Bracelet Connecté

AXELIFE

Antoine Lemaitre, Matthias Eddebi, Mickael Karaman
AXELIFE | EI4 SAGI

Table des matières

1. Remerciement

Nous tenons à remercier plusieurs personnes pour la réalisation de ce projet. Nous aimerions remercier l'Istia de nous avoir donné l'opportunité de réaliser ce projet, qui plus est en collaboration avec une entreprise.

Nous remercions notre tuteur Jean-Baptiste Fasquel pour ses retours lors de nos échanges et pour le temps qu'il a passé à nous expliquer certains concepts

Nous remercions aussi Franck Mouney, pour sa réactivité et son aide. Il a toujours répondu à nos demandes et a su bien définir ce qu'il attendait de nous en retour.

Enfin nous souhaitons remercier les autres étudiants de l'Istia qui nous ont aidés sur certaines parties de nos codes, notamment quand nous utilisions les mêmes technologies.

2. Introduction

Nous sommes trois étudiants en 4ème année d'étude en Système Automatisé et Génie Informatique à l'Istia, Université d'Angers. Nous avons réalisé un projet en collaboration avec l'entreprise Axelife, entreprise spécialisée dans le domaine de la santé (lien). Axelife a fait appel à l'Istia pour pouvoir gérer l'envoi de données et le stockage vers une Base de données. L'entreprise à créer un bracelet connecté ayant pour utilité de prévenir des problèmes de santé imminents, et au cas échéant, appeler les secours. Le médecin du détenteur du bracelet pourra avoir accès aux données de ce dernier afin de pouvoir analyser leur évolution, mais aussi être averti en cas de problème. Nos objectifs dans ce projet étaient le suivant :

- Envoi/Réception de données entre une application Android et un serveur en Flask
- Envoi/Récupération de données entre le serveur et la base de données
- Envoi/Récupération de données entre le serveur NodeJS et la base de données
- Envoi/Réception de données entre le site web et le serveur NodeJS

Nous avons donc séparé le projet en trois parties :

- Application Mobile (Matthias)
- Flask (Mickael)
- NodeJS (Antoine

Nous avons donc travaillé 80 heures sur le projet afin que de l'avancer au maximum. Nous voulions aussi que le projet soit facilement récupérable par un membre de l'équipe d'Axelife ou par d'autres étudiants.

3. L'Application Mobile

3.1. Ionic

Ionic.js est un framework JavaScript, initialement basé sur AngularJS et Cordova, puis TypeScript converti en Angular. Le point fort de ce langage est le Cross-Plateforme et depuis Ionic 2 sa fluidité.

Nous sommes partis sur cette technologie pour utiliser l'avantage du Cross plateforme afin d'éviter la répétition dans l'éventuel développement du projet pour iOS.

3.1.1. L'idée

Pour déployer une application Ionic, très rapidement :

Après avoir installé Node.js depuis le site (www.nodejs.org) :

Installer Ionic :

```
C:\Users\matth>npm install -g ionic
C:\Users\matth\AppData\Roaming\npm\ionic -> C:\Users\matth\AppData\Roaming\npm\node_modules\ionic\bin\ionic
npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@1.1.3 (node_modules\ionic\node_modules\fsevents):
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@1.1.3: wanted {"os":"darwin","arch":"any
"} (current: {"os":"win32","arch":"x64"})

+ ionic@3.20.0
added 253 packages in 12.623s
```

Dans le dossier cible :

Cette ligne de commande créer un projet Ionic complet et déjà déployable. Afin de le tester, tout aussi simple :

```
D:\ISTIA\Projet Axelife>cd testApplication

D:\ISTIA\Projet Axelife\testApplication>ionic serve -lab
```

```
D:\ISTIA\Projet Axelife>ionic start testApplication tutorial
```

En annexe 1 on retrouvera le résultat de ces lignes de commande. À première vue, l'application créée par cette technologie semble très simple et rapide à programmer. Rapidement on se rend compte que pour la partie Back-end de l'application, les choses se corsent assez rapidement.

En effet, Ionic gère le Front-End très simplement, mais pour le Back-End, il faut agir avec Angular et dans les fichiers de configuration.

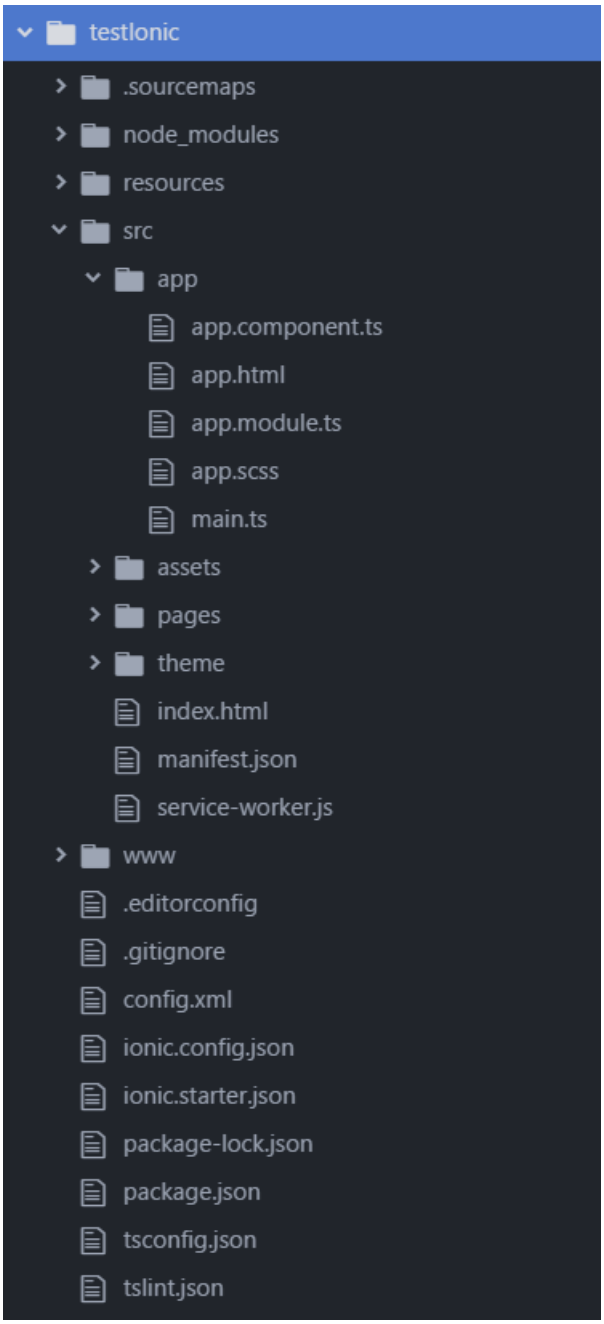


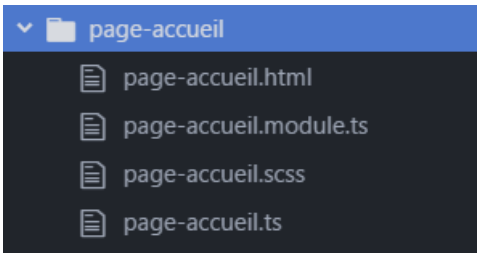
Figure 1 : Architecture d'une application Ionic

Rapidement, nous avons été face à des difficultés sur le Back-End. C'est notamment à cause de ça que Mickaël n'avançait pas. En poussant la réflexion, nous avons trouvé quelques verrous et surtout un gros problème : ce choix de technologie.

Dans l'architecture présente, nous nous intéresserons aux répertoires src/app et src/pages. Dans le répertoire page nous-y retrouverons les pages de l'application. Avec Ionic, une page est composée de 4 fichiers. Pour créer une page en ligne de commande :

```
D:\ISTIA\Projet Axelife\ionicApp\testIonic>ionic
 generate page pageAccueil
[OK] Generated a page named pageAccueil!
```

On obtient donc le sous-répertoire pageAccueil comme ceci :



Dans le .html, on déclarera le Front et les boutons et c'est dans le .ts qu'on définira les méthodes du bouton. Et de plus il faut définir les pages dans app.components.ts et app.module.ts

3.1.2. Les Difficultés

JavaScript est une technologie que nous ne connaissions pas. Beaucoup de temps a été nécessaire pour comprendre le fonctionnement de ce langage. En y ajoutant sa sur-couche Typescript, le temps de lecture de documentation et d'autoformation a été trop long.

Ensuite avec l'environnement, il fallait comprendre comment et où programmer dans l'application. Avec l'émulateur d'application que lançait "ionic serve -lab" on se perdait rapidement dans les erreurs et leurs origines. Le temps de débogage était bien trop long comme nous étions déjà à programmer dans le flou.

Une fois avoir reçu le prototype de l'application du client (sans les librairies de calcul propriétaire soumise à une clause de confidentialité), nous nous sommes rendu compte qu'il était impossible d'intégré le prototype Ionic dans le prototype Android.

En effet, d'un côté l'environnement Ionic est défini d'une certaine façon, en utilisant Node.js et implémentant Angular.js dans l'architecture et d'un autre côté l'application en Android natif à son architecture propre qui n'inclut aucun framework .js

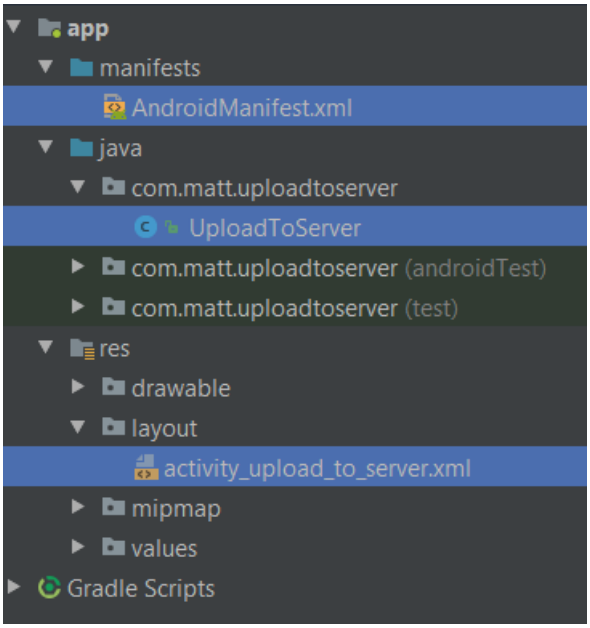
C'est à ce moment-là que nous avons totalement changé pour Android Natif afin d'être plus cohérents avec le prototype du client.

3.2. Android Natif

Google offre avec Android Studio son SDK Android pour du développement spécifique à son OS. C'est avec Graddle, un moteur de production qui s'inspire de Maven que l'application est compilée, les dépendances traitées, etc.

3.2.1. L'idée

À partir d'une application vide, créer une page avec un simple bouton pour envoyer un .csv. Une fois tout l'environnement installé, et configurer, on se focalise sur 3 fichiers :



- AndroidManifest.xml est le fichier de configuration de l'application. Dans celui-ci on retrouve la définition des pages de l'application, les permissions que l'application devra demander à l'utilisateur au téléchargement de l'application et la page principale.
- UploadToServer.java est le fichier contenant tout le back – end.
- Activity_upload_to_server.xml est, comme son chemin l'indique, la ressource qui définit le visuel de la page.

Dans UploadToServer.java, deux méthodes :

- OnCreate, le constructeur de la page dans lequel on définit la vue, les ressources et lance un thread pour la méthode uploadFile.
- uploadFile, la méthode qui gère la connexion et l'envoie du .csv

Cette seconde méthode dans un try, créez une HttpURLConnection, ouvre le fichier, l'envoi et récupère la réponse du serveur.

On prend soin de gérer les exceptions à la sortie du try : avec 2 catch, on intercepte dans le premier une MalformedURLException, dans le deuxième, toute autre exception problématique.

En annexe 2 on retrouvera l'application et l'affichage d'une exception de la seconde partie.

Au lancement de cette application très simpliste, le processus suivant se déclenche :

- UploadToServer est défini comme page principale
- Le nom du fichier, sont adresse locale et l'URI cible (celui du serveur) sont défini

```
final String uploadFilePath = "/sdcard/download/";
final String uploadFileName = "bioWatchAlertTest.csv";
```

- Dans un nouveau Thread on passe ces deux arguments à la méthode uploadFile
- uploadFile créer la connexion http, envoi les fichiers et récupère la réponse du serveur.

À partir de là, nous avons testé l'application depuis l'émulateur. Problème, l'accès à la mémoire de l'émulateur est différent d'un Android classique. On a donc essayé directement sur un téléphone en mode débogage. Cette fois le fichier était bien reconnu, mais le serveur local inaccessible. Dans la sortie console du débogueur, on voit une Exception : "JavaFileNotFound : Permission denied" qui pourrait venir d'une autorisation manquante dans le AndroidManifest.xml

Ce premier prototype d'envoi encore très incomplet mériterait encore quelques heures de réflexion et de développement pour corriger et mettre une place un processus de test correct.

3.2.2. Les Difficultés

Après avoir perdu beaucoup de temps sur Ionic, il manquait encore plus pour fournir un prototype en Android Natif fonctionnel. Encore une fois, beaucoup de documentations et d'autoformation ont occupé le peu de créneaux restant.

La prise en main d'Android aussi a été compliquée bien que Java nous soit familier : le web et le SDK bien spécifique méritent beaucoup d'attention et de rigueur.

3.3. Ce qu'il reste à faire

3.3.1. Connexion

Quand un serveur sera bien en place et accessible dans un réseau ouvert, définir dans le fichier UploadToServer (ligne 47) la bonne adresse d'envoi.

Construire un processus de test de connexion :

- En repartant de l'émulateur, envoyer au localhost
- De l'émulateur, envoyer à un réseau virtuel
- D'un téléphone, à un réseau local
- D'un téléphone à un réseau internet

3.3.2. Implémentation

Ajouter sur la page principale de l'application du client les fonctionnalités de UploadToServer sur un bouton d'envoi. Par la suite il faudra penser à automatiser l'envoi, qu'il soit périodique, ou en attente de réseau s'il n'y en a pas de disponible.

A l'installation, prévoir la création d'un répertoire interne à l'application où celle-ci stockera les fichiers .csv afin que l'appli y cherche directement les bons fichiers à envoyer au serveur sans avoir à redéfinir sans arrêt les variables"uploadFileName" et "uploadFilePath".

3.4. Conclusion Application Mobile

Ce projet a été très intéressant, mais travailler sur une durée si courte, si peu fréquemment a été assez compliqué. Avec l'erreur de technologie que nous avons faite au début nous avons perdu beaucoup de temps.

J'en tire des leçons, prendre plus le temps de la réflexion et mieux saisir le contexte global m'aurait fait rebondir plus tôt certainement.

Ce projet nous a aussi permis de découvrir, outre de nouvelles technologies, un peu plus le web, les serveurs encore peu abordés, et le développement mobile, une première pour nous.

4. Flask

4.1. Introduction Flask

J'ai commencé le projet en m'occupant de la partie application mobile, pour un souci de déploiement sur plusieurs plateformes (IOS, Androïd, Windows Mobile, etc) nous avons choisi d'utiliser Ionic. Mais j'ai rencontré plusieurs problèmes, notamment pour récupérer une variable traitée en couche métier au niveau de la couche UI. Ce problème, à mon sens, était à cause d'une mauvaise compréhension du fonctionnement de l'arborescence d'un projet Ionic ou d'une mauvaise configuration de celle-ci. Sans solutions concrètes, et après discussion avec Matthias qui lui était bloqué sur le serveur Flask. Nous avons alors décidé d'échanger nos tâches, car j'avais mieux compris le principe et le but du serveur.

Dans le contexte général, le serveur Flask se situe entre l'application Android et la base de données MySQL. Le serveur se charge de recevoir un fichier CSV (Comma-separated Values), il le traite de façon à extraire les données voulues et il implémente la base de données avec celles-ci.

L'avantage de Flask est que ce framework web soit en python ce qui le rend compatible avec le programme que M. Mouney a développé pour traiter les données du bracelet connecté. Ce programme se trouve actuellement sur l'application. Cela la rend plutôt lourde et lente, dans le but de l'optimiser M. Mouney souhaite pouvoir faire ce traitement sur le serveur.

De plus, il présente un grand nombre d'extensions de qualité et bien intégrées permettant de réaliser beaucoup d'application différente, ce qui représente un avantage dans la connexion avec la base de données et le traitement du CSV.

Pour implémenter ce serveur, j'ai choisi d'appliquer la méthodologie suivante :

Dans un premier temps je me suis autoformé sur Flask, ce que c'était, comment installer l'environnement, quelles extensions me seront utiles. Pour cela j'ai essentiellement utilisé la documentation des sites "dev.mysql" et "OpenClassRoom".

Ensuite j'ai choisi d'implémenter un fichier pour chaque action primordiale que le serveur doit réaliser soit :

- Un fichier pour gérer la connexion avec une base de données et tester une requête simple (Select).
- Un fichier pour gérer la création de tables à l'intérieur de la base de données.
- Un fichier pour traiter un fichier CSV et l'implémentation de la table appropriée.
- Un fichier pour gérer la communication avec l'application Androïd.

4.2. Développement du serveur

Pour commencer le développement, j'ai d'abord installé Flask sur mon ordinateur. Pour cela c'est assez simple, dans l'invite de commande Windows il suffit de taper : "pip install".

Si la commande "pip" n'est pas installée sur votre ordinateur, des fichiers python existent sur le web qui l'installe pour vous. Une fois ces commandes effectuées, il suffit d'avoir un environnement de développement vous permettant de coder et débugger en python.

J'ai choisi d'utiliser PyCharm, car nous l'avons déjà utilisé dans différents cours.

4.2.1. Établir une connexion avec une BDD & exécuter une requête simple

D'abord je crée un fichier python Connexion.py, ensuite j'importe les différentes librairies nécessaires :

```
1    from flask import Flask
2    from flaskext.mysql import MySQL
3    from mysql.connector import (connection_cext)
```

La première ligne sert à importer les librairies Flask donc à définir que l'application est une application utilisant le framework Flask.

"from flaskext.mysl import" permet d'ajouter l'extension MySQL et les fonctions qu'elle contient.

"from mysql.connnector import (connection_cext)" cet import permet d'utiliser la fonction connection.cext qui définit le contexte de la connexion avec la base de données entre autres les paramètres de connexion telle que l'utilisateur, le mot de passe, etc.

```
mysql = MySQL()          # lance le service MySQL
app = Flask(__name__)    # définit l'application comme une application Flask
```

Une fois les différents imports réalisés je lance le service MySQL, définit l'application comme une application Flask.

```
# variable de connexion
db_name = 'axelife'
db_user_pas = 'root'
db_host = 'axelife'

# MySQL connexion config

cnx = connection_cext.CMySQLConnection(user=db_user_pas, password=db_user_pas, host=db_host, database=db_host)
cursor = cnx.cursor() # On créé un nouveau curseur, un objet MySQL, en utilisant la méthode de connexion "cursor()".
```

Je définis les variables qui sont mises en paramètre du contexte de connexion. Une fois ces variables définies j'ouvre une connexion et stock l'objet de connexion dans la variable "cnx". Ensuite je créé le curseur qui va nous permettre d'exécuter la requête SQL que je stocke elle aussi dans une variable.

```
# Définition de la requête SQL
sh_employee = ("SELECT nom, prenom, sexe FROM users where ID=0")
# Exécute la requête SQL à l'aide de la méthode exécute ().
cursor.execute(sh_employee)
```

Pour finir, je parcours le curseur, qui fonctionne en réalité comme un tableau avec autant de colonnes que la requête a d'arguments.

```
for (nom, prenom, sexe) in cursor: # affiche le résultat de la requête
  print("{} {} {}".format(nom, prénom, sexe))

cursor.close() # Détuit le curseur
cnx.close() # Ferme la connexion
```

Dans ce "for" je stocke le contenu de la première colonne dans la variable "nom" , celui de la deuxième dans la variable « prénom » et enfin le contenu de la dernière dans "ville" que j'affiche ensuite à l'aide de la méthode print() et cela pour chaque ligne que contient la table.

Les deux dernières lignes servent à fermer le contexte de connexion.

4.2.2. Créer une table dans la base de données

Je crée un fichier Create_tab.py. Le début de ce fichier ne change pas j'importe les librairies nécessaires vues précédemment et je rajoute celle des codes d'erreurs de MySQL.

```
from mysql.connector import errorcode
```

Je définis ensuite un dictionnaire ; un dictionnaire est un objet conteneur au même titre que les listes à la différence qu'il n'est pas nécessaire de savoir la position de l'objet contenu comme avec une liste pour le trouver. En effet le dictionnaire utilise un système de clés qui peuvent être de types différents.

```
# on cré un dictionnaire (TABLES) de requête pour créer les tables nécessaires
TABLES = {}
```

Une fois celui-ci définit je l'instancie avec une clé "user_data" pour cet exemple et elle sera liée à la requête SQL CREATE TABLE appropriée (Cf. annexe n°3)

```
for name, ddl in TABLES.items():
  try:
    print("Creating table {}: ".format(name), end='')
    cursor.execute(ddl)
  except mysql.connect().Error as err:
    if err.errno == errorcode.ER_TABLE_EXISTS_ERROR:
      print("already exists.")
    else:
      print(err.msg)
  else:
    print("OK")

cursor.close()
cnx.close()
```

À l'aide de la boucle "for" on parcourt le dictionnaire TABLES. La variable "name" stocke les clés et la variable "ddl" la requête associée à cette clé.

Le "try" sert à gérer les codes d'erreurs qui peuvent être émis par MySQL, plus spécifiquement si la table que l'on essaie de créer le compilateur retourne cette erreur : "_mysql_connector.MySQLInterfaceError: Table 'user_data' already exists"

4.2.3. Ouvrir/traiter un fichier CSV à fin d'implémenter une table de base de données

Je créé un fichier traitementCSV.py Le début de ce fichier ne change pas j'importe les librairies nécessaires vues précédemment et je rajoute celle du CSV.

```
import csv
```

Je définis d'abord des variables ou je stocke les données qui nous serviront à définir les paramètres de la requête SQL.

```
#                                                                    Variables
Id                                                 =                       0
Nom                                                =                       " "
Prenom                                             =                       " "
Age                                                =                       0
Poids                                              =                       0
Taille                                             =                       0
sexe = " "
```

Ensuite j'ouvre le fichier qui "PatientMedecinParam.csv", l'argument "r" définit que l'on ouvre le fichier en lecture seule.

```
#Ouverture et lecture du fichier CSV
with open('PatientMedecinParam.csv', 'r') as csvfile:
  reader = csv.reader(csvfile, delimiter=';')
```

Je lis le fichier csv à l'aide de la méthode reader(). Le premier argument est le fichier csv le second définit comme je découpe les informations contenues dans le fichier.

```
['Données utilisateur:;']
['Nom;Joseph']
['Prénom;Marcus']
['Age;68']
['Poids;87']
['Taille;175']
['Sexe;Homme']
['N°id;157645']
[';']
```

Une virgule séparera les données par ligne.
Ci-contre la sortie console avec le délimiteur définit comme une virgule.

Comme on peut le voir, les données sont séparées entre elles par des points-virgules.

```
['Données utilisateur:', '']
['Nom', 'Joseph']
['Prénom', 'Marcus']
['Age', '68']
['Poids', '87']
['Taille', '175']
['Sexe', 'Homme']
['N°id', '157645']
```

- Un Point-virgule séparera les données par "case".

Chaque case d'un fichier csv est séparée de la suivante de cette manière lorsque l'on l'ouvre dans un fichier texte par exemple.

Ci-dessus la sortie console avec le délimiteur défini comme étant un point-virgule.

"reader" est un objet conteneur comme l'objet curseur vu précédemment. En fonction du délimiteur il peut être sur une ou plusieurs colonnes, 1 colonne avec la virgule et 2 dans ce cas avec le point-virgule.

Une fois le reader créé, je me rends compte que les informations aillant la même définition, par exemple le nom du patient celui du contact d'urgence et celui du médecin, sont difficilement dissociables. Je choisis donc de les différencier dans leurs appellations (nom_u pour le contact d'urgence et nom_m pour le médecin par exemple).

Il faut maintenant que je stocke les données dans les variables définies au début du programme.

```
for row in reader:
    if (row[0] == "N°id"):
        Id = row[1]
    elif (row[0] == "Nom"):
        Nom = row[1]
    elif (row[0] == "Prénom"):
        Prenom = row[1]
    elif (row[0] == "Age"):
        Age = row[1]
    elif (row[0] == "Poids"):
        Poids = row[1]
    elif (row[0] == "Taille"):
        Taille = row[1]
    elif (row[0] == "Sexe"):
        sexe = row[1]
```

Pour cela j'utilise une boucle for ou je parcours reader, qui souvenait vous est sur 2 colonnes.

Ainsi row[0] correspond à la première colonne du fichier csv et row[1] la deuxième. À l'aide de if imbriqué, je trouve les infos que je souhaite et j'implémente les variables à l'aide du row[1] associé.

Pour finir, je dois implémenter la table users de la base de données. Je créé donc une connexion à la base de données et définit la requête suivante :

```
add_user = ("INSERT INTO `users` "
            "(`ID`, `NOM`, `PRENOM`, `AGE`, `POIDS`, `TAILLE`, `SEXE`)"
            "VALUES (%(Id)s, %(Nom)s, %(Prenom)s, %(Age)s, %(Poids)s, %(Taille)s, %(sexe)s)")
```

Pour pouvoir utiliser les variables définies et implémentées au-dessus on utilise le format "%(variable)s" qui est du python étendu. On définit un dictionnaire de data.

```
data_user = {
    'Id': Id,
    'Nom': Nom,
    'Prenom': Prenom,
    'Age': Age,
    'Poids': Poids,
    'Taille': Taille,
    'sexe': sexe,
}
```

Où on associe les variables de data aux ID définis dans la requête.

Finalement on passe la variable de la requête et celle du dictionnaire en paramètre de la méthode execute().

```
cursor.execute(add_user, data_user)
cnx.commit()
```

J'applique le change en appelant la méthode commit() puis on détruit le curseur et on ferme la connexion.

4.3. Difficultés & Solutions

La première difficulté que j'ai rencontrée est le fait que j'ai perdu beaucoup de temps en autoformation. Principalement avec la technologie Ionic, cette perte de temps est due à un mauvais choix technologique et à une mauvaise compréhension de celle-ci. La mauvaise compréhension est peut-être due au fait que je n'utilisais pas la même version de Ionic que les documentations que j'ai consultées.

La seconde difficulté repose sur la syntaxe des requêtes SQL. Effectivement la syntaxe doit être compatible à la fois avec l'interpréteur SQL et le python. La plus grosse difficulté ici a été la gestion d'une clef étrangère. En général, j'utilisais d'abord le compilateur SQL présent sur PhpMyAdmin pour instancier mes requêtes SQL puis je les incorporais au code python.

Le troisième souci que j'ai eu était sur le fichier Create-tab.py. À la fin du code, la gestion d'erreur qui permet de faire un retour console spécifique si la table existe déjà génère une erreur bloquante. Donc si une table du dictionnaire est déjà présente lors de l'exécution de la requête les autres tables qui n'existent pas ne sont pas créées.

Pour corriger cela j'ai enlevé la gestion de l'erreur cette erreur et j'ai préféré ajouter la condition "IF NOT EXISTS" dans les requêtes SQL du dictionnaire.

Le code devient donc :

```
for name, ddl in TABLES.items():
    try:
        print("Creating table {}: ".format(name), end='')
        cursor.execute(ddl)
    except mysql.connect().Error as err:
        print(err.msg)
    else:
        print("OK")
```

Ce qui permet tout de même d'avoir message d'erreur ou la confirmation de la création de la table.

Pour finir, le dernier problème que j'ai rencontré est sur le traitement du fichier CSV des données brutes de l'application. Dans ce fichier les données sont organisées en colonne et non en ligne comme dans le fichier de paramètres utilisateur.

Pour contrer ce problème, j'ai d'abord pensé modifier le code du fichier traitementCSV.py afin de parcourir le dictionnaire "reader" colonne par colonne. L'autre solution est que le fichier soit remanié de façon à avoir les données en ligne dès la réception, le problème serait facilement résolu.

Par manque de temps nous n'avons pas pu développer ces solutions.

4.4. Conclusion Flask

Ce projet, bien que conséquent, a été pour moi un très bon moyen d'être plus à l'aise avec le python et le SQL. Il m'a aussi permis d'apprendre à développer un serveur Flask, technologie que je ne connaissais pas.

Ça m'a aussi permis de me rendre compte que le choix d'une mauvaise technologie pouvait être un réel problème au niveau du temps. Il ne faut pas hésiter à changer de technologie pour éviter une perte de temps trop conséquente et donc que cela ait une répercussion sur l'ensemble du projet. Mon seul regret est que le projet ne soit pas entièrement fonctionnel, mais j'espère que notre travail permettra une réalisation complète pas la suite.

Finalement, ce projet reste encore à être amélioré. La communication avec l'application n'a pas pu être développée par manque de temps. Évidemment, le traitement du fichier csv de données brutes et l'implémentation de la table associée sont à réaliser aussi.

Il serait judicieux de développer un programme gérant la connexion avec un identifiant est un mot de passe. J'ai commencé à développer un code pour cela, mais mes notions actuelles en sécurité informatique m'ont freiné dans cette démarche.

5. NodeJS

5.1. Introduction NodeJS

L'un des objectifs du projet est qu'un médecin puisse lire et accéder à des données de ses clients. Le plus simple est une consultation sur un site web présentant trois avantages :

- Le premier est que le médecin n'a pas besoin d'installer de logiciel sur son ordinateur.
- Le deuxième est que le médecin puisse accéder à ses données depuis n'importe quelle device (en cas d'urgence)
- La troisième est la possibilité de créer une application à partir d'un site web

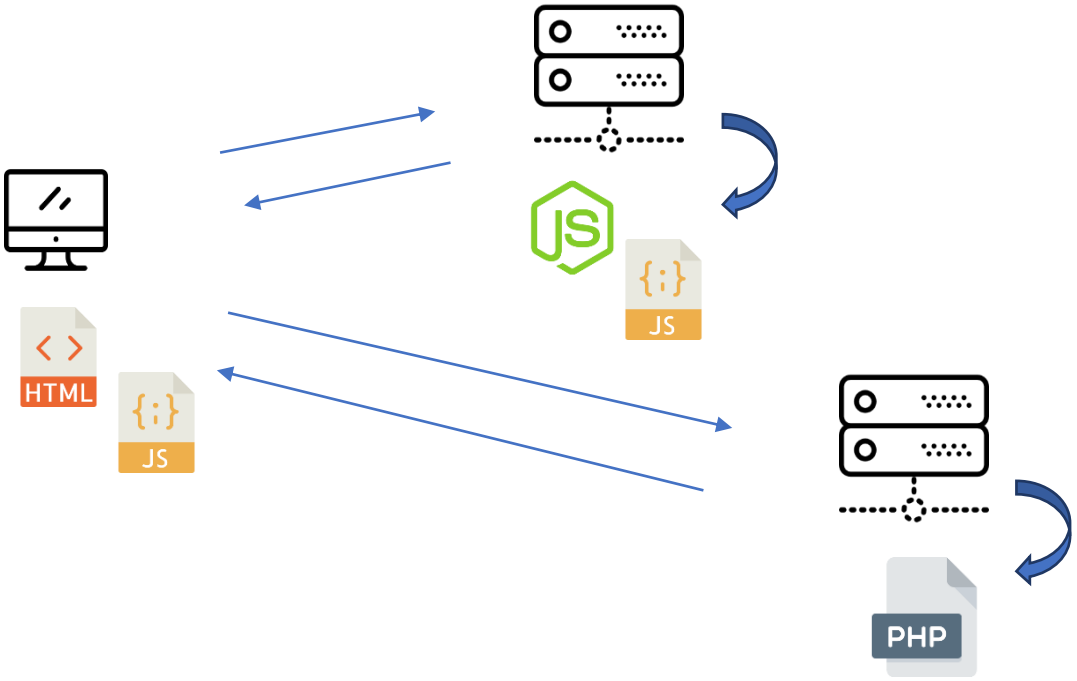
Le site web doit pouvoir récupérer les données des clients pour les afficher au médecin, que ce soit sous forme graphique ou en données brutes.

Le médecin doit pouvoir se connecter avec des identifiants pour lire les données de façon sécurisée.

Dans le contexte général, le serveur NodeJS s'inscrit entre la base de données et le site web. Les données sont envoyées sur le serveur python qui traite et enregistre les données sur une base de données mysql, ces données sont récupérées par le serveur NodeJs puis utilisées.

5.2. Pourquoi NodeJS

Avec le PHP le client envoie une requête au serveur, celui-ci va alors générer le code HTML et JavaScript, mais le code va être exécuté par le client. Le NodeJS est un langage de programmation proche (très proche du JavaScript). La grande différence avec le JavaScript est que NodeJS est du code côté serveur. Concrètement NodeJS nous permet d'écrire du JavaScript directement sur le serveur permettant d'accélérer certaines requêtes de clients. Avec du NodeJS, une partie du code JavaScript peut être généré directement par le serveur. Ceci permet par exemple de faire des serveurs de chats ou du "temps réel". Certaines requêtes sont alors plus rapides à exécuter.



Le NodeJS, nous le verrons, est un langage fonctionnant par événement. En effet quand une tâche longue est lancée, par exemple un upload de fichier. Le NodeJS va pouvoir exécuter la suite du code en attendant l'événement "Fin d'Upload", puis reprendre là où il en était. C'est l'utilité de ce langage. Dans notre cas cela peut servir à récupérer des données client en continu. Le médecin est en train de regarder les graphiques d'un client et les nouvelles données sont accessibles en direct au fur et à mesure que celle-ci sont ajoutée par le serveur Flask dans la Base de données.

5.3. Auto-Formation

Le NodeJS étant un langage que je ne connaissais pas, j'ai dû m'autoformer afin d'avancer dans le projet. Pour cela je me suis aidé de site tel que "OpenClassRoom" ou encore "Stack OverFlow". J'ai donc commencé par installer NodeJS et XWAMP une alternative à WAMP un peu plus efficace.

5.3.1.1. Les Réalisations

NodeJS fonctionne très différemment d'un serveur PHP tel qu'Apache. Avec une serveur PHP on écrit nos pages HTML, JavaScript, PHP dans un dossier. Avec NodeJS c'est un fichier .js qui va générer le serveur. Donc pour exécuter un serveur en PHP on lance WAMP, mais pour exécuter un serveur en NodeJS il faut lancer un le fichier .js depuis l'invité de commande NodeJS.

J'ai eu quelques difficultés après l'installation de NodeJS, une manipulation était nécessaire pour utiliser l'invité de commande.

5.3.1.2. Texte HTML dans une page web

J'ai commencé par l'affichage de texte (en HTML) dans une page web. Avec passage par url

```
1  var http = require('http'); //http
2  var url = require('url'); //récupère url
3  var querystring = require('querystring');
4
5  var server = http.createServer(function(req, res) {
6      var params = querystring.parse(url.parse(req.url).query); //met dans un tableau les paramètre parser grâce à url.parse()query qui parse les infos après le chemin
7      res.writeHead(200, {"Content-Type": "text/plain"}); //http 200 tout fonctionne
8      if ('prenom' in params && 'nom' in params) { //si le tableau param contient des valeurs dans nom et prénom
9          res.write('Vous vous appelez ' + params['prenom'] + ' ' + params['nom']); //on récupère les valeurs contenue au adresses nom et prenom
10     }
11     else {
12         res.write('Vous devez bien avoir un prénom et un nom, non ?');
13     }
14     res.end();
15 });
16 server.listen(8080);
```

Sur l'image si dessus nous pouvons voir un appel de bibliothèque commun à beaucoup de langage ici la librairie "http"la librairie "url"et la librairie "querystring".

Ici http nous permet de créer un serveur web. url nous permet de récupérer l'url et querystring est un module qui nous permettra de parser les données de l'url.

Comme nous pouvons le voir nous avons une fonction createServer à la ligne 5, la subtilité NodeJS est que cette fonction contient une autre fonction (de requête et réponse). createServer ne s'arrête qu'à la ligne 15.

Nous retournons ensuite dans la console le prénom et le nom récupéré depuis l'url grâce à des conditions.

5.3.1.3. Les événements

J'ai par la suite fait quelques exemples sur les événements, l'un des points les plus importants de NodeJS dont voici un exemple.

```
13  server.on('close', function() { // On écoute l'évènement close
14
15      console.log('Bye bye !');
16
17  })
18  server.listen(8080); // Démarre le serveur
19  server.close(); // Arrête le serveur. Déclenche l'évènement close
```

Ici l'événement écoute se déclenche une fois que le serveur est arrêter. La fonction server.on "attend" l'événement server.close() pour se déclencher. Grâce à cette gestion d'événement, nous pouvons effectuer plusieurs tâches, si une tâche est longue à s'exécuter, un événement à la fin de cette tâche pourra déclencher une autre action.

Nous pouvons aussi émettre un événement

```
1   var http = require('http');
2   var EventEmitter = require('events').EventEmitter;
3   var jeu = new EventEmitter();
4
5   var server = http.createServer(function(req, res) {
6       res.writeHead(200);
7       //res.end();
8
9           jeu.on('gameover', function(message){ //réception de l'événement gameover
10
11              console.log(message);
12          });
13
14          jeu.emit('gameover', 'Vous avez perdu !'); //emmission de l'événement
15      });
16  server.listen(8080); // Démarre le serveur
```

Ici il faut importer la librairie "event". Nous créons un serveur puis ligne 14 l'événement "gameover" contenant un message. Ici comme pour l'exemple précédent la ligne 9 intercepte l'évènement.

5.3.1.4. Les Modules

Les modules sont très utiles sur NodeJS lorsque j'ai besoin d'une librairie il me suffit de l'appeler comme vu ci-dessus, mais si je souhaite une librairie qui n'est pas de base on doit l'importer depuis l'invité de commande. Pour cela il suffit d'être à la racine du fichier et de taper `npm install mysql` dans l'invité de commande NodeJS. Dans le cas présent c'est le module mysql qui est installé, j'appellerai la librairie dans mon code grâce à `var mysql = require('mysql');` il est possible de créer son propre module, mais je n'en ai pas eu l'utilité pour mes programmes.

5.3.1.5. Les frameworks

Les framworks ont la même utilité qu'une bibliothèque. Ils sont juste plus développés. Ici nous utiliserons le framework Epress, qui s'installe de la même façon qu'un module. Il nous permettra de nous simplifier le passage d'information via url

5.3.1.6. *Les Sockets*

Nous commençons à entrer dans une partie intéressante de notre sujet. Lorsqu'on parle de web, l'utilisation classique est qu'un client demande une page, et que le serveur lui envoie le code nécessaire pour générer cette page (en HTML, CSS, JavaScript). Les sockets permettent non seulement cet usage classique, mais en plus, autorisent le serveur à envoyer des informations de lui-même. C'est le cas par exemple sur une application de messagerie lorsque l'on voit "*untel est en train d'écrire*". Dans notre application cela peut-être très utile lorsque le médecin est en train de consulter des données. Si de nouvelles données sont disponibles sur la base de données, il sera alors possible de mettre à jour automatiquement le graphique, sans avoir besoin de rafraichir la page.

Voyons l'exemple d'une utilisation de socket.

```
12  // Chargement de socket.io
13  var io = require('socket.io').listen(server);
14
15  // Quand un client se connecte, on le note dans la console
16  io.sockets.on('connection', function (socket) {
17      console.log('Un client est connecté !');
18  });
```

Ici nous chargeons le socket en créant une variable nous permettant la gestion de ses sockets. Nous avons un événement ".on" qui attend une connexion socket. Cette connexion arrive lorsque le client charge la page web si dessous. Dès que la ligne 17 est exécutée, l'événement dans le code NodeJS (ci-dessus) est intercepté.

```
14          <script src="http://code.jquery.com/jquery-1.10.1.min.js"></script>
15          <script src="/socket.io/socket.io.js"></script>
16          <script>
17              var socket = io.connect('http://localhost:8080');
```

Il est bien sûr (et c'est ce qui nous intéresse) possible d'envoyer un événement sur la page web grâce à une ligne du type :

```
23      socket.emit('message', 'C\'est ok t\'es bien connecté !');
```

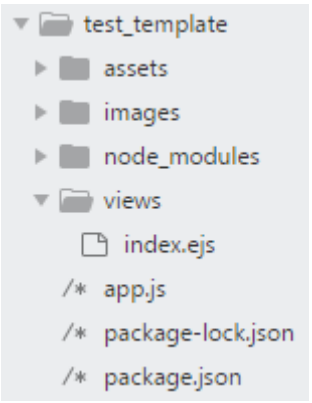
Qui sera affiché sur la page web.

5.3.2. Les pages web et les templates

Afin de pouvoir obtenir une page web correcte, il faut pouvoir y insérer des images et des styles. La méthode la plus utilisée est d'avoir un dossier pour les images et d'utiliser CSS.

Une méthode pour écrire du code HTML est de directement l'insérer dans le code NodeJS (voir Text HTML dans une page web). Seulement le code est presque illisible et la méthode n'est pas intéressante dès que la page commence à être fournie.

Il faut donc séparer le fichier NodeJS du fichier HTML. Ceci est possible grâce aux modules "EJS" et "Express". Dans notre dossier nous créons un sous dossier "views" dans lequel nous créons un fichier "index.ejs"



Lorsque l'on fera appel à index.ejs (le fichier contenant du HTML) NodeJS est configuré pour aller le chercher dans ce fichier. Nous pouvons aussi observer qu'"assets" et "images" deux dossiers classiques de la programmation web sont à la racine.

```
4   //middleware to use static files like css or images
5   app.use('/assets', express.static('assets'));
6   app.use('/images', express.static('images'));
7
8   app.get('/', function(req, res){
9       res.render('index.ejs');
10  });
```

Il ne faudra pas oublier l'installation du module ejs. De plus il faut indiquer dans le fichier NodeJS où se situent les dossiers "assets" et "images". Il suffit ensuite de demander à renvoyer le fichier "index.ejs" (ici pas besoin de spécifier le chemin /views).

5.3.3. MySQL

La connexion MySQL est une des composantes majeures de NodeJS. En effet l'objectif est de pouvoir se connecter à la base de données, d'y effectuer des requêtes et de traiter ses données pour une lecture auprès de l'utilisateur (ici le médecin).

Comme pour tout jusqu'ici il faut importer le module "mysql". Ici je vais présenter le code qui, lorsque le médecin appuie sur un bouton depuis la page web, le résultat de sa recherche lui soit affiché.

```
22  /***** MYSQL Connection *****/
23  var connection = mysql.createConnection({
24      //Connection to mysql
25      //properties of connection
26      host: 'localhost',
27      user: 'root',
28      password: '',
29      database: 'axelife'
30
31  });
```

Ici outre les modules importés plus haut dans le code, nous effectuons la connexion MySQL en entrant le nom de la base ainsi que les identifiants. (Nous retrouvons un principe de NodeJS avec le .createConnection, le principe est le même pour la création d'un serveur)

```
33  //check the connection
34  connection.connect(function(error){
35      //callback
36      if(error)
37      {
38          console.log('Error connection');
39      }
40      else{
41          console.log('Connected');
42      }
43  });
```

Nous effectuons ensuite une vérification de connexion grâce au .connect. En fonction du retour de la fonction nous écrivons dans la console l'état de la connexion.

Ensuite nous intégrons toujours dans ce même code un socket afin de pouvoir faire un échange avec l'utilisateur du site web. Plus haut dans le code nous renvoyons une page HTML (voir Template) et nous écrivons la fonction suivante

```
95      //même chose avec un autre boutton
96      socket.on('Age', function (res) {
97          connection.query("SELECT * FROM `users` WHERE PRENOM='Decher'", function (error, rows, fields){
98              if (error)
99              {
100                 //affiche sur la console
101                 console.log('Error in your query')
102             }
103             else
104             {
105                 //emmet un socket pour la page html
106                 socket.emit('message', rows[0].AGE);
107             }
108
109         });
110     });
```

Celle-ci s'inscrit dans un io.socket.on qui écoute le serveur. Lors d'un appui sur le bouton depuis la page web (voir"Les Sockets") cette action est interceptée par le socket.on ci-dessus. Nous effectuons alors une requête sur notre base de données où nous avons déjà effectué la connexion. Cette requête se passe par la fonction connection.query suivie de la requête SQL. Le résultat de cette requête sera alors renvoyé dans le "*rows*" qui est un tableau. Nous pouvons alors récupérer la donnée qui nous intéresse dans ce tableau de résultat. Ici nous choisissons l'âge (ligne 106) que nous renvoyons à la page web à l'aide de la fonction .emit (voir "Les Sockets").

5.4. Les Difficultés

J'ai rencontré plusieurs difficultés lors du projet. Lors de l'installation, certaines composantes ne s'étaient pas installées correctement.

J'ai consacré une grande partie de mon temps à l'autoformation. Je me suis beaucoup aidé d'internet notamment de "Slack Overflow" et de la documentation NodeJS pour ce qui est de la partie socket et mysql.

Les problèmes rencontrés on uniquement était dû au langage, je ne vais donc pas détailler quelle partie du code ont posé problème puisqu'il s'agissait de problème dû à l'apprentissage du langage et de son fonctionnement.

Le problème le plus important auquel j'ai fait face est le temps. En effet, le fait de ne pas connaitre le langage m'a obligé à passer du temps sur la compréhension de ce dernier afin de pouvoir écrire le code souhaité.

5.4.1. Les Améliorations

J'ai rapidement été conscient que je ne pourrais pas fournir un site fonctionnel à 100%. C'est pour cela que j'ai préféré passer plus de temps à détailler mon code et à le segmenter. Pour cela j'ai fait plusieurs dossiers contenant, pour les derniers, un package.json détaillant les modules installés ainsi que leur version. Le fait d'avoir plusieurs dossiers permet, à la personne qui le reprend, de pouvoir les utiliser comme des briques et dès les assemblés comme il le souhaite. Ceci afin de ne pas avoir à faire le travail de recherche et de débogage auxquels j'ai dû faire face pour réussir à faire ces briques.

Pour ce qui est de l'axe d'amélioration, je pense que j'aurais pu gagner du temps au début sur certaines parties en passant plus vite dessus. Elles ont cependant été utiles pour une meilleure compréhension de la suite.

5.5. Bilan NodeJS

Le bilan de NodeJS. J'ai tout de suite compris que les attentes initiales étaient trop grandes par rapport au temps dont nous disposions pour le réaliser. Je suis en revanche satisfait du travail réalisé. En effet quelqu'un qui reprend le dossier sera en capacité de continuer le projet. J'ai pris le temps de segmenter le travail en dossier. Ainsi, chaque dossier permet de réaliser une ou plusieurs fonctions en NodeJS. Par exemple l'ajout d'un Template HTML, une connexion MySQL, une requête sur la base de données, réception d'interaction entre le client et le serveur. Il est actuellement possible que, lors d'un appui sur un bouton (depuis la page web), le serveur intercepte le l'événement, ce dernier réalise une requête SQL sur la base donnée. La réponse de la base de données est ensuite traitée par le serveur puis renvoyée à la page web où l'utilisateur va pouvoir lire les données qu'il a demandé.

6. Bilan

Les objectifs de ce projet étaient de réaliser une connexion entre une application Android et un server python. Celui-ci a pour but de manipuler des fichiers CSV et d'implémenter une base de données. Cette base de données sera consultée par un serveur NodeJS afin de répondre aux requêtes d'une application web.

La connexion avec l'application mobile est seulement à l'état de prototype.

Les deux serveurs ont été implémentés de façon ce que les fonctionnalités nécessaires soient sur différents fichiers, un fichier pour une fonctionnalité. C'était notre conception de l'agilité pour ce projet afin de faciliter la reprise de notre travail par d'autres étudiants ou l'entreprise elle-même.

La connexion entre Android et le serveur Flask ainsi qu'une procédure de test sont les prochaines étapes à réaliser dans notre projet. Pour la partie NodeJS il s'agit maintenant d'intégrer la connexion MySQL est un Template, mais aussi un traitement des données en HTML pour un affichage graphique. Il reste aussi la partie authentification du médecin afin de sécuriser l'accès aux données clients. Pour la partie MySQL, il faut ajouter un système d'authentification sécurisé et compléter les différentes tables. Pour le serveur Flask, il faut développer une méthode permettant de traiter le fichier csv de données brut.

Ce projet nous a permis d'apprendre de nouveaux langages. Le fait que le projet soit trop important par rapport au nombre d'heures nous a appris à gérer les priorités dans le temps imparti. Nous avons pu travailler en équipe, mais aussi en collaboration avec une entreprise. Nous sommes satisfaits du travail accompli même si nous aurions aimé pouvoir aller plus loin.

7. Annexes

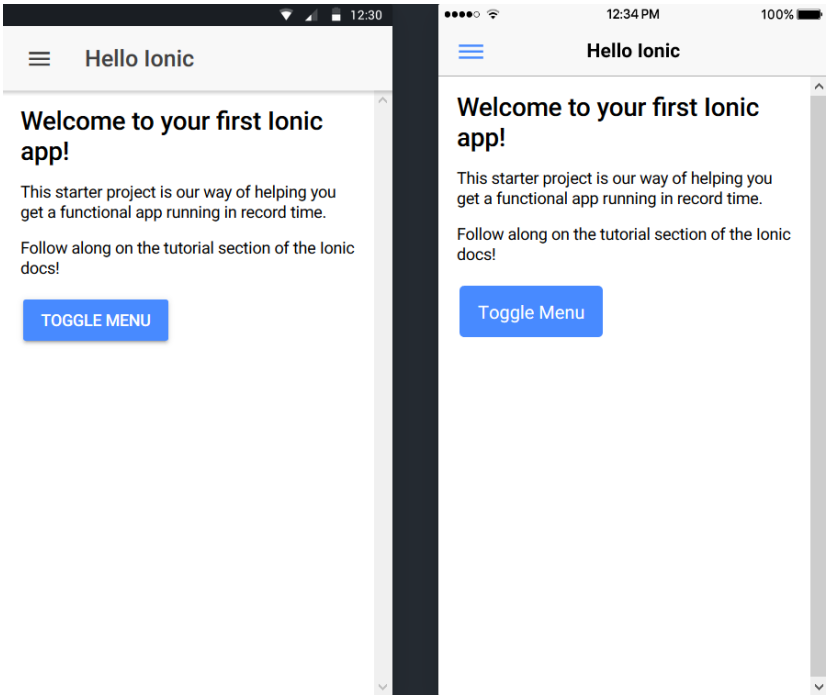


Figure 1 - Application type ionic

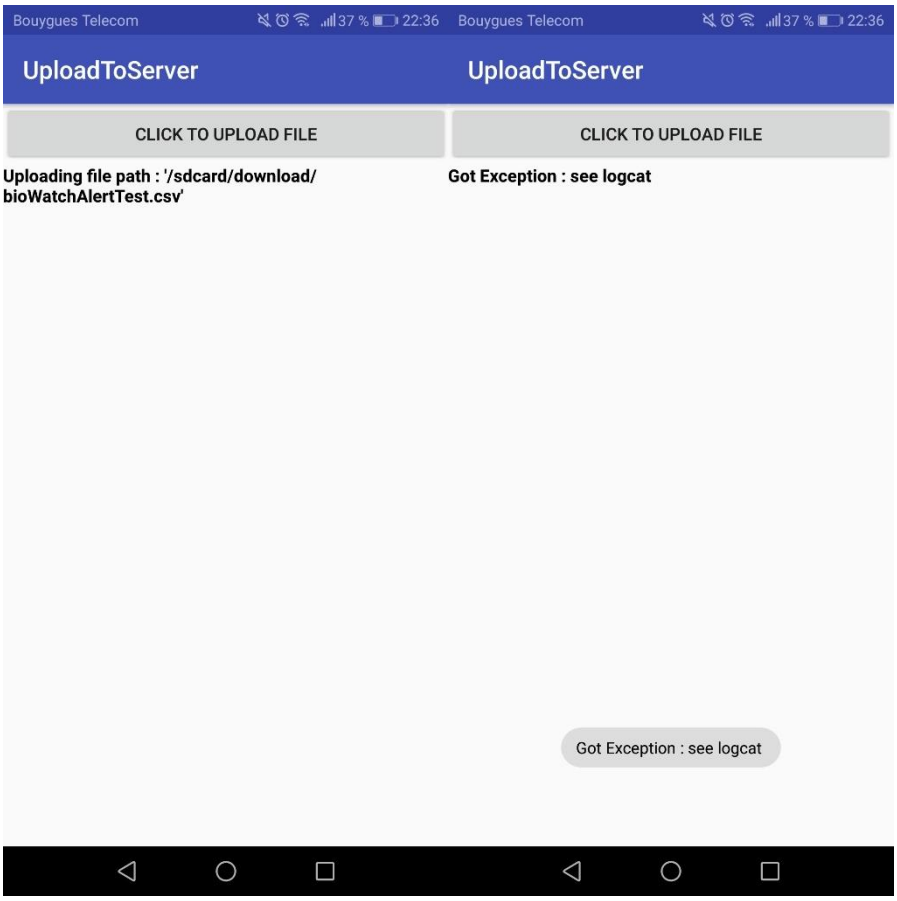


Figure 2 - Application type Android Natif

```
13  # on cré un dictionnaire (TABLES) de requête pour créer les tables nécessaire
14  TABLES = {}
15
16  TABLES['Medecin'] = (              #Requête pour créer la table medecin
17      "CREATE TABLE IF NOT EXISTS Medecin ("
18      "Nom varchar(20),"
19      "Prenom varchar(20),"
20      "Mail varchar (50),"
21      "Telephone varchar(20),"
22      "ID bigint(20),"
23      "FOREIGN KEY (ID) REFERENCES users(ID)"
24      ")"
25
26  )
27
```

```
28  TABLES['user_data'] = (                #Requête pour créer la table user_data
29      "CREATE TABLE IF NOT EXISTS user_data ("
30      "Dates date PRIMARY KEY,"
31      "Horaire time,"
32      "bpm int,"
33      "csp int,"
34      "cdp int,"
35      "psp int,"
36      "pdp int,"
37      "si int,"
38      "oxymetrie int,"
39      "temp varchar(5),"
40      "Alert_PSP int,"
41      "Alert_BPM int,"
42      "Alert_CSP int,"
43      "Alert_PDP int,"
44      "Alert_SI int,"
45      "Alert_Oxy int,"
46      "Alert_Temp int,"
47      "LimitBasse_BPM int,"
48      "LimitHaute_BPM int,"
49      "LimitBasse_CSP int,"
50      "LimitHaute_CSP int,"
51      "LimitBasse_CDP int,"
52      "LimitHaute_CDP int,"
53      "LimitBasse_PSP int,"
54      "LimitHaute_PSP int,"
55      "LimitBasse_PDP int,"
56      "LimitHaute_PDP int,"
57      "LimitBasse_SI int,"
58      "LimitHaute_SI int,"
59      "LimitBasse_Oxy int,"
60      "LimitHaute_Oxy int,"
61      "LimitBasse_Temp int,"
62      "LimitHaute_Temp int,"
63      "Latitude varchar(10),"
64      "Longitude varchar(10),"
65      "City varchar(255),"
66      "ID bigint(20),"
67      "FOREIGN KEY (ID) REFERENCES users(ID)"
68      ")"
69  )
```

Figure 3 - Requêtes création de tables

8. Bibliographie

[1]

« Android : faire des requêtes HTTP simplement | SUPINFO, École Supérieure d'Informatique ». [En ligne]. Disponible sur: https://www.supinfo.com/articles/single/2592-android-faire-requetes-http-simplement.

[2]

« Angular - User Input ». [En ligne]. Disponible sur: https://angular.io/guide/user-input.

[3]

Drifty, « Ionic Framework ». [En ligne]. Disponible sur: https://ionicframework.com/docs/.

[4]

« MySQL :: MySQL Connector/Python Developer Guide :: 5.4 Querying Data Using Connector/Python ». [En ligne]. Disponible sur: https://dev.mysql.com/doc/connector-python/en/connector-python-example-cursor-select.html.

[5]

« MySQL :: MySQL Connector/Python Developer Guide :: 7.1 Connector/Python Connection Arguments ». [En ligne]. Disponible sur: https://dev.mysql.com/doc/connector-python/en/connector-python-connectargs.html.

[6]

« MySQL :: MySQL Connector/Python Developer Guide :: 10.2.6 MySQLConnection.cursor() Method ». [En ligne]. Disponible sur: https://dev.mysql.com/doc/connector-python/en/connector-python-api-mysqlconnection-cursor.html.

[7]

« Upload File To Server - Android Example ». [En ligne]. Disponible sur: https://androidexample.com/.

[8]

F. Kaddouri, « Uploader un fichier sur un serveur avec OKHttp », 20-sept-2015.

[1]

« JSON.parse() », MDN Web Docs. [En ligne]. Disponible sur: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/JSON/parse. [Consulté le: 12-avr-2018].

[2]

« npm ». [En ligne]. Disponible sur: https://www.npmjs.com/. [Consulté le: 12-avr-2018].

[3]

« Stack Overflow - Where Developers Learn, Share, & Build Careers ». [En ligne]. Disponible sur: https://stackoverflow.com/. [Consulté le: 12-avr-2018].

[4]

« Une première application avec Node.js », OpenClassrooms. [En ligne]. Disponible sur: https://openclassrooms.com/courses/des-applications-ultra-rapides-avec-node-js/une-premiere-application-avec-node-js. [Consulté le: 12-avr-2018].

[5]

« YouTube ». [En ligne]. Disponible sur: https://www.youtube.com/watch?v=-lRgL9kj_h0. [Consulté le: 12-avr-2018].

Bracelet connecté

Projet réalisé par : Karaman Mickael, Eddebbi Matthias, Lemaitre Antoine

Projet encadré par : M.Franck Mouney & M.Jean-Baptiste Fasquel

Résumé

L'objectif de ce projet est d'aider l'entreprise Axelife à développer son système d'information, plus précisément la connexion entre l'application et le serveur Flask, la gestion de base de données ainsi que l'affichage des données sur une application web via un serveur NodeJS. Ainsi un patient portant le bracelet pourra être alerté en cas de problème imminent, au cas échéant le médecin et/ou le contact d'urgence seront eux aussi alertés. Dans ce rapport il est expliqué la méthodologie appliquée, les technologies utilisées, les difficultés rencontrées ainsi que les axes d'amélioration à développer.

Mots-Clés

Autoformation / Web services / Serveur / Base de données / Android / Medical / Objet connecté / Système d'information

Summary

The objective of this project is to help Axelife company to develop its information systems more specifically the connection between the Android application and the Flask server, the management of data base and the interpretation of data with a web application which use a NodeJS server. In this case a patient wearing the wristband can be alert in case of an imminent problem, the doctor and/or the emergency contact will be alerted. In this report it's explained the methodology applied, the technology used, the difficulties met and the axis for improvement for a future development.

Key Word

Self-Study / Web services / Server / Database / Android / Medical / Connected Objects / Information System