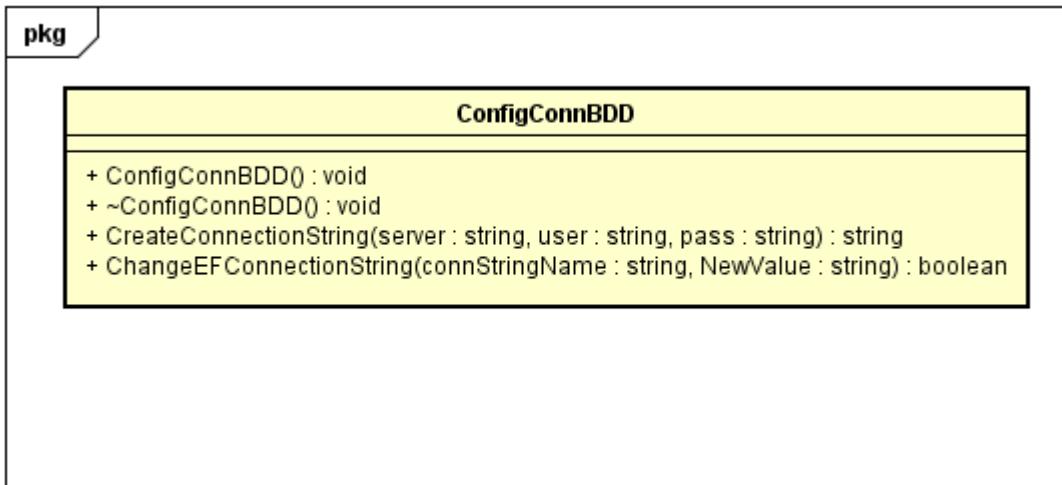
Le dernier onglet est l'onglet configuration :

The screenshot shows a software interface titled 'PROJET_LIFI_V1'. The 'CONFIGURATION' tab is selected. It contains fields for 'Adresse' (mysql-projetlifi.alwaysdata.net), 'Login' (156739), and 'Password' (projetlifi). Below these is a large 'ENREGISTRER' button.

Il est séparé en deux parties distinctes. Une première qui correspond à la configuration de la base de données et la deuxième qui est la configuration socket.

Dans ce dernier il y a la possibilité de se connecter à une base de données en insérant son adresse avec le login et le mot de passe de la base mais aussi la communication Socket et la possibilité d'envoyer et de réception de messages.

Pour ce qui est de la configuration de la connexion à la base de données je vais vous présenter le diagramme de classe de la classe **ConfigConnBDD** :



La fonction **CreateConnectionString()**

Pour ce qui est de l'aperçu visuel, j'enregistre automatiquement à chaque changement les données pour la connexion. Les informations dans les textboxes sont enregistrer dans le fichier **AppConfig** pour qu'elle s'affiche à chaque nouveau lancement de l'application. Voilà comment je récupère ces données et comment je gère le stockage :

```

// Récupérer les informations du fichier de config
string recupererinfos = ConfigurationManager.ConnectionStrings["Entities"].ConnectionString.ToString();
string[] str = recupererinfos.Split(';'); // On découpe la chaîne de caractère dans un tableau par le caractère ;

tb_adresse.Text = str[3].Remove(0, 7); // Représente la ligne 3
tb_login.Text = str[4].Remove(0, 8); // Représente la ligne 4
tb_passwordconfig.Text = str[5].Remove(0, 9); // Représente la ligne 5

```

Je crée tout d'abord une variable qui récupère la chaîne de connexion (ConnectionString en anglais) et j'utilise la fonction **Split()** pour découper comme je le souhaite cette chaîne à l'aide du caractère ; .

Ce découpage me permet de sélectionner ligne par ligne ce que je souhaite modifier, enregistrer supprimer... Str[x], la valeur x correspond à la ligne que je souhaite récupérer pour effectuer des modifications dessus. Cette valeur n'est pas choisie aléatoirement elle correspond à la valeur dans le fichier App.config. Voici les lignes de 1 à 6 du fichier de config, la ligne numéro 3 correspond à l'adresse de la base de données, la ligne numéro 4 au login et la ligne numéro 5 au mot de passe.

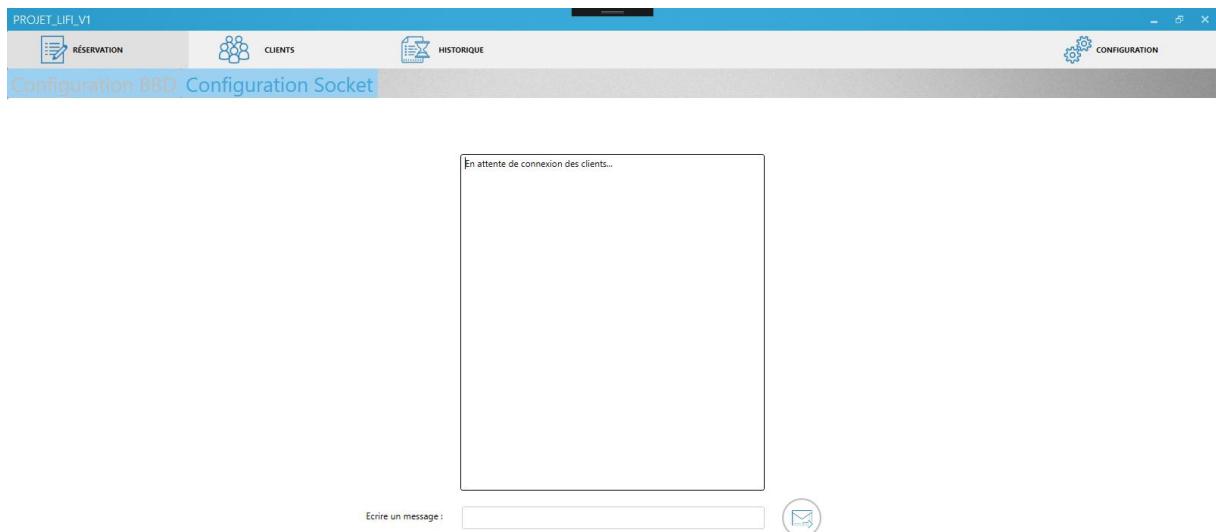
```
<add name="Entities" connectionString=
      "metadata=res://*/DAL.SmartParking_Li-fi.csdl|res://*/DAL.SmartParking_Li-fi.
      provider=MySql.Data.MySqlClient;provider connection string=";
      server=mysql-projetlifi.alwaysdata.net;
      user id=156739;
      password=projetlifi;
      database=projetlifi_bdd"; providerName="System.Data.EntityClient" />
```

La fonction **Remove()** supprimer les caractère qui me sont inutiles. Pour mieux vous expliquer je vais prendre l'exemple de la première ligne :

```
tb_adresse.Text = str[3].Remove(0, 7); // Représente la ligne 3
```

Je souhaite récupérer uniquement l'adresse de la base de données qui correspond à **mysqlprojetlifi.alwaysdata.net** je n'ai donc pas besoin des 7 premiers caractères **server=** avec la fonction **Remove()** je les supprime dans ma récupération et retient uniquement l'adresse en question.

Partie communication socket :



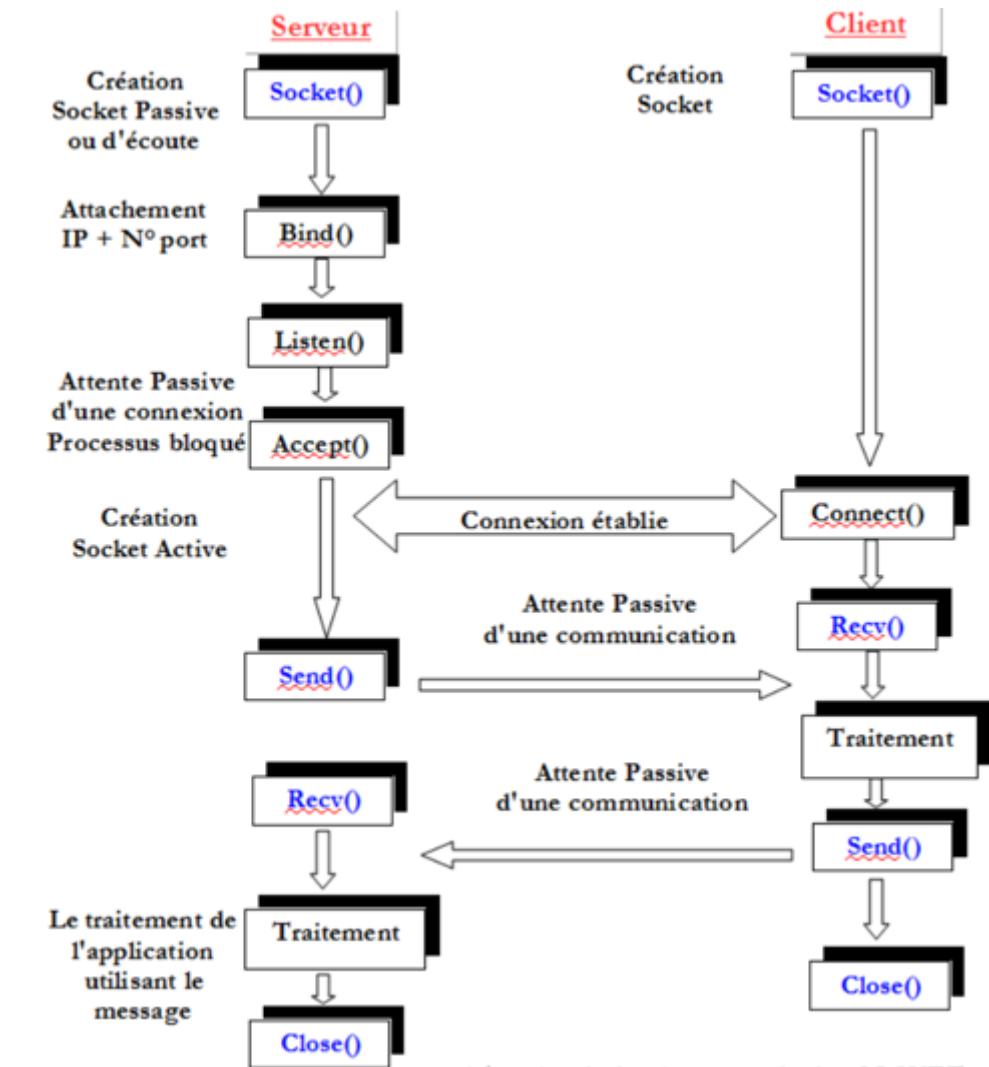
Comme il est imposé dans mon cahier des charges j'ai dû travailler avec les sockets en asynchrone. Ma tâche consiste à développer un serveur Socket en asynchrone pour communiquer avec les capteurs de **la partie Carte Embarquée**. Je vais commencer par vous expliquer le principe des Sockets et ensuite celui des Sockets asynchrone.

Les Sockets servent à communiquer entre deux hôtes appelés Client / Serveur (Il peut y avoir plusieurs clients) à l'aide d'une adresse IP et d'un port. Ces Sockets permettront de gérer des flux entrant et sortant afin d'assurer une communication entre les deux de manière fiable à l'aide du protocole **TCP/IP** ou de manière non fiable mais plus rapide à l'aide du protocole **UDP**. Les langages qui utilisent les sockets sont :

- C
- C++

- C#
- PHP
- Action Script
- Erlang
- Et bien d'autres.

Pour expliquer plus en détail le déroulement de la communication socket je vais vous présenter un schéma de déroulement du processus :



Comme expliqué au-dessus, pour une communication socket il faut un serveur et un ou plusieurs clients. Je vais vous prendre l'exemple d'une communication simple avec un seul client pour vous expliquer ce schéma. Il faut tout d'abord créer un socket avec la déclaration suivante :

```
sock = socket(AF_INET, SOCK_STREAM, 0);
```

AF_INET est la famille de protocole utilisée, **SOCK_STREAM** est le type de service, le **0** correspond au protocol.

La fonction qui suit est au niveau du serveur avec la fonction **Bind()**. Elle permet de définir l'adresse IP et le port pour communiquer entre le client et le serveur.

Le serveur passe ensuite en mode « écoute » ce qui signifie qu'il attend qu'il client se connecte et le processus est bloqué tant qu'aucun client ne se connecte.

Les fonctions **Accept()** et **Connect()** se font en simultanée, d'un côté le client effectue la fonction **Connect()** et de l'autre le serveur effectue la fonction **Accept()** pour établir la connexion entre les deux.

Dès que la connexion est établi entre le serveur et le client, l'échange de données est possible avec la fonction **Send()** et **Recv()** qui permettent d'envoyer et recevoir des messages.

Quand il n'y a plus besoin de communication entre le serveur et le client l'un comme l'autre ils utilisent la fonction **Close()** qui ferment le socket.

Pour ma part, j'ai décidé d'utiliser le protocole TCP/IP car je n'ai pas besoin d'envoyer des paquets de données de manières très rapide mais j'ai besoin qu'elles soient sécurisées. Pour travailler avec les sockets asynchrones en C# WPF je dois ajouter les usings suivant :

```
// Ajout pour les sockets
using System.IO;
using System.Net;
using System.Net.Sockets;
using System.Threading;
```

Pour l'appelle de mes fonctions je suis une structure bien particulière. En premier lieu j'appelle la fonction **Start()** qui initialise le socket avec ses paramètres comme son type d'adresse acceptées, le port sur lequel il va écouter... . Ensuite j'appelle la fonction **BeginAccept()** qui va commencer à accepter les clients qui cherche à se connecter à mon serveur. Je délimite le nombre maximum de clients qui peuvent se connecter à l'aide d'une liste de clients. C'est la fonction **EndAcceptTCPClient()** qui clôtures les connexions. Ces deux fonctions sont appelées au chargement que la grille ce qui me permet d'écouter directement quand mon application est ouverte je n'ai pas besoin d'appuyer sur un bouton pour mettre le serveur en écoute.

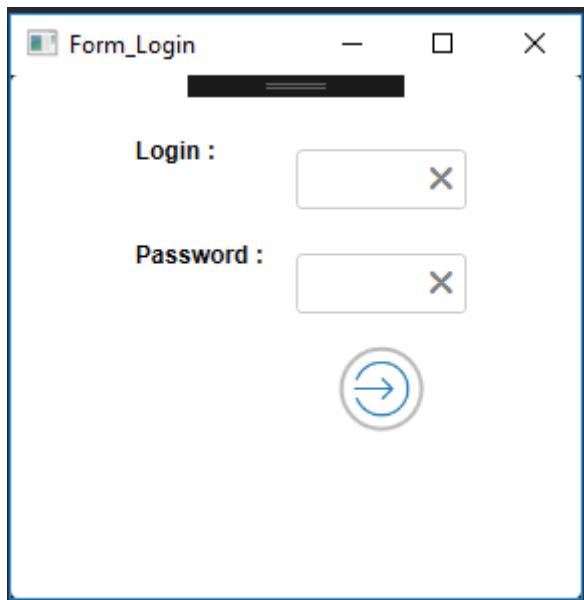
Une fois les clients connectés je peux échanger avec eux grâce à la fonction **Write()** qui envoie les données et la fonction **Read** qui les lit. Pour recevoir j'utilise la fonction **ClientReceiveData()**.

```
// client receiveData
public void ClientReceiveData(object client)
{
    int ClientId = (int)client; // Initialisation d'un client par l'objet client
    int Clientidx = ClientId - 1; // ID d'un client
    int bufferSize; // Initialisation d'un buffer
    byte[] DataBuffer; // Initialisation des données d'un buffer

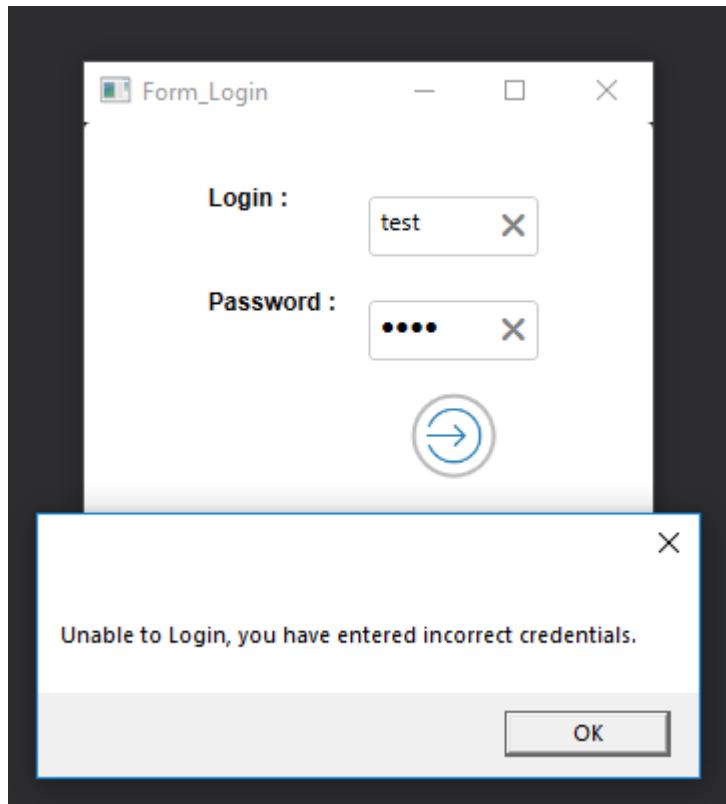
    while (true) // Réalisation d'un try catch quand je reçois des données
    {
        try
        {
            DataBuffer = new byte[1048]; // 1048 = valeur maximale du message
            // Lecture des données
            bufferSize = L_stream[Clientidx].Read(DataBuffer, 0, DataBuffer.Length);
            data = System.Text.Encoding.ASCII.GetString(DataBuffer, 0, 12);
            // Affichage des données dans une MessageBox
            MessageBox.Show(Clientidx.ToString() + ":" + data);
            if (OnReceiveData != null) OnReceiveData(ClientId, DataBuffer);
        }
        catch (Exception e) // Affichage des erreurs
        {
            MessageBox.Show(e.ToString());
        }
    }
}
```

Maintenant que je vous ai présenté la totalité des onglets de mon applications ainsi que ces fonctionnalités je vais pour présenter l'authentification à la base de données en se connectant en tant qu'admin.

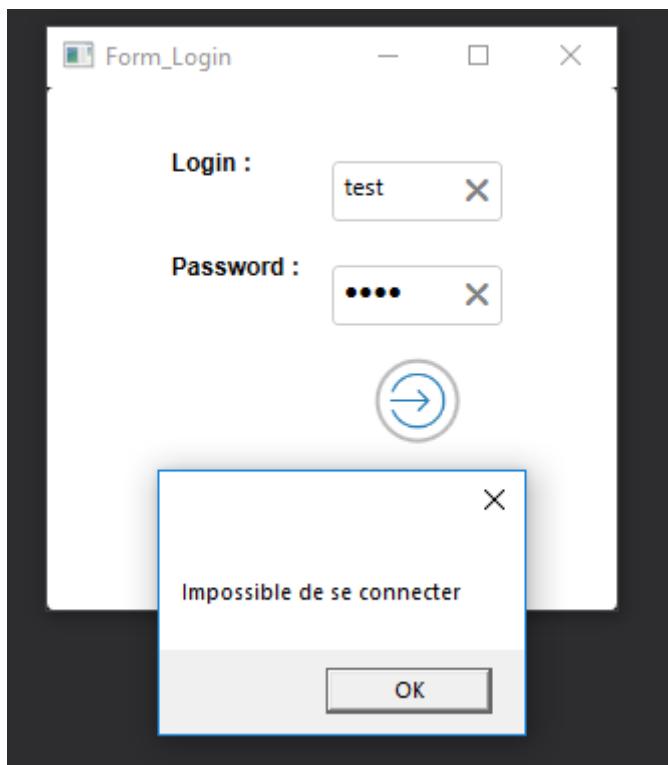
Au lancement de l'application une fenêtre s'ouvre en me demandant de me connecter. Je dois saisir l'identifiant et le mot de passe du compte administrateur pour accéder à mon application. Voici un aperçu graphique de cette fenêtre :



C'est à ce moment que je dois saisir l'identifiant et le mot de passe, cependant j'ai géré les erreurs, si l'identifiant ou le mot de passe est incorrecte l'application m'affiche l'erreur suivante :



Une fois que vous avez cliquez sur le bouton **OK** un nouveau message d'erreur s'affiche pour vous informer qu'il est impossible de se connecter à la base de données :



3.3.5 Conclusion

Pour conclure, ma partie dans le projet est terminé, j'ai terminé le design de mon application, et établie la connexion à la base de données. Pour ce qui de la communication socket, je communique comme il faut avec la partie Carte Embarquée. Toutes les fonctionnalités nécessaires pour mes DAOs sont terminées.

Par la suite, plusieurs améliorations sont possibles pour développer d'avantages le projet, pour commencer une partie reporting pour imprimer les différentes réservations et pouvoir faire des graphiques pour voir les fréquences de réservation.

3.4 Partie SYNCHRONISER web service – base de données

Partie effectuée par Thomas Da Silva

Dans le projet Smart Parking Li-Fi 2018, je dois faire deux parties distinctes : la base de données, ainsi que la partie web service qui fait le lien entre le smartphone et la base de données.

Les deux parties seront décrites séparément.

3.4.1 Synoptique de ma partie

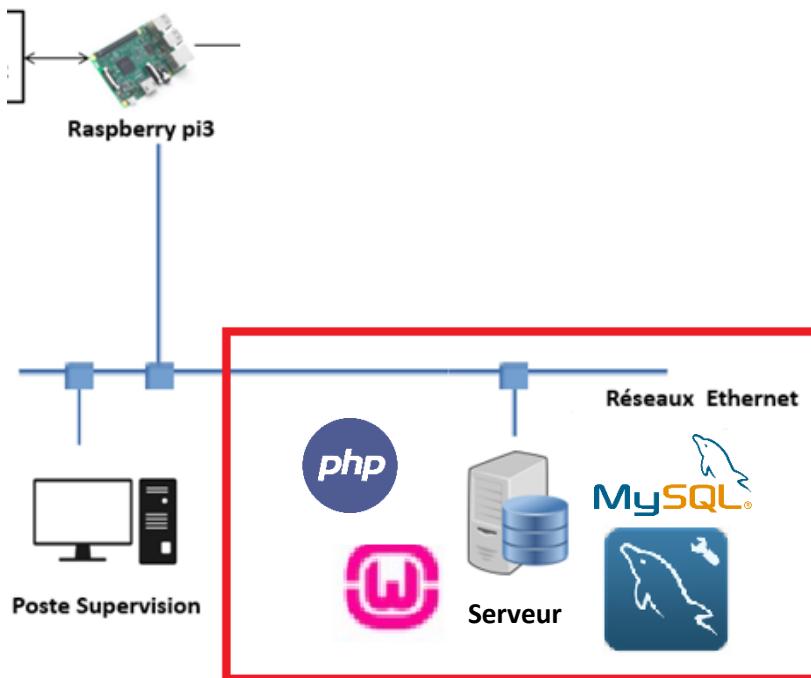


Figure 55: Structure de la partie SYNCHRONISER web service - BDD

3.4.2 Diagramme de Gantt

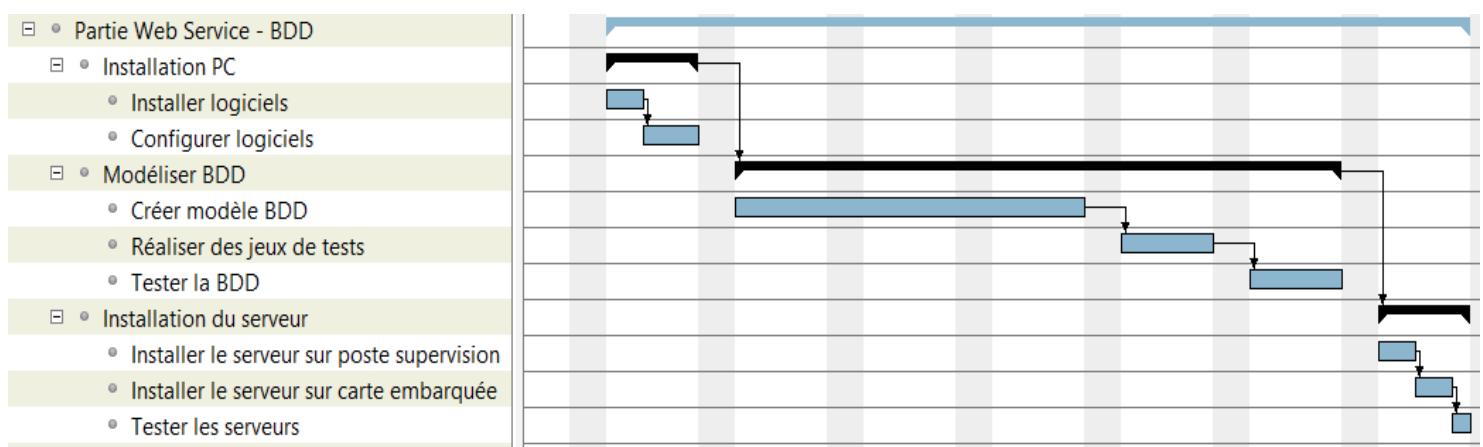


Figure 56: Diagramme de Gantt de la partie SYNCHRONISER

3.4.3 Travail réalisé

3.4.3.1 Partie base de données

Une base de données permet de stocker et de retrouver l'intégralité de données brutes ou d'informations en rapport avec un thème ou une activité ; celles-ci peuvent être de natures différentes (texte, images...) et plus ou moins reliées entre elles. Ces informations sont très structurées, et la base est localisée dans un même lieu et sur un même support, généralement informatisé.

Tout d'abord, j'ai créé une première version du modèle de base de données sur MySQL Workbench. Le tutoriel d'installation de MySQL Workbench est en annexe.

Il a fallu réfléchir aux tables que nous aurions besoin, puis aux éléments à mettre à l'intérieur, avec les types de données. Enfin, les liaisons ont été ajoutées.

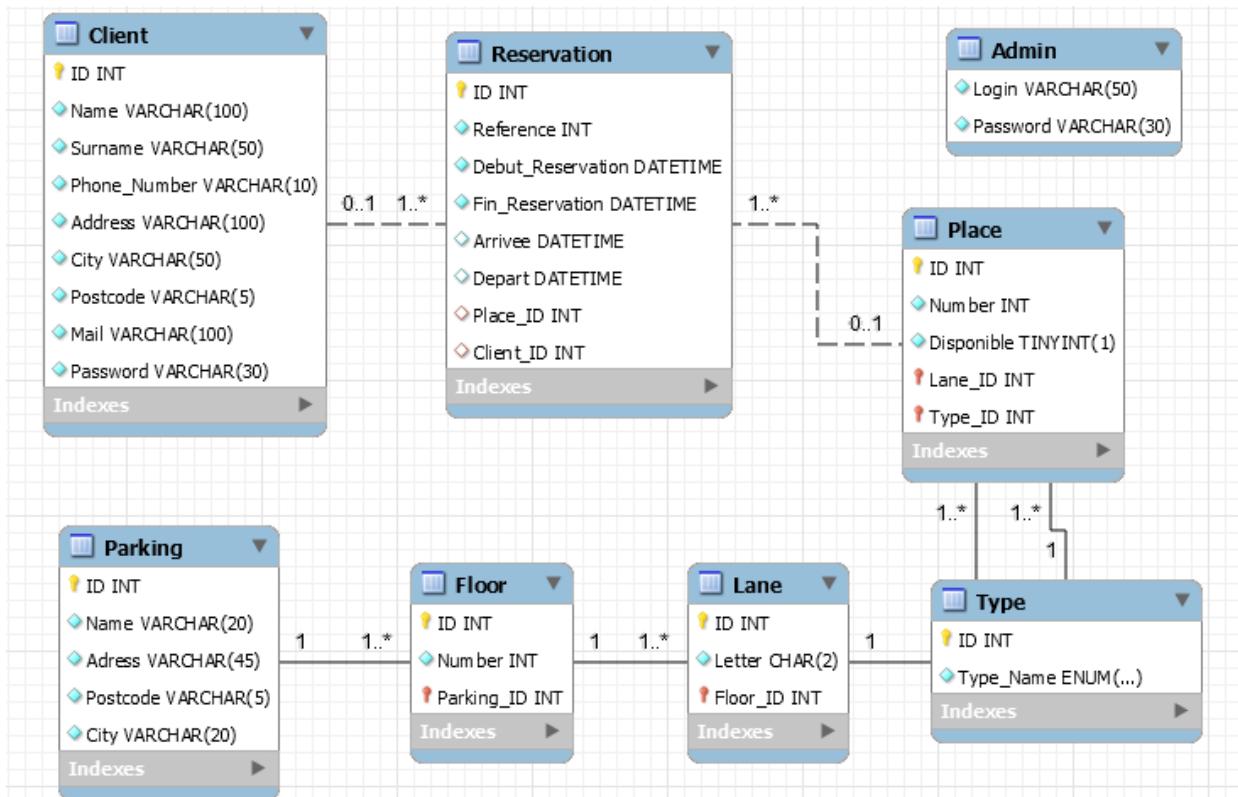


Figure 57: Modèle MWB de la base de données

Dans la table « Place », le champ « Disponible » aurait dû être un booléen (BOOL), c'est-à-dire une variable qui ne peut prendre que deux valeurs distinctes (0 ou 1) mais ce type n'était pas pris en charge par MySQL Workbench, je l'ai donc remplacé par un entier le plus petit possible (TINYINT).

Ensuite, j'ai fait un jeu de tests, pour chaque table, toujours sous MySQL Workbench.

	ID	Reference	Debut_Reservation	Fin_Reservation	Arrivée	Depart	Place_ID	Client_ID
▶	1	1001	2018-03-01 14:00:00	2018-03-01 15:30:00	2018-03-01 14:07:25	2018-03-01 15:24:48	1	1
	2	1002	2018-03-01 16:15:00	2018-03-01 18:00:00	2018-03-01 16:38:53	2018-03-01 18:07:52	2	2
	3	1003	2018-03-01 13:00:00	2018-03-01 15:00:00	2018-03-01 13:00:24	2018-03-01 14:06:58	5	3
	4	1004	2018-03-02 09:30:00	2018-03-02 11:00:00	2018-03-02 09:17:42	2018-03-02 10:48:05	12	6
	5	1005	2018-04-01 15:00:00	2018-04-01 17:00:00	NULL	NULL	NULL	1
	6	1006	2018-04-01 16:00:00	2018-04-01 17:00:00	NULL	NULL	NULL	7
	7	1007	2018-05-01 12:15:00	2018-05-01 14:45:00	NULL	NULL	NULL	9
	8	1008	2018-04-01 15:00:00	2018-04-02 10:30:00	NULL	NULL	NULL	4

Figure 58: Jeux de test de la table "Réervation"

Une fois le jeu de tests terminé, j'ai mis la base de données sur le serveur Wamp qui est une distribution. Le tutoriel d'installation de Wamp Server est en annexe, et le tutoriel pour transférer la base de données sur le serveur Wamp en annexe. Une fois ces étapes terminées, j'ai commencé à créer les requêtes pour la base de données sur MySQL Workbench, afin de pouvoir la tester directement.

Figure 59: Script de test MySQL



```

1 SET SQL_SAFE_UPDATES = 0;
2
3 /* client */
4 select * from client where surname="vincent" or name="brunaud";
5
6 insert into client (name, surname, phone_number, address, city, postcode, mail, password) values ("BETTEGA", "Océane", "0650846241", "Rue centrale", "Aix", "73000", "bettega.oceane@free.fr", "ob");
7
8 update client set surname="vincent", phone_number="0777777777", address="Rue de loverchy", city="Bons", postcode="74750", mail="brunaud@outlook.com", password="vb" where name="brunaud";
9
10 /* historique */
11 select * from reservation;

```

Le projet se développant, j'ai dû à plusieurs reprises faire des modifications de la base de données, car il fallait ajouter ou modifier des paramètres en fonction des besoins de l'équipe. Nous avons donc à ce jour une version améliorée de la base de données.

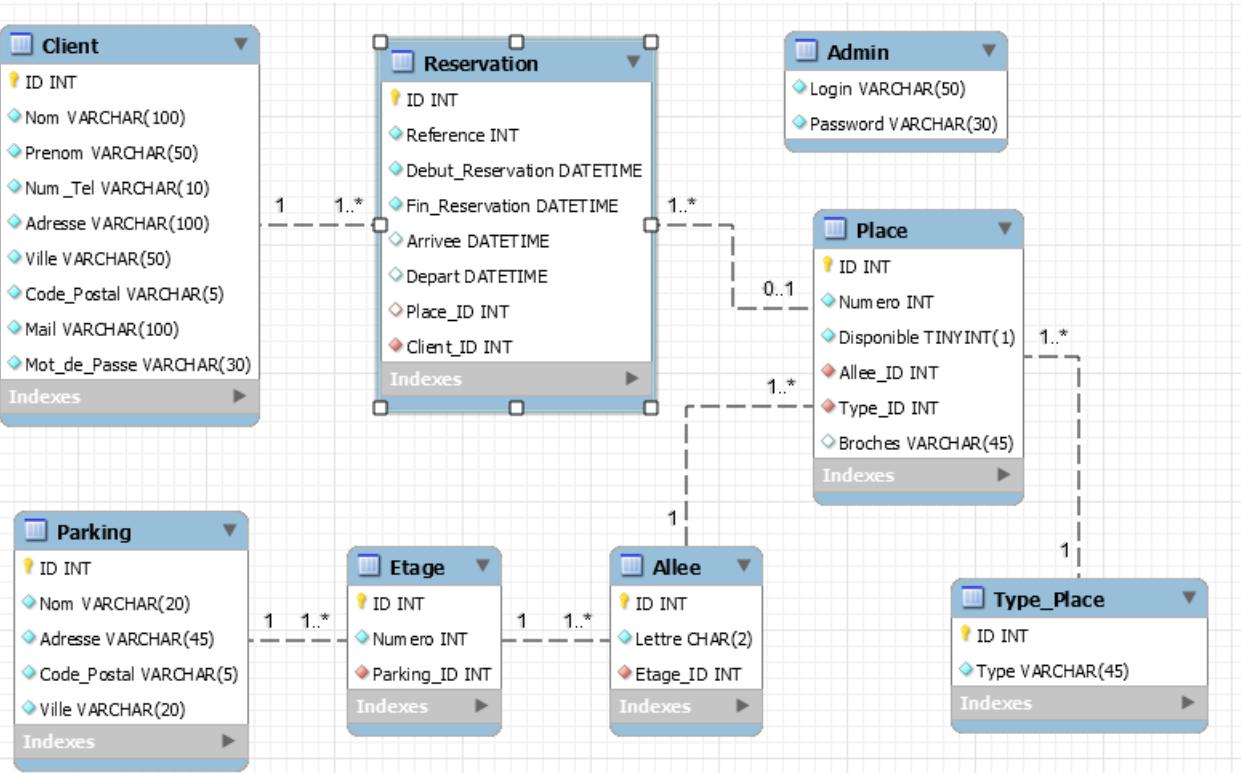


Figure 60: Seconde version du modèle MWB de la base de données

Les jeux de tests ont également été modifiés ou complétés car certains posaient problèmes pour l'avancement de la partie smartphone. Ces derniers ont été longuement réfléchis, afin d'être le plus juste et se rapprochant le plus possible de la réalité.

	ID	Reference	Debut_Reservation	Fin_Reservation	Arrivee	Depart	Place_ID	Client_ID
1	1001		2018-03-08 13:00:00	2018-03-08 17:00:00	2018-03-08 13:08:54	2018-03-08 16:47:12	1	1
2	1002		2018-03-08 15:00:00	2018-03-08 19:00:00	2018-03-08 15:08:54	2018-03-08 18:47:12	2	2
3	1003		2018-03-09 13:00:00	2018-03-09 17:00:00	2018-03-09 13:08:54	2018-03-09 16:47:12	3	3
4	1004		2018-03-09 15:00:00	2018-03-10 17:00:00	2018-03-09 15:08:54	2018-03-09 18:47:12	4	4
5	1005		2018-03-10 13:00:00	2018-03-10 17:00:00	2018-03-10 13:08:54	2018-03-10 16:47:12	5	5
6	1006		2018-03-10 15:00:00	2018-03-10 19:00:00	2018-03-10 15:08:54	2018-03-10 18:47:12	6	6
7	1007		2018-03-11 13:00:00	2018-03-11 17:00:00	2018-03-11 13:08:54	2018-03-11 16:47:12	7	7
8	1008		2018-03-11 15:00:00	2018-03-11 19:00:00	2018-03-11 15:08:54	2018-03-11 18:47:12	8	8
9	1009		2018-03-12 13:00:00	2018-03-12 17:00:00	2018-03-12 13:08:54	2018-03-12 16:47:12	9	9
10	1010		2018-03-12 15:00:00	2018-03-12 19:00:00	2018-03-12 15:08:54	2018-03-12 18:47:12	10	10
11	1011		2018-03-13 13:00:00	2018-03-13 17:00:00	2018-03-13 13:08:54	2018-03-13 16:47:12	11	11
12	1012		2018-03-13 15:00:00	2018-03-13 19:00:00	2018-03-13 15:08:54	2018-03-13 18:47:12	12	12
13	1013		2018-03-14 13:00:00	2018-03-15 10:00:00	2018-03-14 13:08:54	2018-03-14 16:47:12	13	13
14	1014		2018-03-14 15:00:00	2018-03-14 19:00:00	2018-03-14 15:08:54	2018-03-14 18:47:12	14	14
15	1015		2018-03-15 13:00:00	2018-03-15 17:00:00	2018-03-15 13:08:54	2018-03-15 16:47:12	15	15
16	1016		2018-03-15 15:00:00	2018-03-15 19:00:00	2018-03-15 15:08:54	2018-03-15 18:47:12	16	16
17	1017		2018-03-16 13:00:00	2018-03-16 17:00:00	2018-03-16 13:08:54	NULL	17	17
18	1018		2018-03-16 15:00:00	2018-03-16 19:00:00	2018-03-16 15:08:54	NULL	18	18
19	1019		2018-03-17 13:00:00	2018-03-17 17:00:00	2018-03-17 13:08:54	NULL	19	19
20	1020		2018-03-17 15:00:00	2018-03-17 19:00:00	2018-03-17 15:08:54	NULL	20	20
21	1021		2018-04-01 13:00:00	2018-04-01 17:00:00	NULL	NULL	NULL	1
22	1022		2018-04-01 15:00:00	2018-04-01 19:00:00	NULL	NULL	NULL	2
23	1023		2018-04-02 13:00:00	2018-04-02 17:00:00	NULL	NULL	NULL	3
24	1024		2018-04-02 15:00:00	2018-04-02 19:00:00	NULL	NULL	NULL	4
25	1025		2018-04-03 13:00:00	2018-04-03 17:00:00	NULL	NULL	NULL	5
26	1026		2018-04-03 15:00:00	2018-04-03 19:00:00	NULL	NULL	NULL	6
27	1027		2018-04-04 13:00:00	2018-04-04 17:00:00	NULL	NULL	NULL	7
28	1028		2018-04-04 15:00:00	2018-04-04 19:00:00	NULL	NULL	NULL	8
29	1029		2018-04-05 13:00:00	2018-04-05 17:00:00	NULL	NULL	NULL	9
30	1030		2018-04-05 15:00:00	2018-04-05 19:00:00	NULL	NULL	NULL	10
31	1031		2018-04-06 13:00:00	2018-04-06 17:00:00	NULL	NULL	NULL	11
32	1032		2018-04-06 15:00:00	2018-04-06 19:00:00	NULL	NULL	NULL	12

Figure 61: Second jeux de test de la table "Réservation"

Pour faciliter le travail de chacun et que la base de données puisse être accessible de n'importe où et n'importe quand, cette dernière a été mise en ligne sur un serveur gratuit. J'ai choisi <https://www.alwaysdata.com/fr/> car ce dernier possède les fonctionnalités nécessaires pour héberger la base de données et la rendre accessible depuis l'application Supervision, qui n'utilise pas le web service pour y accéder. Retrouvez en annexe 11 le tutoriel pour mettre en ligne la base de données (sur alwaysdata.com).

BASE DE DONNÉES PROJETLIFI_BDD

Informations

Nom*

Le nom doit impérativement commencer par : projetlifi_

Permissions

156739 tous les droits lecture seule aucun droit

VALIDER

Figure 62: Gestionnaire de bases de données en ligne

Table	Action	Lignes
admin	Parcourir Structure Rechercher Insérer Vider Supprimer	1
allee	Parcourir Structure Rechercher Insérer Vider Supprimer	2
client	Parcourir Structure Rechercher Insérer Vider Supprimer	28
etage	Parcourir Structure Rechercher Insérer Vider Supprimer	1
parking	Parcourir Structure Rechercher Insérer Vider Supprimer	1
place	Parcourir Structure Rechercher Insérer Vider Supprimer	4
reservation	Parcourir Structure Rechercher Insérer Vider Supprimer	29
type_place	Parcourir Structure Rechercher Insérer Vider Supprimer	3
8 tables	Somme	63

Figure 63: La base de données sur le serveur

3.4.3.2 Web service

Un web service est un mécanisme de communication à distance entre plusieurs applications (qui peuvent être sur différentes plateformes et sous différents langages).

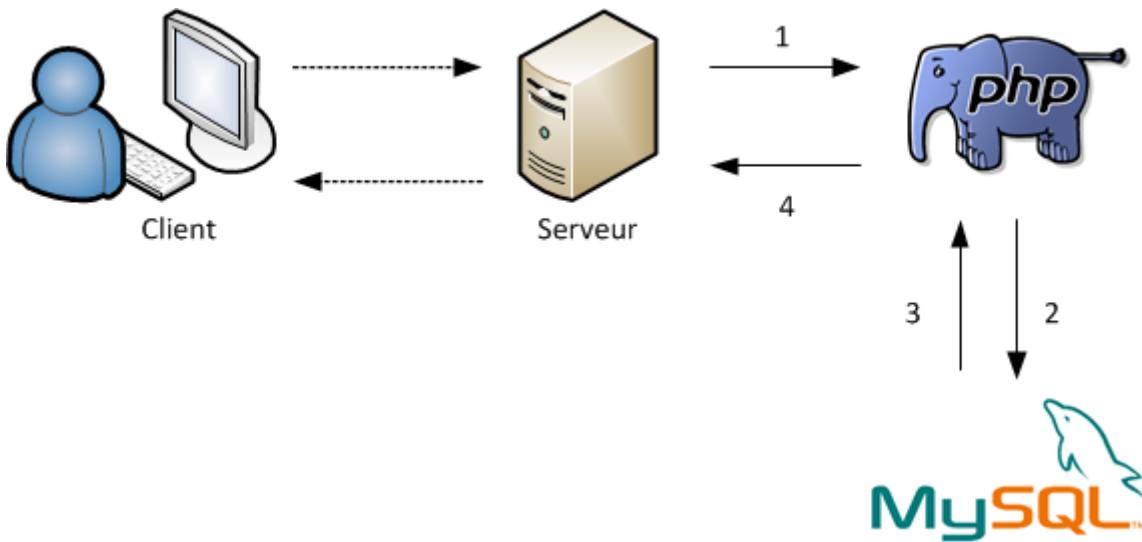


Figure 64: Communication Client <-> Web Service <-> BDD

Le PHP est le langage utilisé pour mon web service. Ce dernier fait l'intermédiaire entre le client (ici, l'application smartphone) et la base de données MySQL.

Après avoir revu mes cours sur le PHP, j'ai commencé le web service en codant le fichier de connexion.

```

1 <?php
2 /* DA SILVA Thomas, 06/03/18 : WebService Projet Lifi */
3
4 $AdresseBDD = 'localhost';
5 $User = 'root';
6 $Password = '';
7 $BDDName = 'smartparkinglifi_2017-2018_v1.0';
8
9 $Connect = new mysqli($AdresseBDD, $User, $Password, $BDDName) or die(mysqli_error())
10 ?>

```

Figure 65: Fichier de connexion du web service

Puis, j'ai ensuite codé et commenté tous les autres fichiers du web service. Ces derniers permettent à l'application smartphone d'accéder et de modifier le contenu de la base de données.



Figure 66: Fichiers du web service PHP

Voici les fonctions des différents fichiers du web service :

 **Connect** : Permet la connexion à la base de données, c'est le fichier le plus important, car sans lui, aucun des autres ne peuvent fonctionner.

 **CountDispoPlace** : Permet de compter le nombre de places disponibles durant un créneau horaire choisi par l'application smartphone.

 **DeleteReserv** : Permet à un utilisateur de l'application smartphone de supprimer une réservation qu'il a effectuée.

 **InsertReserv** : Permet à un utilisateur de l'application smartphone de faire une réservation au parking.

 **Login** : Permet à un utilisateur de se connecter à son compte sur l'application, en rentrant son mail et son mot de passe.

 **SelectParking** : Renvoie la liste des parkings.

 **SelectPlace** : Indique à l'utilisateur la place qui lui a été attribuée lors de son arrivée au parking.

 **SelectReserv** : Permet à l'utilisateur de voir l'historique de ses réservations.

 **SelectReservInvite** : Permet à un utilisateur n'ayant pas de compte sur l'application de voir les détails de l'une de ses dernières réservations.

 **UpdateReserv** : Permet à un utilisateur de l'application de modifier sa réservation.

Le web service a également été mis en ligne afin d'être accessible depuis n'importe où et n'importe quand. Ce dernier a été mis en ligne sur un serveur gratuit. J'ai choisi <https://fr.000webhost.com/> car ce dernier est simple d'utilisation, et il n'a pas été possible d'utiliser le même hébergeur que la base de données. Retrouvez en annexe 12 le tutoriel pour mettre en ligne le web service (sur fr.000webhost.com).

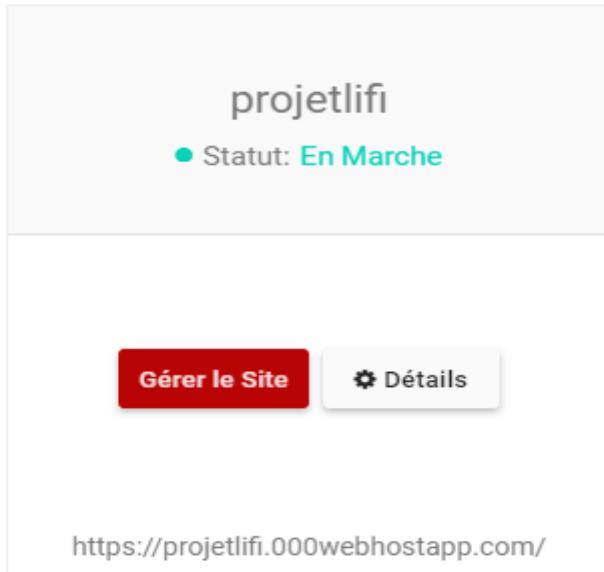


Figure 67:Statut et adresse du web service

Hooray, your free website has been started!

projetlifi.000webhostapp.com

You see this page because your website doesn't have "**index.php**" or "**index.html**" file in **public_html** folder.
[Create index file](#)
Below you can see your current files in **public_html** folder.

File	Size	Last Modified
Connect.php	1KB	May 29 2018 02:20:17 PM
CountDispoPlace.php	1KB	May 07 2018 12:40:40 PM
DeleteReserv.php	1KB	May 07 2018 12:40:40 PM
InsertReserv.php	1KB	May 29 2018 02:23:24 PM
Login.php	1KB	May 07 2018 12:40:40 PM
SelectParking.php	1KB	May 07 2018 12:40:40 PM
SelectPlace.php	1KB	May 07 2018 12:40:40 PM
SelectReserv.php	1KB	May 07 2018 12:40:40 PM
SelectReservInvite.php	1KB	May 07 2018 12:40:40 PM
UpdateReserv.php	1KB	May 07 2018 12:40:40 PM

Figure 68:Fichiers en ligne

Lors de l'insertion d'une réservation, une référence unique et aléatoire est générée automatiquement.

<input type="button" value="←"/>	<input type="button" value="→"/>	ID	Reference	Debut_Reservation	Fin_Reservation	Arrivée	Départ	Place_ID	Client_ID
<input type="checkbox"/>	<input type="button" value="Éditer"/>	<input type="button" value="Copier"/>	<input type="button" value="Supprimer"/>	1	6813	2018-04-23 13:00:00	2018-04-23 17:00:00	2018-04-23 13:08:54	2018-04-23 16:47:12

Pour cela j'ai utilisé la fonction **rand(1000, 9999)** qui permet de générer un nombre aléatoire compris entre 1000 et 9999 inclus.

Définition de la fonction **rand()** :

- **rand()** retourne un nombre pseudoaléatoire entre 0 et 1. Pour obtenir un nombre aléatoire entre deux valeurs incluses, il faut utiliser **rand(min, max)**.

Or, cette fonction ne garantit pas que ce nombre sera unique et un nombre peu très bien tomber plusieurs fois. Pour pallier ce problème, j'ai tout d'abord réalisé un algorithme.

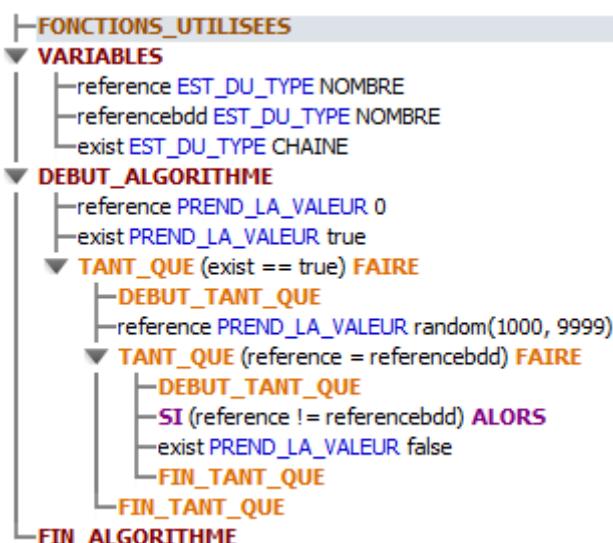


Figure 69: Algorithme pour générer nombr unique ET aléatoire

Une fois prêt, je l'ai codé et intégré au fichier pour insérer les réservations.

```

$reference = 0;
$exist = true;

while ($exist == true)
{
    $reference = rand(1000, 9999);
    $req = $connect->query("select count(*) from reservation where Reference =". $reference .";");
    while ($r = mysqli_fetch_assoc($req))
    {
        if ($r["count(*)"] == 0)
        {
            $exist = false;
        }
    }
}
  
```

Figure 70: Algorithme codé en PHP

Ce dernier compte le nombre de références qui sont égale à la référence générée grâce à la fonction **count()** qui ici retournera 0 si la référence n'existe pas et 1 si elle existe déjà, s'il en compte une, l'algorithme en génère une nouvelle qui est elle-même comparée aux références déjà existantes dans la base de données, et ce jusqu'à en générer une qui n'existe pas.

3.4.4 Conclusion

Notre mission est de développer un projet en utilisant une technologie encore peu connue et en développement, le Li-Fi. Pour ma part, je suis en charge de la partie web service ainsi que la création d'une base de données.

La base de données peut être utilisée par mes camarades et a été modifiée par mes soins à chaque fois qu'il a été nécessaire.

Quant au web service, il est fonctionnel, toutes les requêtes ont été créées, ainsi que les fonctions.

Le diagramme de Gantt a été respecté, je n'ai pas pris de retard.

Les évolutions possibles sont la mise en ligne sur un seul hébergeur pour plus de facilité de gestion.

4 Conclusion générale

L'objectif de notre projet est de développer un système de parking complexe utilisant différentes technologies comme l'ultrason et le Li-Fi.

Tout d'abord, nous avons pris beaucoup de temps pour réfléchir et faire une analyse pertinente afin de ne pas modifier tout le fonctionnement du projet à chaque problème technique rencontré. Chacun d'entre nous a bien compris l'objectif et nous sommes sur la même longueur d'onde. Nous n'avons pas pris de retard et l'avancement général du projet a rapidement pris une bonne allure.

Le diagramme de Gantt et les tâches ont été respectés malgré quelques modifications du modèle de la base de données ainsi que les cas d'utilisation. Certaines solutions ont été apportées pour donner suite à des difficultés techniques rencontrées dans certaine partie mais le fonctionnement général du projet est resté le même.

Le projet est fonctionnel et l'intégralité du système a été testé sous de multiples conditions.

Table des annexes

5.1	Annexe 1 : Installation de Raspbian et de Code::Blocks	77
5.2	Annexe 2 : Installation des ports UART	79
5.3	Annexe 3 : Programme C++ pour l'envoie et la réception de données en UART.....	80
5.4	Annexe 4 : Classes C# des modèles de la base de données	81
5.5	Annexe 5 : Classe HttpRequests contenant les requêtes envoyées au web service.....	82
5.6	Annexe 6 : Programme de l'activité principale de l'application smartphone	83
5.7	Annexe 7 : Méthode permettant la réception de données Li-Fi sur smartphone	84
5.8	Annexe 8 : Programme principale de l'application de la borne Li-Fi	85
5.9	Annexe 9 : Programme C++ pour mesurer une distance	86
5.10	Annexe 10 : Câblage du test du capteur	87
5.11	Annexe 11 : Programme C++ d'initialisation WiringPI	88
5.12	Annexe 12 : Programme C++ pour détecter un véhicule	89
5.13	Annexe 13 : Programme C++ pour la connexion à la base de données	90
5.14	Annexe 14 : Programme C++ des requêtes SQL et traitement de données.....	91
5.15	Annexe 15 : Installation de MahApps.Metro et Entity Framework	92
5.16	Annexe 16 : Installation de MySQL Workbench	94
5.17	Annexe 17 : Installation de Wamp Server	96
5.18	Annexe 18 : Transférer la base de données sur le serveur MySQL (local)	99
5.19	Annexe 19 : Transférer la base de données sur le serveur en ligne.....	102
5.20	Annexe 20 : Transférer le web service sur le serveur en ligne.....	105

5 Annexes

5.1 Annexe 1 : Installation de Raspbian et de Code::Blocks

Cette notice permet d'installer une carte Raspberry Pi jusqu'au premier démarrage puis de configurer le système afin d'installer Code::Blocks.

Matériels requis :

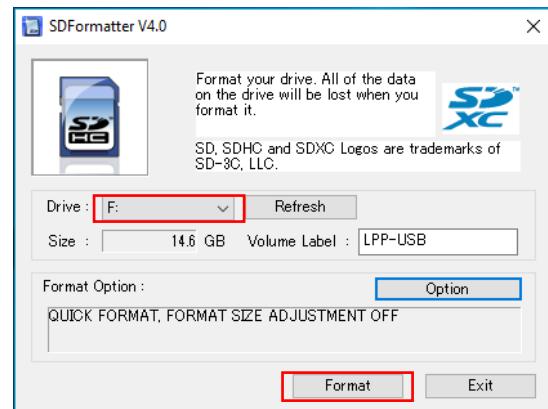
- Une carte Raspberry Pi ainsi que son câble d'alimentation
- Un clavier, une souris et un écran
- Une carte microSD d'au moins 16 Go (Pref. : Sandisk)
- Un ordinateur équipé du système d'exploitation Windows

Dans un premier temps, sur machine Windows, téléchargez :

- L'OS Linux Raspbian sur le site officiel Raspberry
<https://www.raspberrypi.org/downloads/raspbian>
- Le logiciel SDCardFormatter si vous avez déjà utilisé votre microSD :
https://www.sdcard.org/downloads/formatter_4
- Le logiciel Win32DiskImager pour déployer l'image disque :
<https://sourceforge.net/projects/win32diskimager>

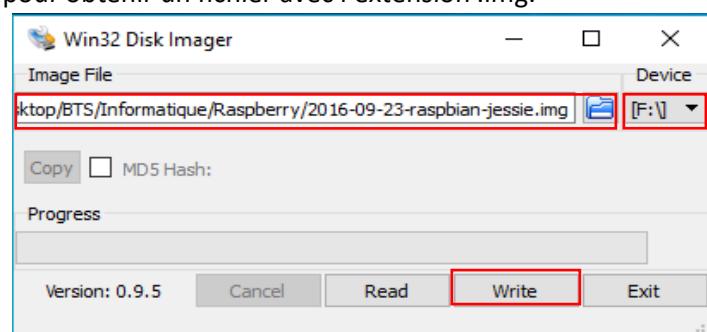
Cette étape est facultative, elle permet de formater votre carte microSD si vous l'avez déjà utilisé auparavant.

1. Lancez l'application SDCardFormatter.
2. Sélectionnez votre carte microSD (connecté à votre machine).
3. Cliquez sur Format puis patientez jusqu'à la fin du chargement.



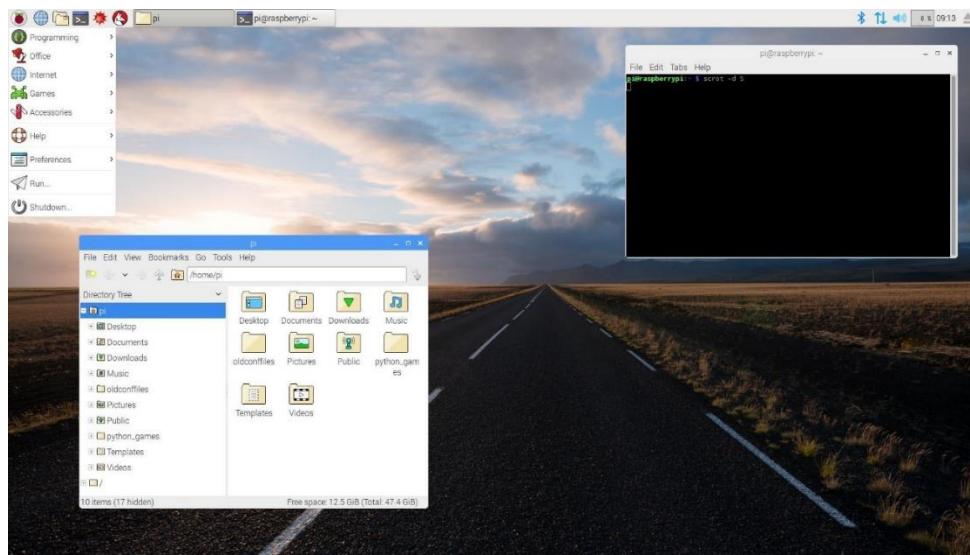
Suivez ces étapes pour déployer l'image disque de l'OS Linux Raspbian téléchargé précédemment. Il vous faut décompresser le fichier obtenu pour obtenir un fichier avec l'extension .img.

1. Lancez l'application SDCardFormatter.
2. Sélectionnez l'image disque de l'OS Linux Raspbian (.img).
4. Sélectionnez la carte microSD.
5. Appuyez sur Write et attendez la fin du chargement.



Linux Raspbian est désormais installé sur la microSD et prêt à être utilisé !

Insérez la microSD dans la carte Raspberry PI, branchez l'écran à la carte puis alimentez la carte. Vous devez normalement atterrir sur cette interface.



Après s'être connecter à un réseau en Ethernet ou Wi-Fi, vous devez faire la mise à jour de Raspbian.

Pour obtenir la liste des dernières versions des paquets installés, ouvrez le terminal et tapez

```
sudo apt-get update
```

Puis pour télécharger et mettre à jour automatiquement ces paquets tapez

```
sudo apt-get upgrade
```

Pour installer un quelconque paquet (logiciels, librairies...), il vous suffit de tapez la commande « sudo apt-get » suivis du nom, exemple pour Code::Blocks :

```
sudo apt-get codeblocks
```

Pour lancer Code::Blocks en administrateur :

```
sudo codeblocks
```

5.2 Annexe 2 : Installation des ports UART

Il faut créer un script SHELL et l'exécuter pour installer le service UART pour les broches TX/RX.

```
1 echo '[Service]
2 Environment="TERM=xterm" > /etc/systemd/system/serial-getty@ttyS0.service.d/xt
3 systemctl daemon-reload && \
4 systemctl enable serial-getty@ttyS0.service && \
5 systemctl start serial-getty@ttyS0.service && \
6 systemctl status serial-getty@ttyS0.service
7 echo '# ttyS0 - getty
8
9 start on stopped rc RUNLEVEL=[2345]
10 stop on runlevel [!2345]
11 respawn
12 exec /sbin/getty -L 115200 ttyS0 xterm' > /etc/init/ttys0.conf && \
13 start ttys0
14'
```

5.3 Annexe 3 : Programme C++ pour l'envoie et la réception de données en UART

Premièrement, on ouvre le port UARD et on le configure (lecture/écriture, 115200 baud, 8bits), ensuite on envoie « Hello World ! » en sortie (bouclé). La seconde partie est pour la réception.

```

1 #include <iostream>
2 #include <stdio.h>
3 #include <stdlib.h>
4 #include <wiringPi.h>
5 #include <unistd.h>           //Used for UART
6 #include <fcntl.h>            //Used for UART
7 #include <termios.h>          //Used for UART
8
9 using namespace std;
10
11 int main()
12 {
13     //At bootup, pins 8 and 10 are already set to UART0_TxD, UART0_RxD (ie the alt0 function) respectively
14     int uart0_filestream = -1;
15
16     //OPEN THE UART
17     uart0_filestream = open("/dev/ttyS0", O_RDWR | O_NOCTTY | O_NDELAY); //Open in non blocking read/write mode
18     if (uart0_filestream == -1)
19     {
20         //ERROR - CAN'T OPEN SERIAL PORT
21         printf("Error - Unable to open UART. Ensure it is not in use by another application\n");
22     }
23
24     //CONFIGURE THE UART
25     struct termios options;
26     tcgetattr(uart0_filestream, &options);
27     options.c_cflag = B115200 | CS8 | CLOCAL | CREAD;      //Set baud rate
28     options.c_iflag = IGNPAR;
29     options.c_oflag = 0;
30     options.c_lflag = 0;
31     tcflush(uart0_filestream, TCIFLUSH);
32     tcsetattr(uart0_filestream, TCSANOW, &options);
33
34     while(1)
35     {
36         //---- TX BYTES ----
37         unsigned char tx_buffer[20];
38         unsigned char *p_tx_buffer;
39
40         p_tx_buffer = &tx_buffer[0];
41         *p_tx_buffer++ = 'H';
42         *p_tx_buffer++ = 'e';
43         *p_tx_buffer++ = 'l';
44         *p_tx_buffer++ = 'l';
45         *p_tx_buffer++ = 'o';
46         *p_tx_buffer++ = ' ';
47         *p_tx_buffer++ = 'W';
48         *p_tx_buffer++ = 'o';
49         *p_tx_buffer++ = 'r';
50         *p_tx_buffer++ = '!';
51         *p_tx_buffer++ = ' ';
52
53         if (uart0_filestream != -1)
54         {
55             int count = write(uart0_filestream, &tx_buffer[0], (p_tx_buffer - &tx_buffer[0])); //Filestream, bytes to write
56             if (count < 0)
57             {
58                 printf("UART TX error\n");
59             }
56             else
57             {
58                 printf("Writing data success\n");
59             }
60         }
61
62
63         //---- CHECK FOR ANY RX BYTES ----
64         if (uart0_filestream != -1)
65         {
66             Read up to 255 characters from the port if they are there
67             unsigned char rx_buffer[256];
68             int rx_length = read(uart0_filestream, (void*)rx_buffer, 255); //Filestream, buffer to store in, number of bytes to read
69             if (rx_length < 0)
70             {
71                 printf("Error reading data\n");
72             }
73             else if (rx_length == 0)
74             {
75                 printf("No data to read\n");
76             }
77             else
78             {
79                 Bytes received
80                 rx_buffer[rx_length] = '\0';
81                 printf("%i bytes read : %s\n", rx_length, rx_buffer);
82             }
83         }
84
85         delay(1000);
86     }
87
88     //---- CLOSE THE UART ----
89     close(uart0_filestream);
90
91     return 0;
92 }
93
94
95
96
97
98
99

```

5.4 Annexe 4 : Classes C# des modèles de la base de données

Modèle de la table Client

```
1 // User database model
2
3 namespace PLSport.Data.Model
4 {
5     public class User
6     {
7         public int ID { get; set; }
8         public string Surname { get; set; }
9         public string Name { get; set; }
10        public string Mail { get; set; }
11        public string Password { get; set; }
12        public string Gender { get; set; }
13        public System.DateTime Birthday { get; set; }
14    }
15 }
16
```

Modèle de la table Place

```
1 // Place database model
2
3 namespace SmartParkingLiFi.Data.Model
4 {
5     public class Place
6     {
7         public int Number { get; set; }
8         public char Lane { get; set; }
9         public int Floor { get; set; }
10    }
11 }
12
```

5.5 Annexe 5 : Classe HttpRequests contenant les requêtes envoyées au web service

Il y a en tout 7 requêtes, voici celle pour supprimer une réservation, pour en insérer une et pour afficher celles concernant l'utilisateur.

```
public class HttpRequests
{
    Page RequesterPage;
    HttpClient Client;
    const string ServerAddress = "192.168.2.1";

    public HttpRequests(Page _Page)
    {
        RequesterPage = _Page;
        Client = new HttpClient();
    }

    // Suppression de la réservation
    public async Task<bool> DeleteReserv(string Reference)
    {
        try
        {
            HttpResponseMessage Response = await Client.GetAsync($"http://{ServerAddress}/WebService/DeleteReserv.php?Reference={Reference}");
            if (Response.IsSuccessStatusCode)
            {
                return true; // Success
            }
            else { RequesterPage.DisplayAlert("Erreur", "Impossible d'annuler la réservation. [DR]", "OK"); }
        }
        catch { RequesterPage.DisplayAlert("Erreur", "Aucune réponse du serveur. Veuillez réessayer ultérieurement.", "OK"); }
        return false;
    }

    // Insertion de la réservation
    public async Task<bool> InsertReserv(StringContent Content)
    {
        try
        {
            HttpResponseMessage Response = await Client.PostAsync($"http://{ServerAddress}/WebService/InsertReserv.php", Content);
            if (Response.IsSuccessStatusCode)
            {
                return true; // Success
            }
            else { RequesterPage.DisplayAlert("Erreur", "Impossible de valider la réservation, veuillez réessayer ultérieurement. [IR]", "OK"); }
        }
        catch { RequesterPage.DisplayAlert("Erreur", "Aucune réponse du serveur. Veuillez réessayer ultérieurement.", "OK"); }
        return false;
    }

    // Récupération des réservations client
    public async Task<string> SelectReservs(int ConnectedClientID)
    {
        try
        {
            HttpResponseMessage Response = await Client.GetAsync($"http://{ServerAddress}/WebService>SelectReserv.php?Client_ID={ConnectedClientID}");
            if (Response.IsSuccessStatusCode)
            {
                return await Response.Content.ReadAsStringAsync();
            }
            else { RequesterPage.DisplayAlert("Erreur", "Impossible de charger les réservations, veuillez réessayer ultérieurement.", "OK"); }
        }
        catch { RequesterPage.DisplayAlert("Erreur", "Aucune réponse du serveur. Veuillez réessayer ultérieurement. [GE]", "OK"); }
        return "";
    }
}
```

5.6 Annexe 6 : Programme de l'activité principale de l'application smartphone

```
// Algorithme principale de l'application (threading)
private async void GetReservationAsync()
{
    string JsonData = await Requests.SelectReservs(ConnectedClient.ID); // On récupère les réservations du client

    if (JsonData != "")
    {
        if (JsonData == "No result found") // Pas de réservation
        {
            ClientReservationStatus = ReservationStatus.None; // Mis à jour du status de la réservation client
            ValidateDeleteReservation.Text = "Valider";
            ValidateDeleteReservation.IsEnabled = true; // Il peut valider une nouvelle réservation
            AccessButton.Text = "Accès sans réservation"; // Peut accéder seulement sans réservation
            AccessButton.IsEnabled = true;
            Reference.Text = "-";
            return;
        }

        JsonData = JsonData.Replace("null", "\"1998-05-26 14:12:34\""); // On évite les champs null dans la déserialisation
        JsonData = JsonData.Replace($""\Place_ID":\"{1998-05-26 14:12:34}\", \"$\Place_ID\":0");
        Reservations = JsonConvert.DeserializeObject<List<Reservation>>(jsonData);
        Reservations.Sort((x, y) => x.Fin_Reservation.CompareTo(y.Fin_Reservation)); // On trie les réservations depuis plus récente
        Reservations.Reverse();

        // Reservations[0] est la dernière réservation effectuée
        if (Reservations[0].Depart != new DateTime(1998, 5, 26, 14, 12, 34)) // Si pas de réservation validé
        {
            ClientReservationStatus = ReservationStatus.None; // Mis à jour du status de la réservation client
            ValidateDeleteReservation.Text = "Valider";
            ValidateDeleteReservation.IsEnabled = true; // Il peut valider une nouvelle réservation
            AccessButton.Text = "Accès sans réservation"; // Peut accéder seulement sans réservation
            AccessButton.IsEnabled = true;
            Reference.Text = "";
        }
        else if (Reservations[0].Arrivee != new DateTime(1998, 5, 26, 14, 12, 34)) // Si réservation en cours (dans le parking)
        {
            ClientReservationStatus = ReservationStatus.InProgress; // Mis à jour du status de la réservation client
            ValidateDeleteReservation.Text = "Annuler";
            ValidateDeleteReservation.IsEnabled = false; // Ne peut pas annuler sa réservation
            AccessButton.IsEnabled = false; // Ne peut pas re-acceder au parking
            Reference.Text = Reservations[0].Reference;
            UpdatePickers(Reservations[0].Debut_Reservation, Reservations[0].Fin_Reservation);
            // TODO: update info place dans lifi tab
            if (Reservations[0].Fin_Reservation.AddMinutes(15) < DateTime.Now) // Si dépassement de + de 15 minutes = alerte
            {
                DisplayAlert("Attention", "Vous avez dépassé l'heure de fin de réservation. Vous devez sortir du parking.", "OK");
            }
            Reservations.Remove(Reservations[0]); // On affichera pas la réservation en cours dans l'historique
        }
        else // Si réservation valider (hors parking)
        {
            ClientReservationStatus = ReservationStatus.Validate; // Mis à jour du status de la réservation client
            ValidateDeleteReservation.Text = "Annuler";
            ValidateDeleteReservation.IsEnabled = true; // Peut annuler sa réservation
            AccessButton.Text = "Accès avec réservation"; // Peut accéder mais pas sans réservation
            AccessButton.IsEnabled = true;
            Reference.Text = Reservations[0].Reference;
            UpdatePickers(Reservations[0].Debut_Reservation, Reservations[0].Fin_Reservation);
            if (Reservations[0].Debut_Reservation.AddMinutes(15) < DateTime.Now) // Si dépassement de + de 15 minutes = annule
            {
                if (await Requests.DeleteReserv(Reference.Text))
                {
                    Reference.Text = "-";
                    ClientReservationStatus = ReservationStatus.None;
                    ValidateDeleteReservation.Text = "Valider";
                }
                DisplayAlert("Attention", "Vous avez dépassé l'heure de début de réservation. La réservation est annulé.", "OK");
            }
            Reservations.Remove(Reservations[0]); // On affichera pas la réservation en cours dans l'historique
        }
    }

    // Affichage de l'historique des réservations de l'utilisateur avec tri par mois
    var ReservationList = new List<ReservationCellGroup>();
    var CurrentEventCellGroup = new ReservationCellGroup() { Month = "Aucun événement" };
    foreach (var Reservation in Reservations)
    {
        if (Reservation.Fin_Reservation.ToString("MMMM yyyy") != CurrentEventCellGroup.Month)
        {
            if (CurrentEventCellGroup.Count != 0) { ReservationList.Add(CurrentEventCellGroup); }
            CurrentEventCellGroup = new ReservationCellGroup() { Month = Reservation.Fin_Reservation.ToString("MMMM yyyy") };

            CurrentEventCellGroup.Add(new ReservationCell())
            {
                ID = Reservation.ID,
                Date = "du " + Reservation.Debut_Reservation.ToString("ddd d à HH:mm") + " au " + Reservation.Fin_Reservation.ToString("ddd d à HH:mm"),
                Duration = "Durée: " + (Reservation.Fin_Reservation - Reservation.Debut_Reservation).ToString(@"hh\:mm"),
                Reference = $"Reference {Reservation.Reference}"
            });
        }
        ReservationList.Add(CurrentEventCellGroup);
        ReservationHistoryListView.ItemsSource = ReservationList;
    }
}
```

5.7 Annexe 7 : Méthode permettant la réception de données Li-Fi sur smartphone

```
// Réception de données Li-Fi avec port série micro USB
private async void OpenLiFiReceiverPort()
{
    UsbManager UsbSerialManager = AppActivity.ApplicationContext.GetSystemService(Context.UsbService) as UsbManager;
    var Table = UsbSerialProber.DefaultProbeTable;
    Table.AddProduct(0x1b4f, 0x0008, Java.Lang.Class.FromType(typeof(CdcAcmSerialDriver))); // IOIO OTG
    var Prober = new UsbSerialProber(Table);
    var Drivers = await Prober.FindAllDriversAsync(UsbSerialManager);

    LiFiReceiverPort = null;
    foreach (var Driver in Drivers) // On cherche notre driver (le récepteur Li-Fi)
    {
        foreach (var Port in Driver.Ports)
        {
            if (HexDump.ToHexString((short)Port.Driver.Device.VendorId) == "0403" && HexDump.ToHexString((short)Port.Driver.Device.ProductId) == "6015")
                LiFiReceiverPort = Port;
        }
    }

    if (LiFiReceiverPort == null) { ReceiverStatus.Text = "Récepteur Li-Fi absent"; } // Si il n'est pas branché on affiche un message
    else
    {
        var IsPermissionGranted = await UsbSerialManager.RequestPermissionAsync(LiFiReceiverPort.Driver.Device, AppActivity.ApplicationContext);
        if (IsPermissionGranted) // On demande la permission à l'utilisateur d'utiliser le récepteur (Android)
        {
            SerialIOManager = new SerialInputOutputManager(LiFiReceiverPort) // Configuration du port série
            {
                BaudRate = 115200,
                DataBits = 8,
                StopBits = StopBits.One,
                Parity = Parity.None
            };

            SerialIOManager.DataReceived += (source, args) => // Thread de réception de données
            {
                AppActivity.RunOnUiThread(() => { ReceivedData.Text = Encoding.UTF8.GetString(args.Data); }); // Données reçus
            };

            SerialIOManager.ErrorReceived += (source, args) => // Thread si il y a une erreur
            {
                AppActivity.RunOnUiThread(() => { ReceiverStatus.Text = "Récepteur Li-Fi absent"; SerialIOManager.Close();}); // On affiche un message de débranchement
            };
            try
            {
                SerialIOManager.Open(UsbSerialManager); // On ouvre le port
                ReceiverStatus.Text = "Récepteur Li-Fi opérationnel";
            }
            catch (Java.IO.IOException Exception) { ReceiverStatus.Text = "Erreur récepteur Li-Fi: " + Exception.Message; } // Gestion d'erreur
        }
        else { ReceiverStatus.Text = "Permission requise"; } // Permission refusée
    }
}
```

5.8 Annexe 8 : Programme principale de l'application de la borne Li-Fi

```

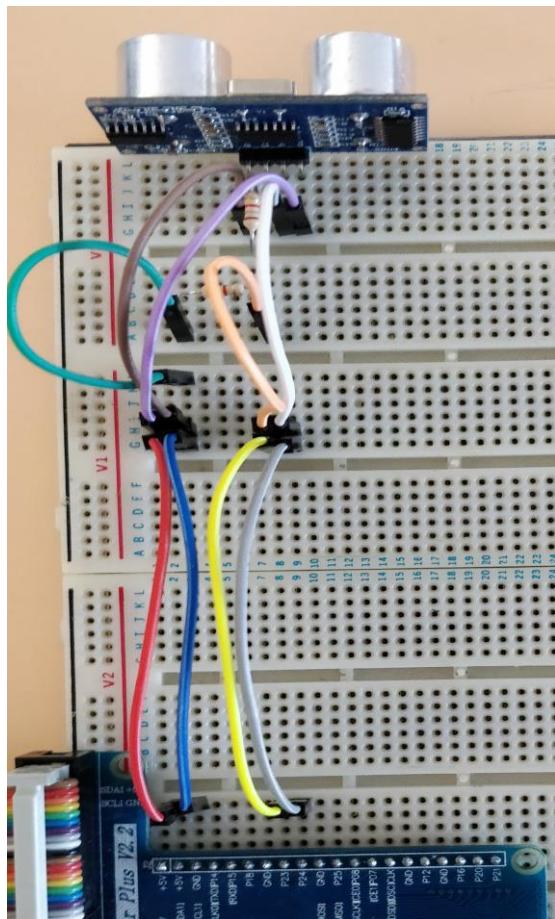
while (true) // Main loop
{
    string Reference = ReadFlashData(); // Attente d'un client
    cout << "Reference: " << Reference << endl;
    if (atoi(Reference.c_str()) >= 1000)
    {
        string Place = ""; // Récupère une place disponible
        string PlaceID = "";
        sql::ResultSet* resl = stmt->executeQuery("SELECT e.Numero, a.Lettre, p.Numero, p.ID FROM place p, allee a, étage e"
        + "WHERE p.Disponible = 1 AND p.allee_id = a.id AND a.etage_id = e.id");
        while (resl->next())
        {
            Place = resl->getString(1) + "|" + resl->getString(2) + "|" + resl->getString(3);
            PlaceID = resl->getString(4);
        }
        delete resl;
        if (Place != "") // Si il y en a une
        {
            if (Reference == "1000") // Invité
            {
                Reference = GenerateUniqueReference();
                string BeginDate = ""; string EndDate = "";
                GenerateReservationDateTimes(BeginDate, EndDate);
                if (!IsPlaceAvailable(BeginDate, EndDate)) // Si parking non complet avec l'arrivée d'adhérents ayant réservés
                {
                    cout << "Erreur: Pas de place disponible en invité \n" << endl;
                }
                else // Insertion réservation invité
                {
                    stmt->execute("INSERT INTO reservation VALUES (NULL, " + Reference + ", "
                    + BeginDate + ", '" + EndDate + "', NULL, NULL, " + PlaceID + ", 1)");
                    SendLiFiData(Reference + "|" + Place); // Envoi référence + place
                    AllumerLEDVerte(); // Allume la LED verte de passage
                }
            }
            else // Adhérent
            {
                sql::ResultSet* res = stmt->executeQuery("SELECT Reference FROM reservation WHERE Reference = "
                + Reference); // Récupère laréservation adhérent
                bool Exist = false;
                while (res->next())
                {
                    Exist = true;
                }
                if (Exist) // Si référence existe bien
                {
                    stmt->execute("UPDATE reservation SET Place_ID = " + PlaceID + " WHERE Reference = "
                    + Reference); // Met à jour la réservation adhérent avec la place attribué
                    SendLiFiData(Reference + "|" + Place); // envoi référence + place
                    AllumerLEDVerte(); // Allume la LED verte de passage
                }
                else { cout << "Erreur: Reference inexisteante \n" << endl; }
            }
        }
        else { cout << "Erreur: Pas de place disponible \n" << endl; }
    }
}

```

5.9 Annexe 9 : Programme C++ pour mesurer une distance

```
main.cpp ✘
1 #include <wiringPi.h>
2 #include <iostream>
3 #include <unistd.h>
4 #include <time.h>
5 #include <math.h>
6
7
8 #define TriggerInput 5
9 #define Echo 4
10
11 using namespace std;
12
13 int main()
14 {
15     clock_t start_t, end_t;
16     double distance;
17
18     //Vérification du bon fonctionnement de la librerie wiringPi
19     if(wiringPiSetup() < 0){
20         cout << "Setup wiringPi failed !" << endl;
21         return 1;
22     }
23     //Initialisation des pin
24     pinMode(TriggerInput, OUTPUT);
25     pinMode(Echo, INPUT);
26
27     while(true){
28         //triggerInput mise de la sortie du trigger a 1 puis à 0 pendant 10µs
29         digitalWrite(TriggerInput, HIGH);
30         delayMicroseconds(2);
31         digitalWrite(TriggerInput, LOW);
32
33         //tant qu'il n'y a pas d'impulsion ne rien faire
34         while(digitalRead(Echo) == false){
35             }
36
37         start_t = clock(); // prise de temps au début de l'impulsion
38         //boucle qui dure le temps de l'impulsion
39         while(digitalRead(Echo) == true){
40             }
41         end_t = clock(); // prise de temps à la fin de l'impulsion
42
43         start_t = end_t - start_t; // calcule de la durée de l'impulsion
44
45         distance = (0.00034*start_t)/2; // calcule de la distance
46
47         cout<< distance << "m\n" << endl;
48         delay(1000); //attente d'une seconde pour avoir une mesure toute les 1s environ
49     }
50
51 }
52 }
```

5.10 Annexe 10 : Câblage du test du capteur



5.11 Annexe 11 : Programme C++ d'initialisation WiringPI

```
//Initialisation de la librairie WiringPI
string CLS_DetecterVehicule::InitialiserWiringPI(){
    //Vérification du bon fonctionnement de la librerie wiringPi
    if(wiringPiSetup() < 0){
        return "Setup wiringPi failed !";
    }else{
        ifstream myFlux(app.c_str());
        if(!myFlux){
            return "impossible d'ouvrir app.txt";
        }
        string distanceDetec;
        int i=0;
        //recupere la 6eme ligne du fichier de config
        while(getline(myFlux, distanceDetec) && i <= 8){
            i++;
        }
        distanceDetec = myUtil->DecouperStringConfig(distanceDetec);
        distanceDeDetection = myUtil->StringToInt(distanceDetec);

        return "Setup wiringPi succes !";
    }
}
```

5.12 Annexe 12 : Programme C++ pour détecter un véhicule

```
//Permet de détecter un véhicule sur une place
bool CLS_DetecterVehicule::Detection(int LEDgreen, int LEDred, int Echo, int TriggerInput){
    //Initialisation des pin
    pinMode(LEDgreen, OUTPUT);
    pinMode(LEDred, OUTPUT);
    pinMode(TriggerInput, OUTPUT);
    pinMode(Echo, INPUT);
    //triggerInput mise de la sortie du trigger à 1 puis à 0 pendant 10µs
    digitalWrite(TriggerInput, HIGH);
    delayMicroseconds(2);
    digitalWrite(TriggerInput, LOW);

    //tant qu'il n'y a pas d'impulsion ne rien faire
    while(digitalRead(Echo) == false){
    }

    start_t = clock(); // prise de temps au début de l'impulsion
    //boucle qui dure le temps de l'impulsion
    while(digitalRead(Echo) == true){
    }
    end_t = clock(); // prise de temps à la fin de l'impulsion

    start_t = end_t - start_t; // calcule de la durée de l'impulsion
    distance = ((0.00034*start_t)/2)*100; // calcule de la distance

    if(distance <= distanceDeDetection){
        //Voiture détecté
        digitalWrite(LEDgreen, LOW);
        digitalWrite(LEDred, HIGH);
        return true;
    }else{
        //Voiture absente
        digitalWrite(LEDgreen, HIGH); ////mettre à LOW pour éteindre toute les leds avant de démonter
        digitalWrite(LEDred, LOW);
        return false;
    }
}
```

5.13 Annexe 13 : Programme C++ pour la connexion à la base de données

```
//Initialise la connexion a la base de données
string CLS_ConexionBDD::InitialiserConnexion(){
    ifstream myFlux(app.c_str());

    if(!myFlux){
        return "impossible d'ouvrir app.txt";
    }
    string serveur, user, password, database;
    getline(myFlux, numEtage);
    numEtage = myUtil->DecouperStringConfig(numEtage);
    getline(myFlux, serveur);
    serveur = myUtil->DecouperStringConfig(serveur);
    getline(myFlux, user);
    user = myUtil->DecouperStringConfig(user);
    getline(myFlux, password);
    password = myUtil->DecouperStringConfig(password);
    getline(myFlux, database);
    database = myUtil->DecouperStringConfig(database);

    myFlux.close();

    mysql_init(&mysql);
    connexion = mysql_real_connect(&mysql, serveur.c_str(), user.c_str(), password.c_str(), database.c_str(), 0, 0, 0);
    if(!connexion){
        return mysql_error(connexion);
    }
    return "Succes d'initialisation de la connexion !\n";
}
```

5.14 Annexe 14 : Programme C++ des requêtes SQL et traitement de données

```
//Recupere la reservation qui n'a pas d'heure d'arrivée et de départ pour la place
bool CLS_ConexionBDD::RecupererReservationSansHeureArrivee(int idPlace){
    string req = "SELECT * FROM reservation WHERE Place_ID=";
    req += myUtil->IntToString(idPlace);
    req += " AND Arrivee IS NULL ORDER BY Arrivee LIMIT 1";

    state = mysql_query(connexion, req.c_str());
    if(state != 0){
        cout << mysql_error(connexion) << endl;
        return false;
    }
    result = mysql_store_result(connexion);

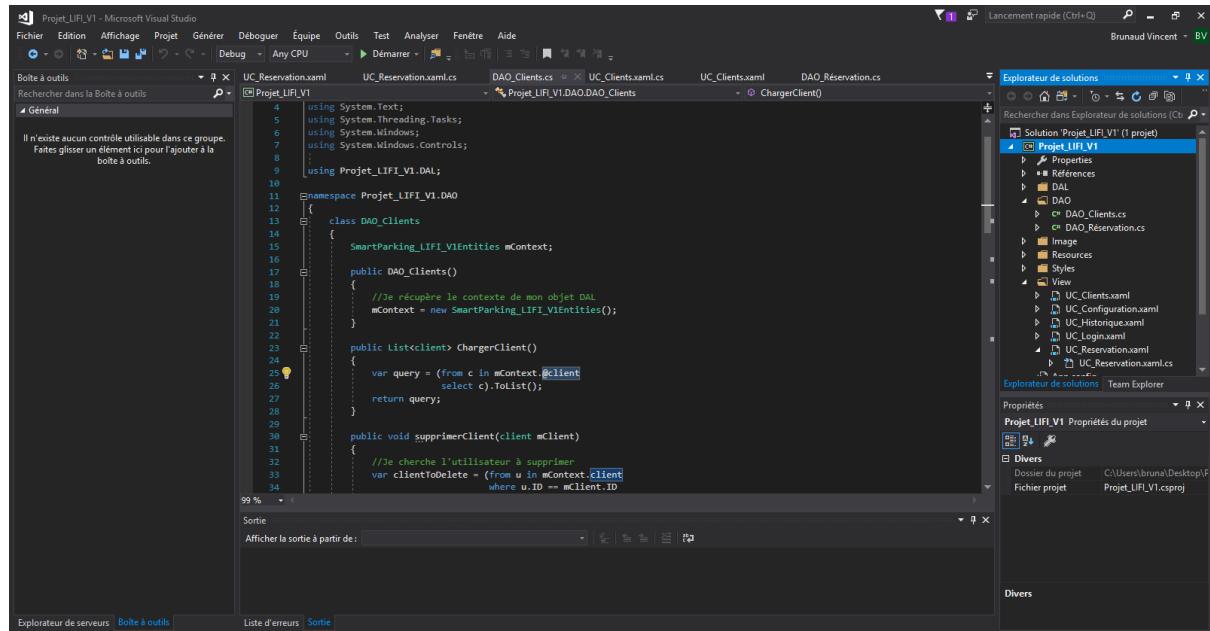
    return true;
}

//Attribu une heure d'arrivée a une place
string CLS_ConexionBDD::AttribuerHeureArrivee(){
    char* arrivee = myUtil->RecupererDateHeure();
    string req = "UPDATE reservation SET Arrivee=''";
    req += arrivee;
    while((row = mysql_fetch_row(result)) != NULL){
        req += "' WHERE ID=";
        req += row[0];
    }

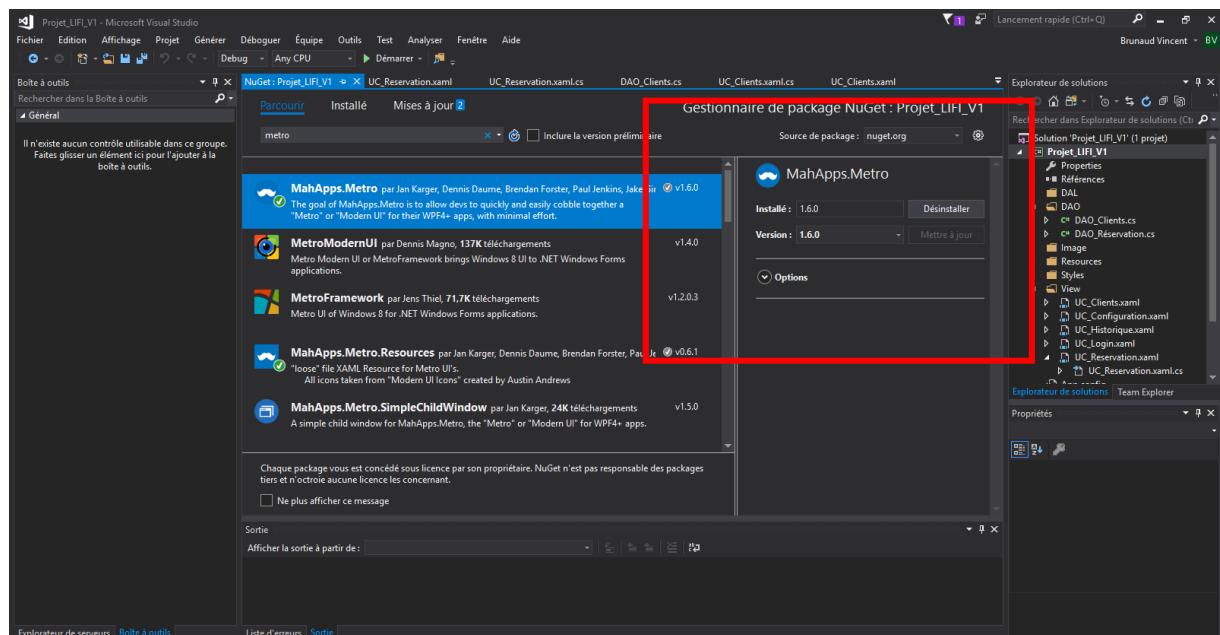
    state = mysql_query(connexion, req.c_str());
    if(state != 0){
        return mysql_error(connexion);
    }
    return "Succes Arrivee";
}
```

5.15 Annexe 15 : Installation de MahApps.Metro et Entity Framework

Dans le logiciel Visual Studio, faites clique droit sur le projet > gérer les packages NuGet.

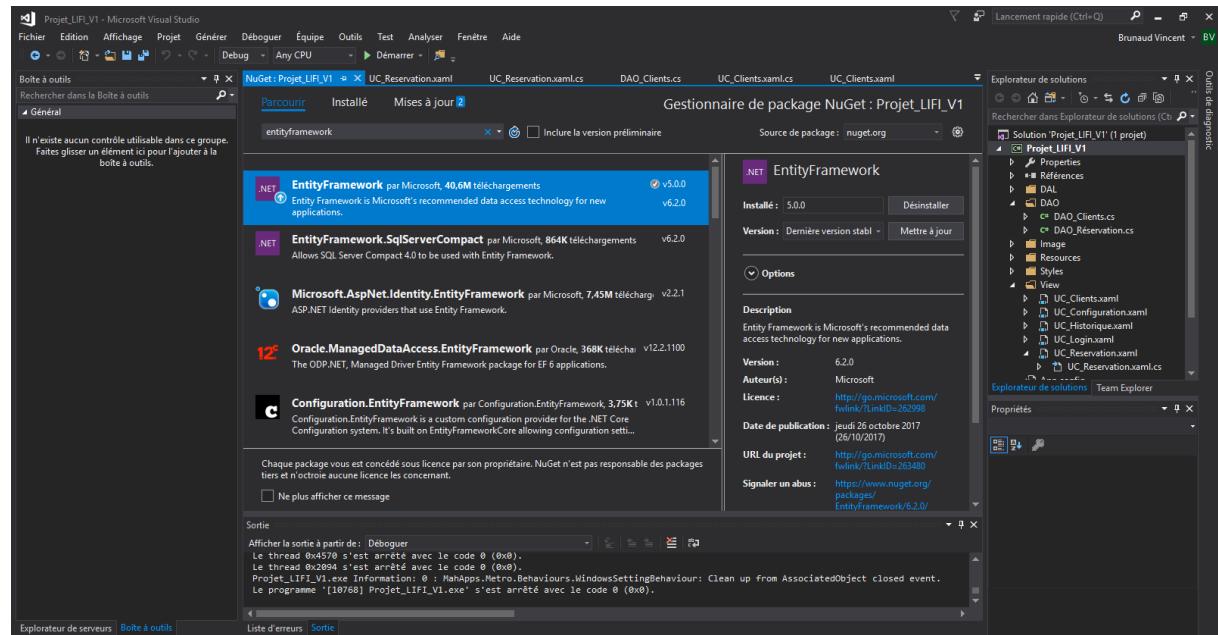


Une fois cette étape réalisée, dans la barre de recherche de parcourir taper **MahApps.Metro** et sur la partie droite choisissez la dernière version disponible (Zone encadrée en rouge).



Pour l'installation d'Entity Framework, suivez le même processus. Cliquez droit sur le projet puis Gérer les packages NuGet et dans la barre de recherche taper Entity Framework.

ATTENTION à bien installé la version 5.0 d'Entity Framework.

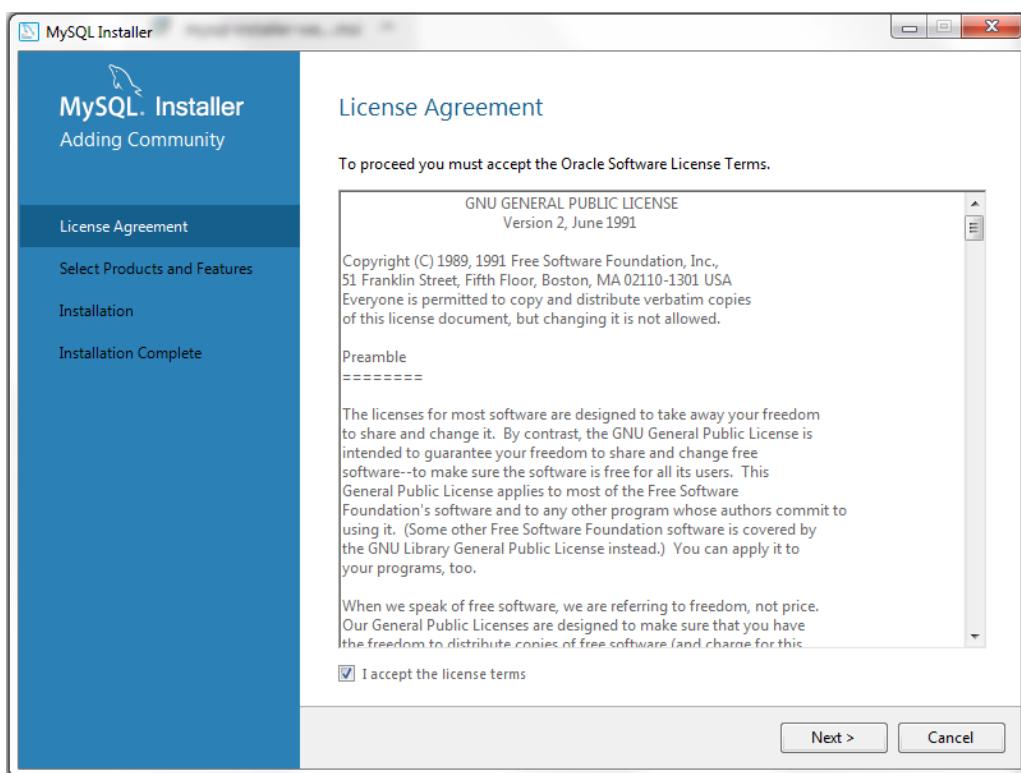
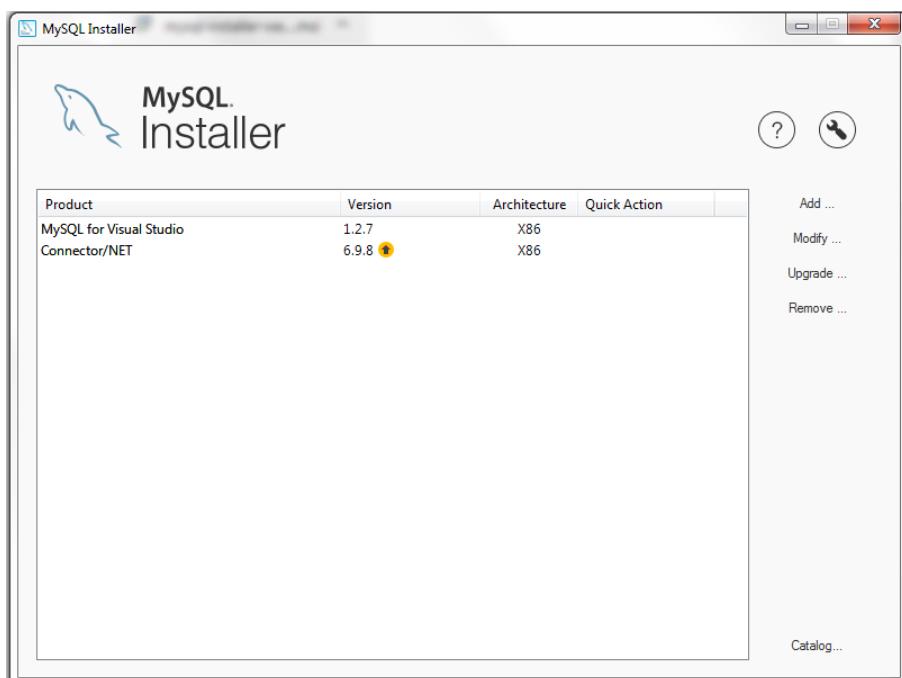


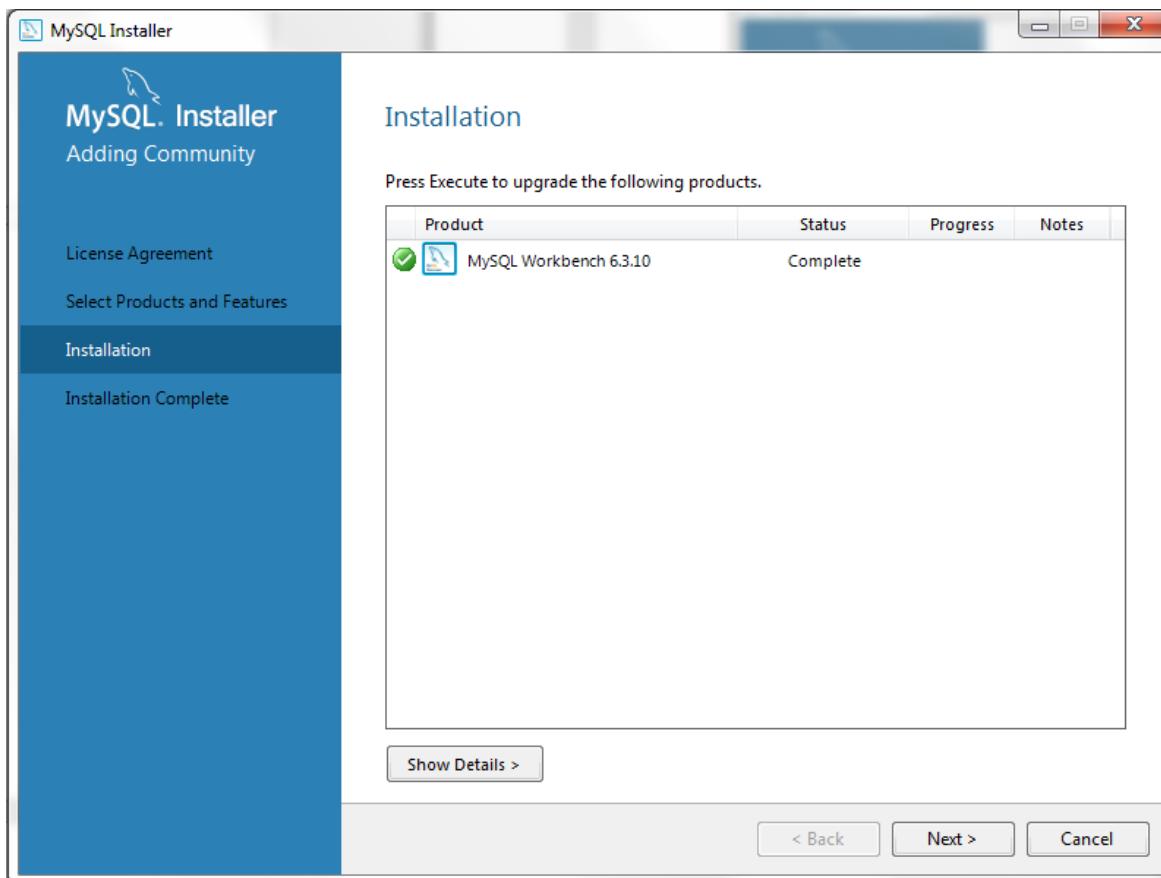
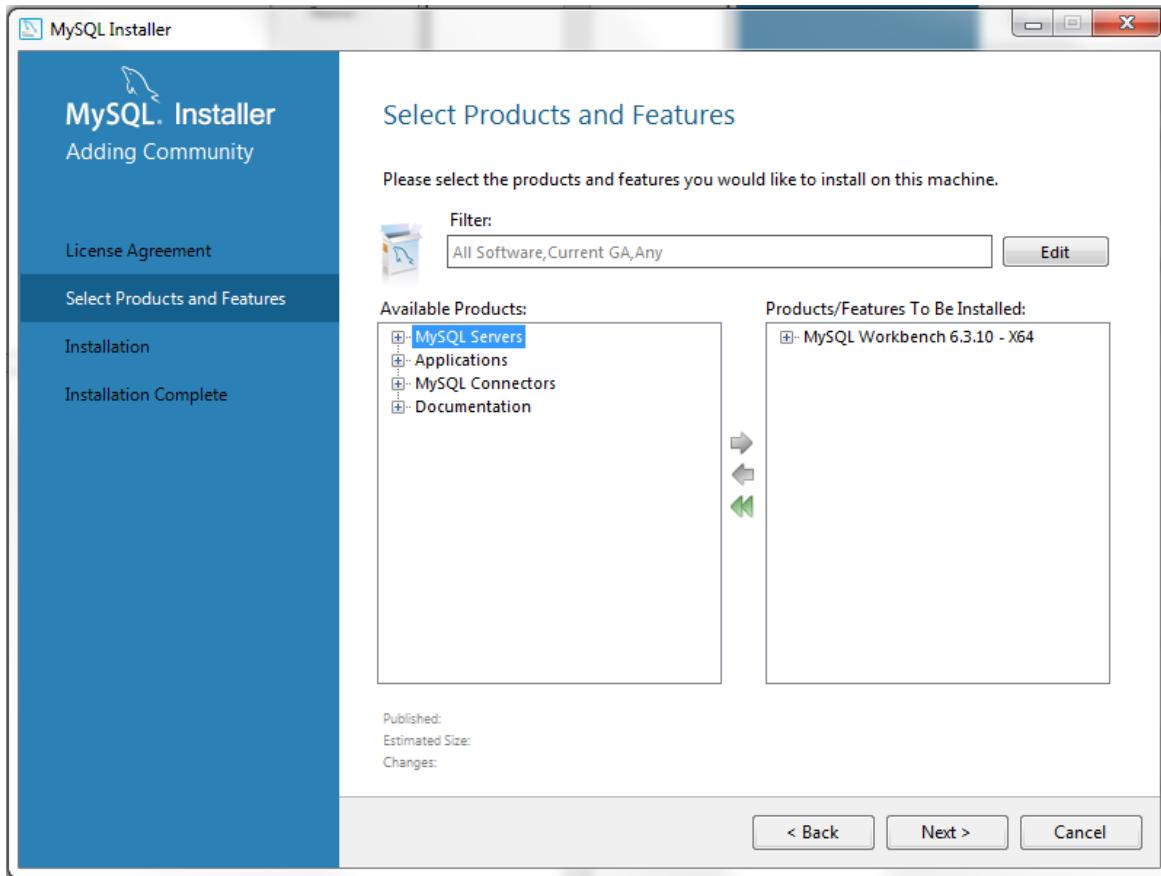
5.16 Annexe 16 : Installation de MySQL Workbench

Tout d'abord on installe MySQL Installer, qui permet d'installer Workbench et tous les utilitaires.



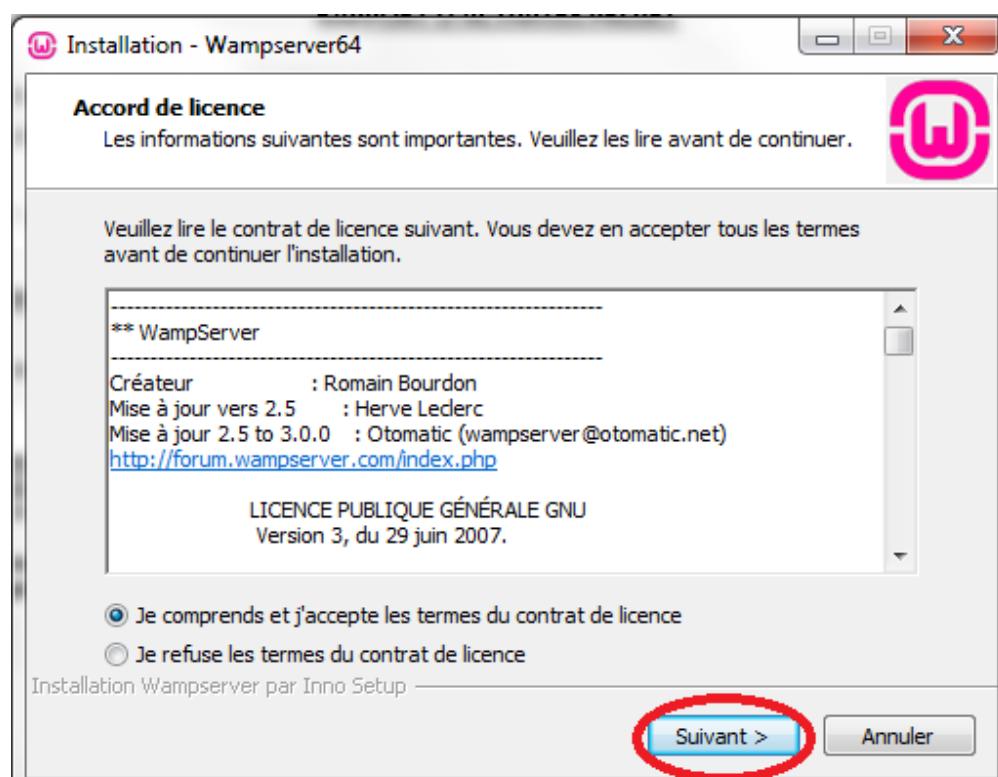
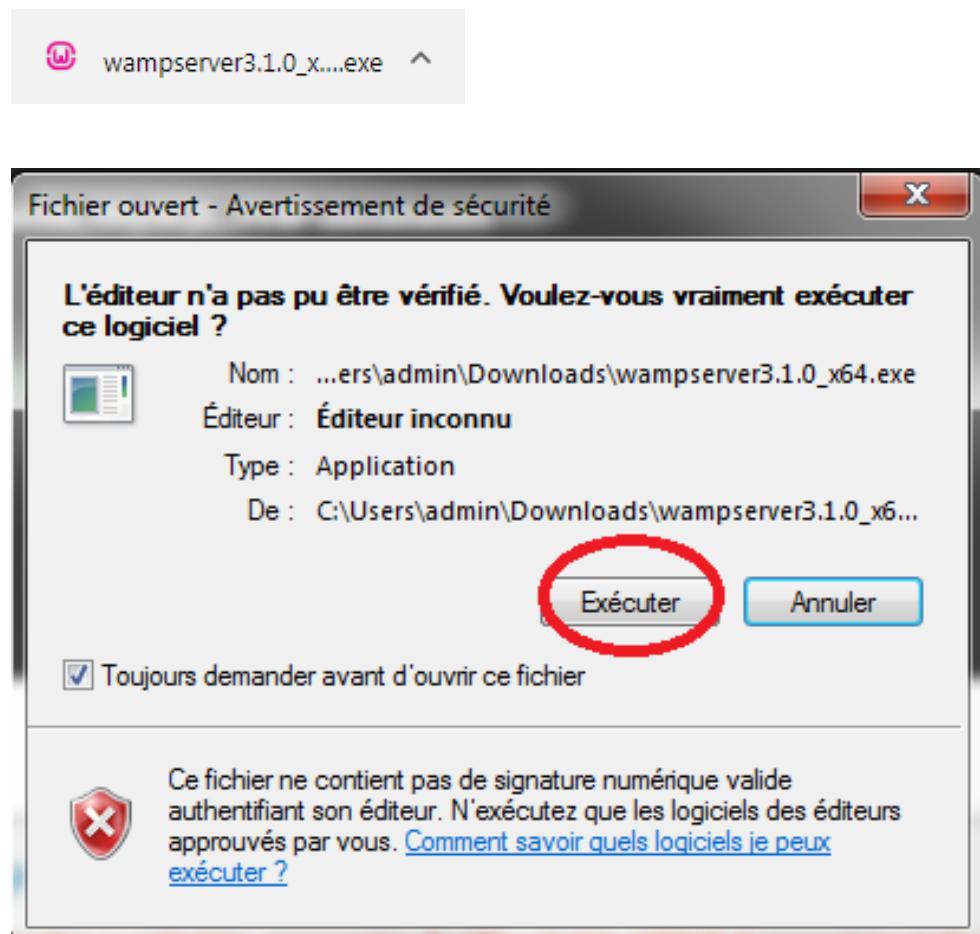
Une fois fait, on peut télécharger et installer les utilitaires dont on a besoin.

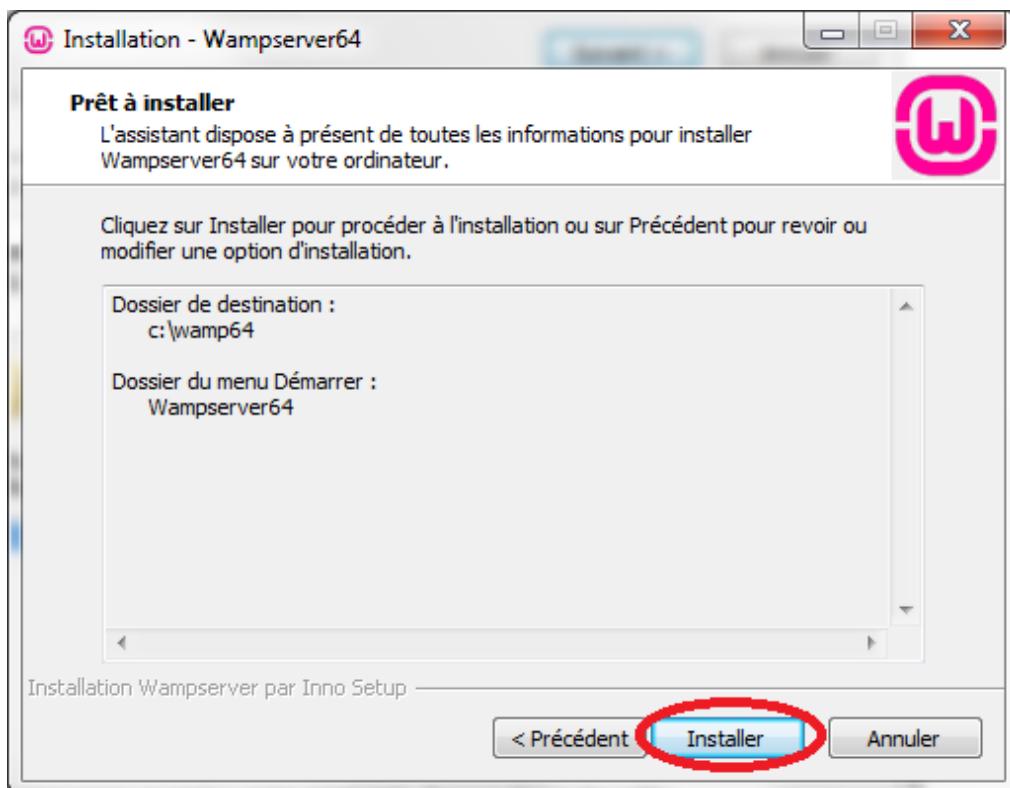
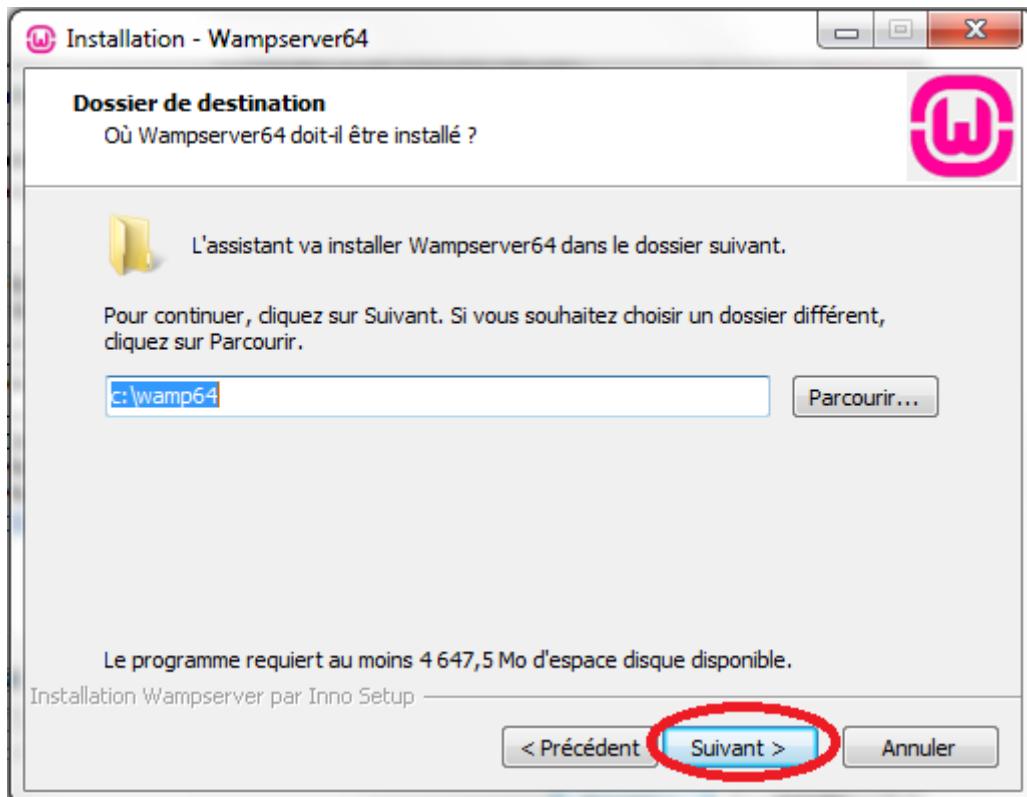




MySQL Workbench est maintenant installé et prêt à l'emploi.

5.17 Annexe 17 : Installation de Wamp Server



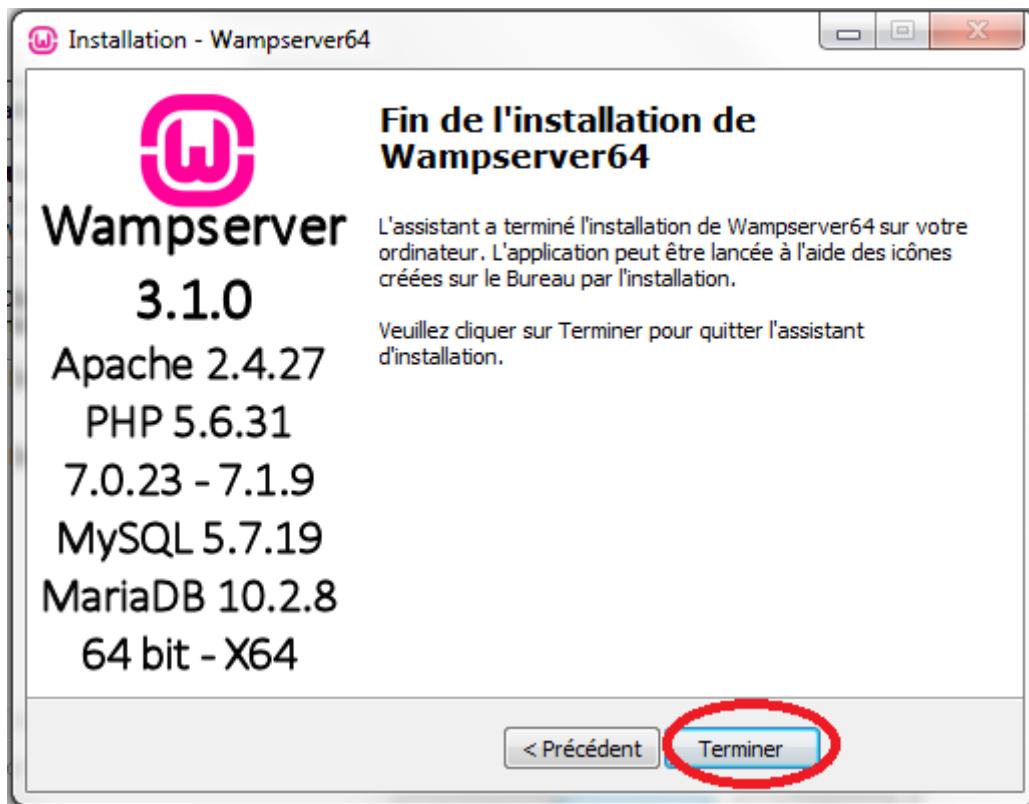


Lorsque proposé, choisir un navigateur :

C:\Program Files (x86)\Google\Chrome\Application\chrome.exe

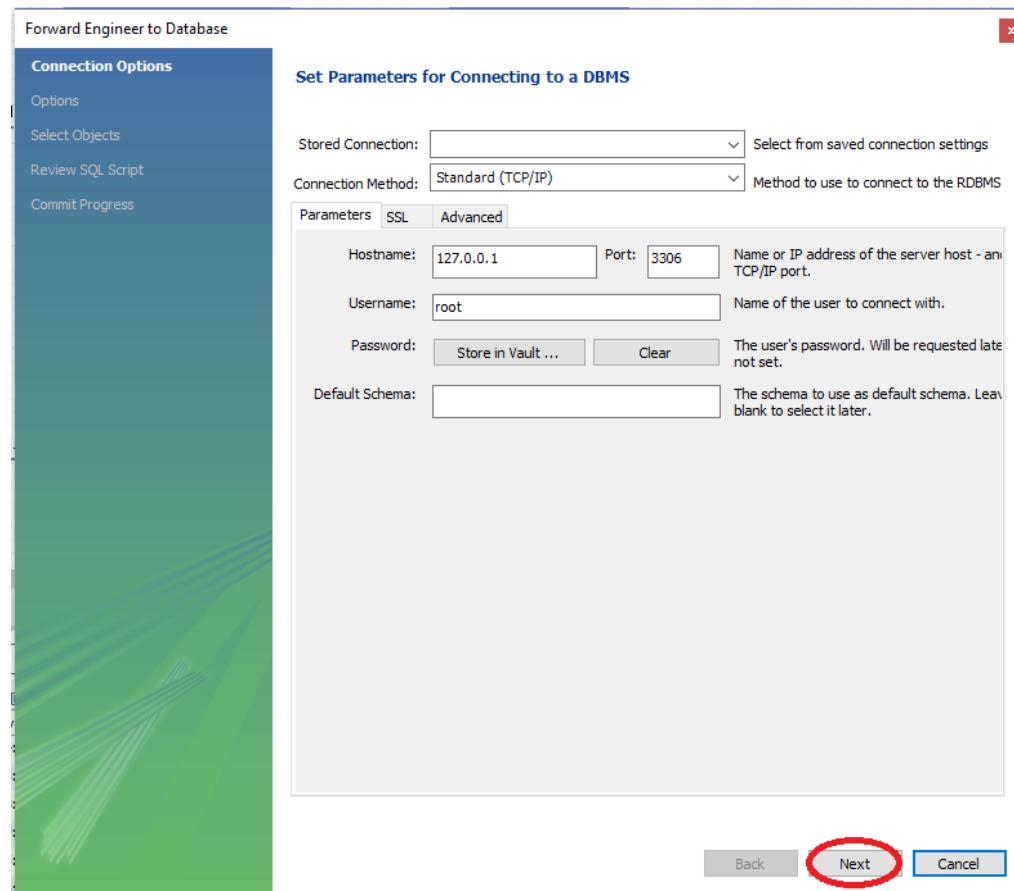
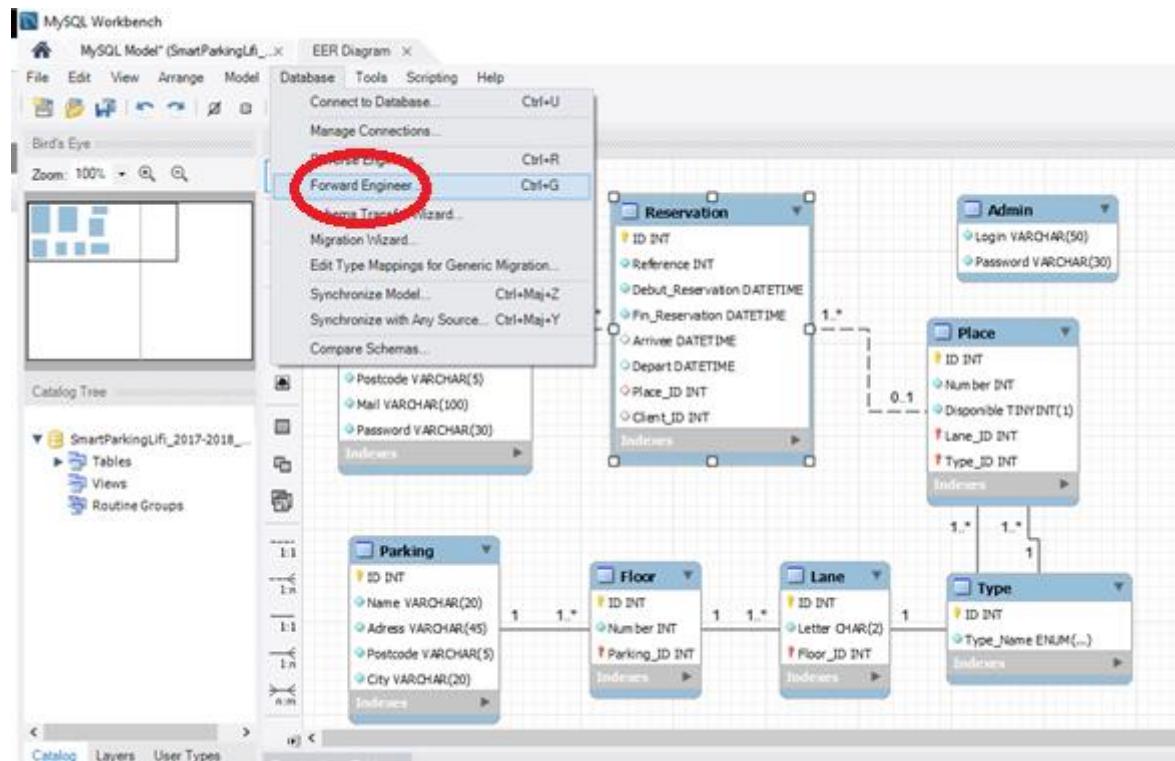
Ensuite, faire de même pour l'éditeur de texte :

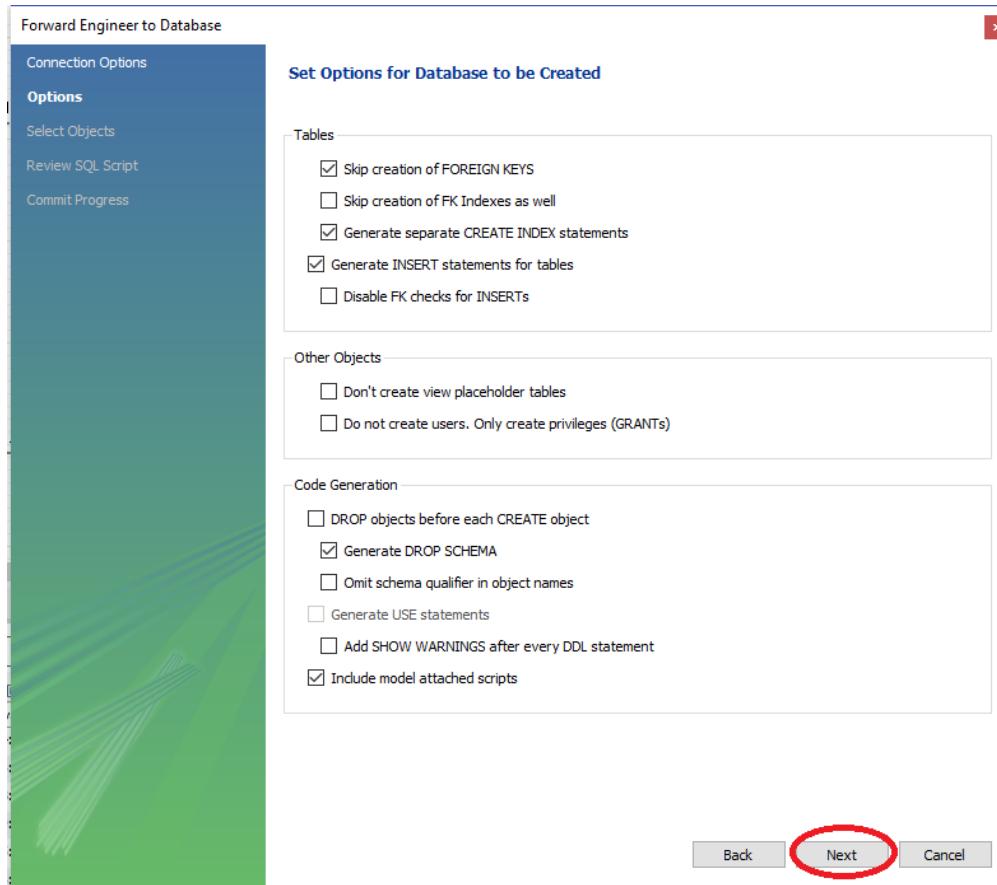
C:\Program Files (x86)\Notepad++\notepad++.exe



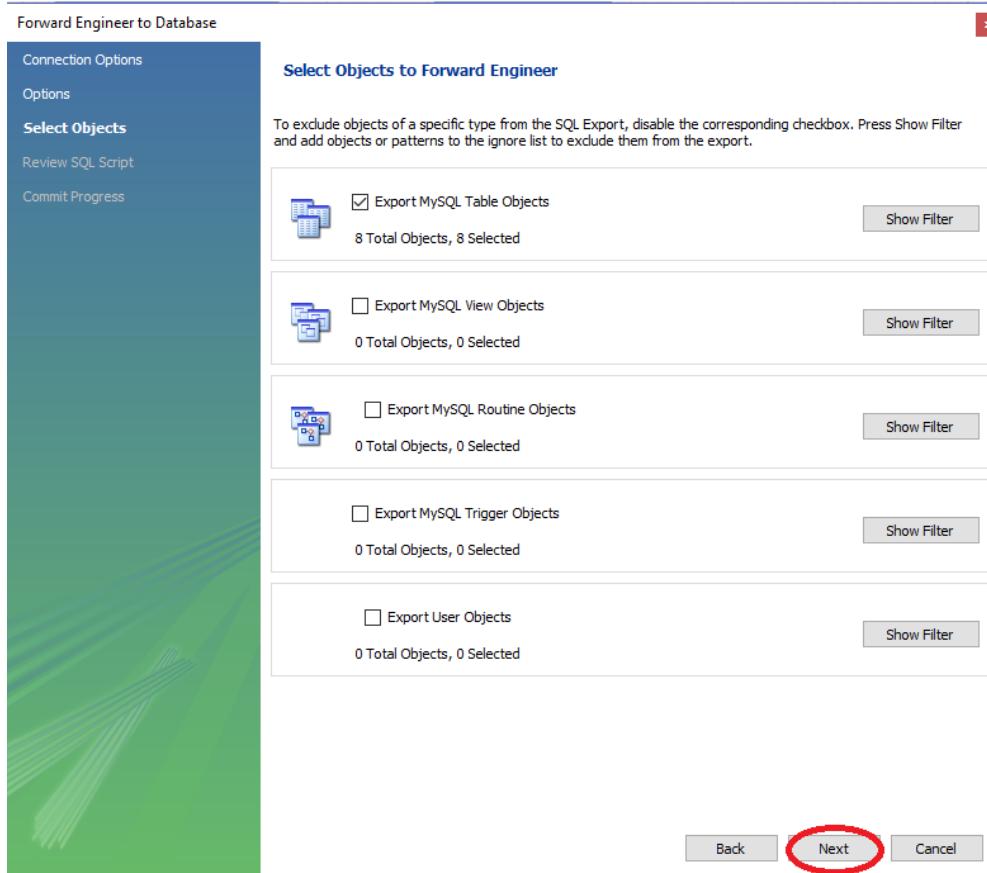
Wamp Server est maintenant installé et prêt à l'emploi.

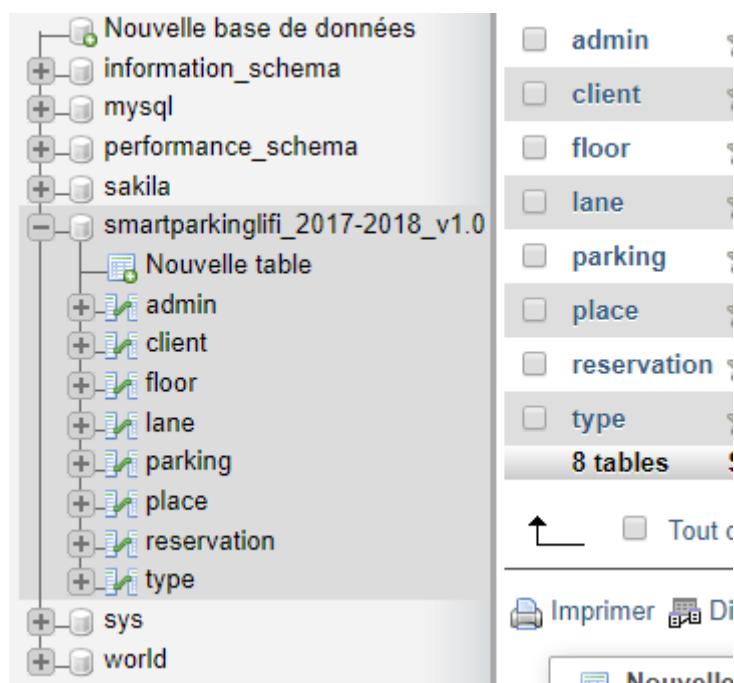
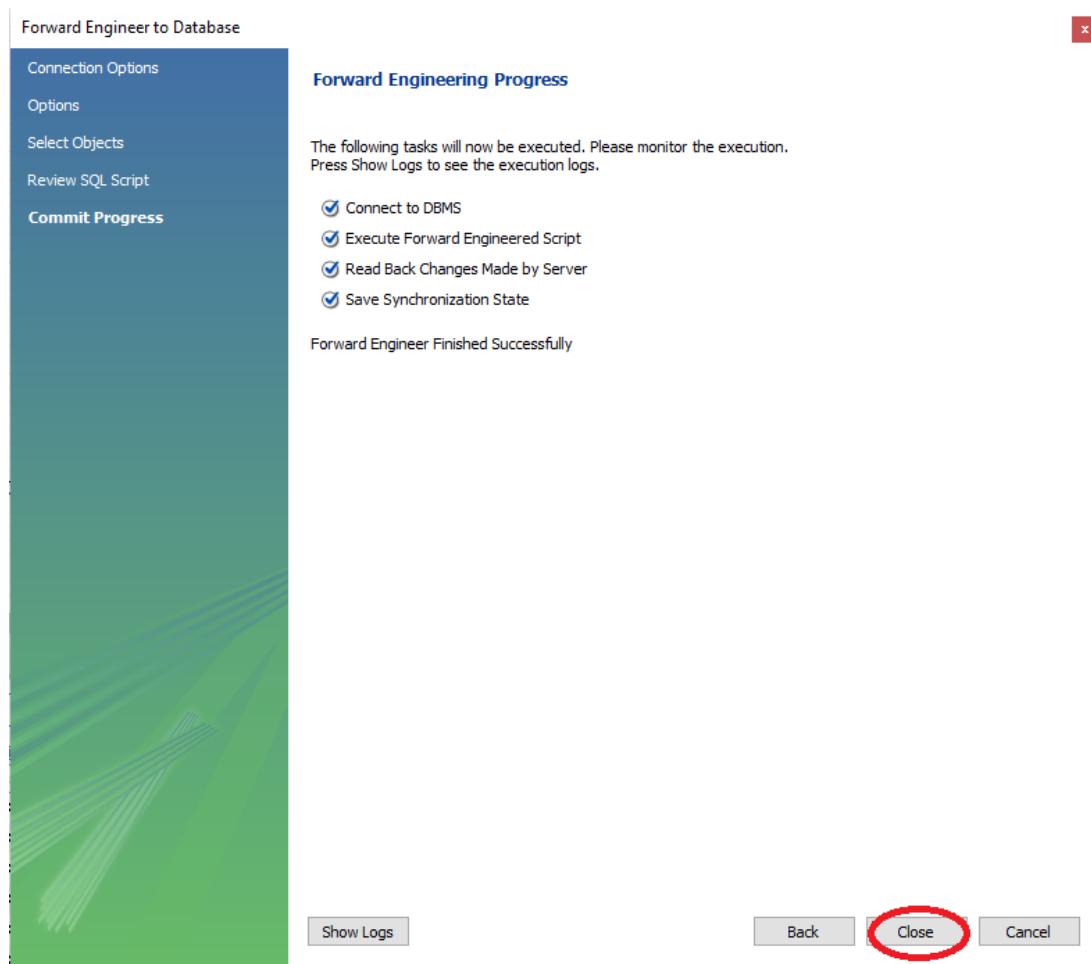
5.18 Annexe 18 : Transférer la base de données sur le serveur MySQL (local)





Pensez à cocher les cases, puis cliquez sur Next.





La base de données est maintenant sur le serveur.

5.19 Annexe 19 : Transférer la base de données sur le serveur en ligne

Tout d'abord, il faut se rendre sur le site de l'hébergeur et se créer un compte.

<https://www.alwaysdata.com/fr/>



The screenshot shows the first step of creating a new account on the Alwaysdata website. It features a light gray header with the word "Inscription" in pink. Below it is a form field labeled "Adresse email*" containing "projetlifi2018@gmail.com". A small explanatory note below the field states: "Cette adresse est utilisée pour vous connecter à votre interface d'administration. Elle ne sera jamais communiquée à des tiers." To the right of the input field is a "VOIR" button. The next section, titled "Compte alwaysdata" in pink, contains a "Nom du compte*" field with "projetlifi" entered. An explanatory note next to it says: "Le nom du compte détermine notamment l'adresse du sous-domaine qui vous sera attribué. Par exemple, si le nom est superman, votre sous-domaine sera http://superman.alwaysdata.net.". Below this is a "Produit*" field with "Pack gratuit (100 Mo)" selected, accompanied by a dropdown arrow icon.

Sur la page d'accueil, à gauche, cliquez sur

+ Ajouter une base de données

Remplir tous les champs puis valider

BASE DE DONNÉES

Informations

Nom*

Le nom doit impérativement commencer par : projetlifi_

Permissions

156739

tous les droits lecture seule aucun droit

VALIDER

Il faut maintenant se connecter à PHPMyAdmin.

Le nom d'utilisateur est le nombre donné lors de la création, ici 156739.



La base de données est maintenant créée. En vous rendant sur PHPMyAdmin vous pouvez lui ajouter du contenu en cliquant sur « Importer », puis sélectionner le fichier SQL

[Choisir un fichier](#) `projetlifi_bdd.sql` puis exécutez.

Après ceci, la base de données est complète et en ligne.

5.20 Annexe 20 : Transférer le web service sur le serveur en ligne

Nouveau site Web

It is your last free website on 000webhost.
You can create unlimited websites when you **Passer à PRO**

Nom du site Web (facultatif)

Mot de Passe

Afficher le mot de passe **GÉNÉRER UN AUTRE MOT DE PASSE**

Créer

En me connectant sur l'hébergeur, je peux voir d'un simple coup d'œil si mon web service est en ligne.

Agréable, voici votre liste de sites Web
Allez dans votre site Web en cliquant dessus ou cliquez sur le bouton pour créer un nouveau site Web.

projetlifi • Statut: En Marche Gérer le Site Détails https://projetlifi.000webhostapp.com/	+	
---	-------------------	--

J'ai ensuite deux manières de mettre des fichiers en ligne, soit directement sur le site de l'hébergeur grâce à l'adresse <https://files.000webhost.com> qui me redirige vers le gestionnaire de fichiers.



Website Name:

projetlifi

Password:

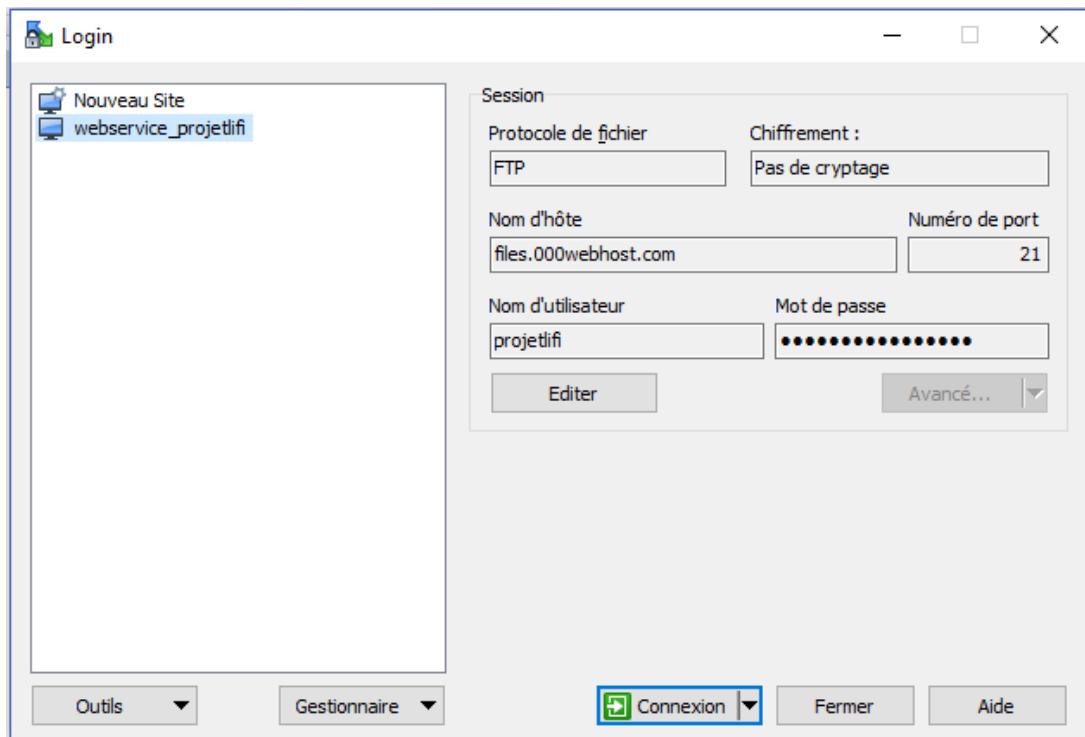
LOG IN

J'ai ensuite accès à l'ensemble des fichiers et peut en ajouter/supprimer.

A screenshot of the 000webhost File Manager interface. The top navigation bar shows the website name "projetlifi" and the current directory "public_html". The left sidebar shows a file tree with root "/", "public_html" (which is expanded to show "index.php" and "css"), and "tmp". The main content area displays a list of files in "public_html":

<input type="checkbox"/>	Nom ▼
<input type="checkbox"/>	.htaccess
<input type="checkbox"/>	Connect.php
<input type="checkbox"/>	CountDispoPlace.php
<input type="checkbox"/>	DeleteReserv.php
<input type="checkbox"/>	InsertReserv.php
<input type="checkbox"/>	Login.php
<input type="checkbox"/>	SelectParking.php
<input type="checkbox"/>	SelectPlace.php
<input type="checkbox"/>	SelectReserv.php
<input type="checkbox"/>	SelectReservInvite.php
<input type="checkbox"/>	UpdateReserv.php

La seconde méthode nécessite le logiciel WinSCP, il faut tout d'abord remplir les champs afin de se connecter au serveur



Ensuite les fichiers peuvent être modifiés d'un simple glisser/déposer (partie de droite)

The screenshot shows the WinSCP interface with two panes. The left pane shows the local directory 'C:\Users\admin\Documents\' containing various files and folders. The right pane shows the remote directory '/public_html/' on the server 'files.000webhost.com'. Both panes have columns for Name, Size, Date modified, Rights, and Properties.

Nom	Taille	Date de modification	Droits	Proprié...
..		25/05/2018 16:26:24		
Agenda		30/03/2018 15:48:29		
Arduino		12/12/2017 08:23:53		
Modèles Office perso...		28/03/2018 09:23:53		
PortFolio		08/11/2017 10:36:03		
TrackMania		02/05/2018 09:55:14		
Virtual Machines		23/05/2018 09:22:21		
Visual Studio 2013		17/10/2017 10:07:41		
Visual Studio 2017		23/05/2018 13:11:12		
bowling_corange_ro...	3 KB	19/01/2018 10:45:40		
Bowling_leo2.mwb	8 KB	19/01/2018 10:49:00		
dgdgd.xlsx	10 KB	25/05/2018 16:26:24		
siteavecconnexionph...	2 KB	09/05/2018 17:14:27		
SN4 Mission.docx	492 KB	02/05/2018 13:26:40		
SQL exam Derain.zip	7 KB	23/01/2018 14:43:16		
Test.rar	379 KB	07/05/2018 11:11:52		
Connect.php	1 KB	29/05/2018 16:20:17	rw-r--r--	5870247
CountDispoPlace.php	1 KB	07/05/2018 14:40:40	rw-r--r--	5870247
DeleteReserv.php	1 KB	07/05/2018 14:40:40	rw-r--r--	5870247
InsertReserv.php	2 KB	29/05/2018 16:23:24	rw-r--r--	5870247
LogIn.php	1 KB	07/05/2018 14:40:40	rw-r--r--	5870247
SelectParking.php	1 KB	07/05/2018 14:40:40	rw-r--r--	5870247
SelectPlace.php	1 KB	07/05/2018 14:40:40	rw-r--r--	5870247
SelectReserv.php	1 KB	07/05/2018 14:40:40	rw-r--r--	5870247
SelectReservInvite.php	1 KB	07/05/2018 14:40:40	rw-r--r--	5870247
UpdateReserv.php	1 KB	07/05/2018 14:40:40	rw-r--r--	5870247

At the bottom, status bars show '0 B de 897 KB dans 0 de 15', '4 cachés 0 B de 7,88 KB dans 0 de 10', '1 cachés', and a timer '0:00:14'.