

JANVIER 2022



OXIMETRE DE POULS

PROJET S7

ACHRAF LAAZIZI | HATIM LAKCHILI
INSA EUROMED
UEMF

Table des matières

Introduction	2
I. Frontend	3
1. Théorie	3
a) Principe d'oxymétrie de pouls.....	3
b) Schéma de l'oxymètre.....	9
c) Description du matériel	9
2. Pratique	15
a) Tests	15
b) Montage de l'oxymètre	19
c) ESP8266 Feather HUZZAH.....	21
II. Backend	26
1. Outils utilisés	26
a) Logiciels	26
b) Librairies	26
2. Communication entre application et MAX30105.....	27
a) GUI (JAVA).....	27
b) Lecture des données(MYSQL)	33
c) Envoi de données(C & PHP)	35
Conclusion	42
Bibliographie	43

Introduction

L'oxymétrie est un test non invasif et indolore mesurant le niveau de saturation en oxygène dans votre sang. Il peut analyser rapidement l'efficacité avec laquelle l'oxygène est acheminé vers les parties du corps les plus éloignées du cœur, tels que les doigts et les orteils.

Notre projet consiste à construire un oxymètre de pouls à base de circuit électronique en utilisant une carte microcontrôleur et un capteur de rythme cardiaque et de saturation d'oxygène.

L'intérêt de notre projet n'est pas de créer un capteur qui fonctionne et qui renvoie des valeurs de saturation d'oxygène et de rythme cardiaque, mais qu'il agit comme un objet connecté.

Notre approche d'oxymétrie de pouls se démarque par l'établissement d'une plateforme IoT. Les valeurs mesurées par le capteur ne vont pas simplement être affichées dans un écran. Les informations sur l'utilisateur vont être communiquées à une application consultable qui servira à stocker, traiter et afficher ces informations afin d'offrir une meilleure compréhension de la situation de santé des patients.

Le travail sur ce projet combinera des connaissances dans le domaine des circuits électroniques, la programmation des cartes microcontrôleurs ainsi que la programmation orientée objet pour la préparation de l'application.

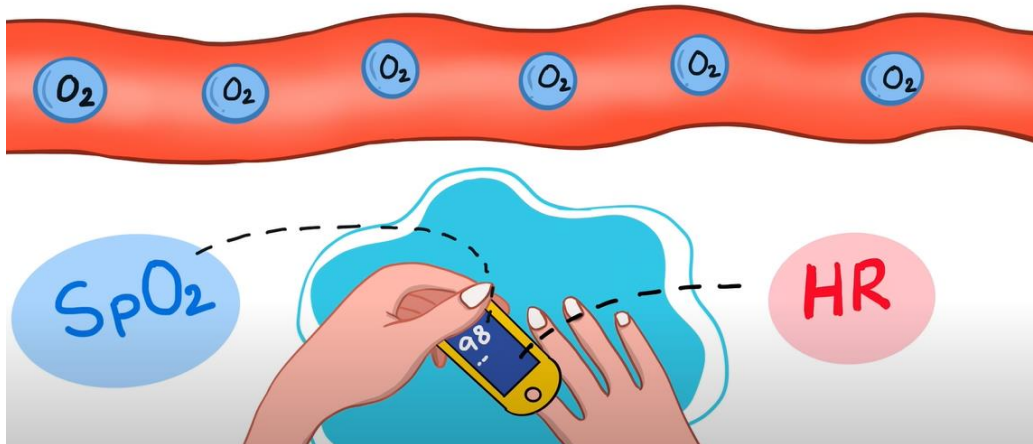
Le résultat du projet fera objet du lien entre une partie Frontend, où on traitera tout ce qui suit avec la construction l'oxymètre de pouls et sa programmation, avec une partie Backend qui se présente comme une application de traitement de données envoyés depuis l'oxymètre.

I. Frontend

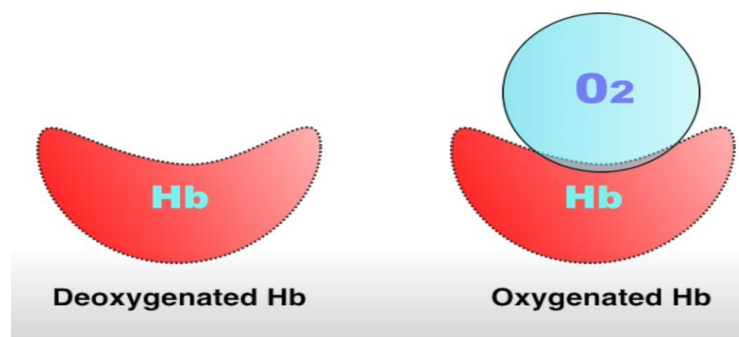
1. Théorie

a) Principe d'oxymétrie de pouls

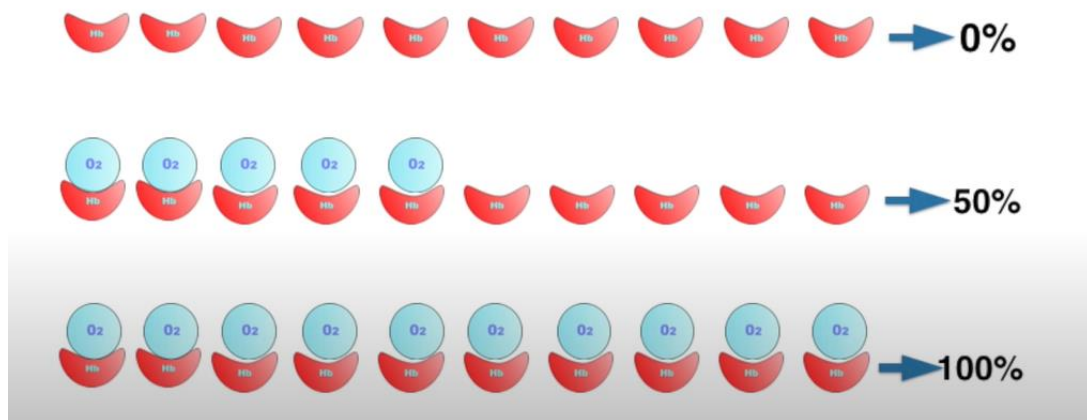
L'oxymètre de pouls est un appareil utilisé pour surveiller la saturation d'oxygène d'un individu en mesurant son niveau dans le sang. La grandeur retournée s'appelle « SpO2 ». Le rythme cardiaque de la personne est automatiquement calculé à côté de l'SpO2.



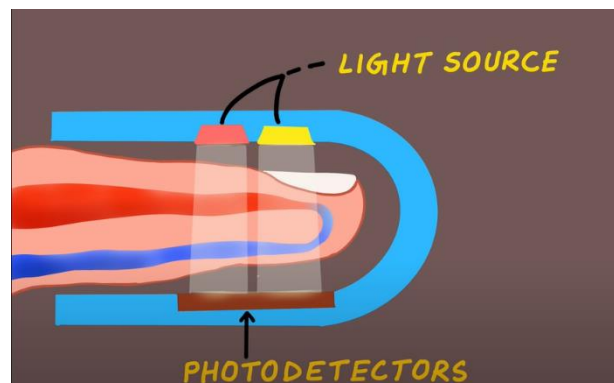
Avant d'aller vers le mécanisme, nous devons savoir ce que c'est que la saturation d'oxygène. L'oxygène, nécessaire pour le fonctionnement de tous les organes du corps, est transporté par le sang. L'oxygène est particulièrement déplacé par les molécules d'Hémoglobine. Les molécules d'Hémoglobine contenant de l'oxygène sont appelées « Hémoglobine oxygénée », tandis que celles ne contenant pas d'oxygène s'appellent « Hémoglobine désoxygénée ».



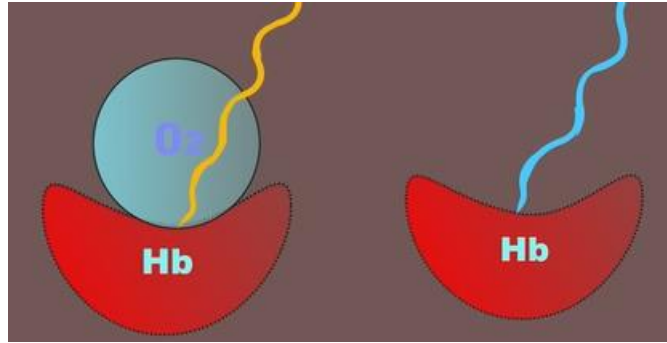
La saturation d'oxygène est simplement le pourcentage des molécules d'Hémoglobine oxygénées sur la totalité des molécules d'Hémoglobine. A titre d'exemple, Si aucune molécule ne transporte de l'oxygène, la saturation est de 0%. Si 5 entre 10 molécules transportent de l'oxygène, la saturation est de 50%.



Passons au mécanisme. L'oxymètre de pouls utilise la lumière pour calculer la saturation d'oxygène. La lumière est émise depuis la source et traverse le doigt de la personne pour arriver au détecteur de lumière

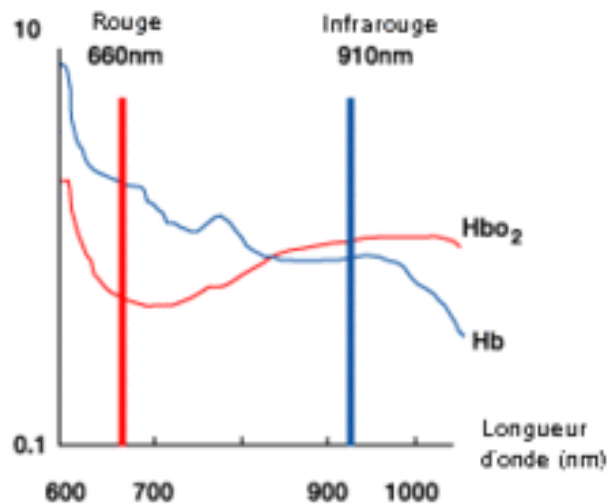


Une partie de la lumière émise est absorbée par le doigt et la partie non absorbée restante est reçue au détecteur de lumière. L'oxymètre utilise la propriété qui dit que l'Hémoglobine oxygénée et désoxygénée absorbent différentes quantités de lumière et de différentes longueurs d'onde.

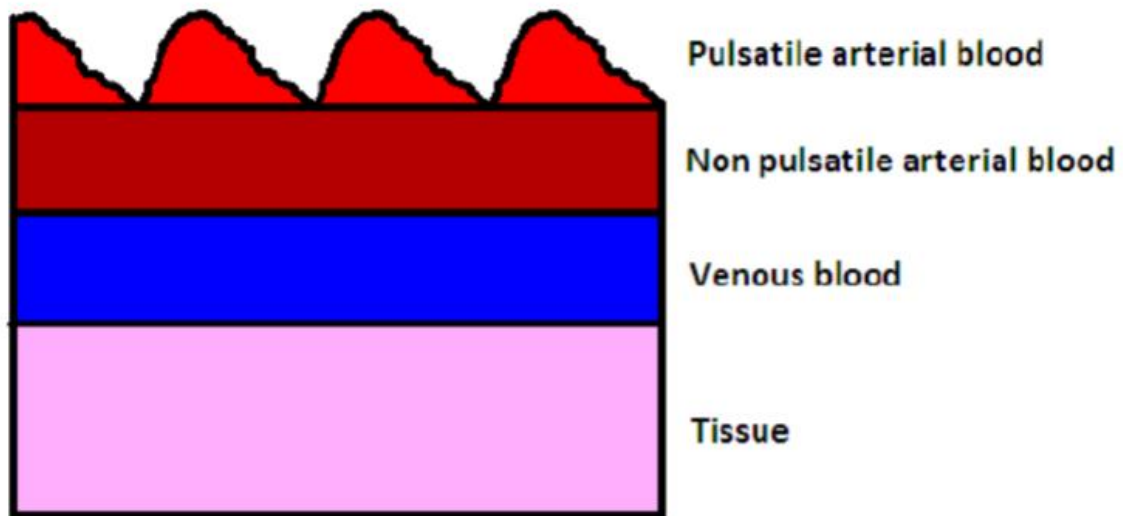


L'oxymètre émet deux lumières ; Lumière rouge (640nm) et lumière infrarouge (960nm). La lumière rouge est absorbée par l'hémoglobine désoxygénée plus que l'hémoglobine oxygénée, tandis que pour la lumière infrarouge le cas est contraire.

Voici un graphe montrant l'évolution du coefficient d'absorption en fonction de la longueur d'onde lumineuse



La composante DC représente la lumière absorbée des tissus, du sang veineux et du sang artériel non pulsatile. La composante AC représente l'artère pulsatile du sang.



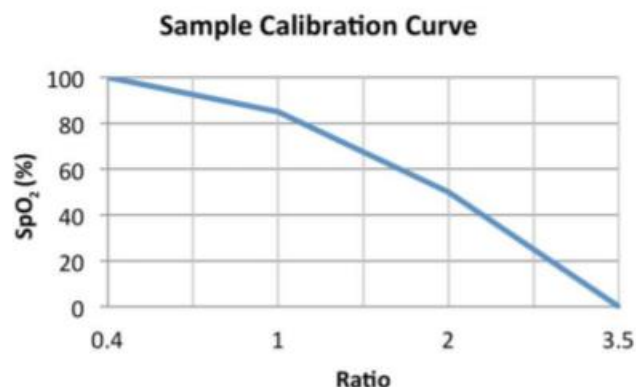
Un double rapport entre les deux types de lumière est calculé avec la formule suivante :

$$R = \frac{A_{red(AC)} / A_{red(DC)}}{A_{IR(AC)} / A_{IR(DC)}}$$

La SpO₂ peut être déterminée à l'aide de la valeur du rapport et d'une table de consultation composée de formules empiriques.

Le rythme cardiaque peut être calculé sur la base du numéro d'échantillon et taux d'échantillonnage du convertisseur analogique-numérique de l'oxymètre de pouls (ADC).

Voici un exemple de courbe donnant la valeur de la SpO₂ en fonction du rapport R :



On montre également les valeurs du rythme cardiaque et de la SpO2 prévue en fonction de l'âge et l'état sanitaire des individus :

Rythme cardiaque :

Age	Normal Pulse Rate
Neonate (<28 days)	100-205
Infant (1 month - 1 year)	100-190
Toddler (1-2 years)	98-140
Pre-school (2-5 years)	80-120
6-11 years	75-118
12 years - adult	60-100
Athlete	40-60

SpO2 :

Casualty	SpO2
Normal - Healthy	≥ 94%
Normal - COPD	88% - 92%
Hypoxic	85% - 93%
Severely Hypoxic	< 85%

SpO2 supérieur ou égale à 94% : Personne en bonne santé

SpO2 entre 88% et 92% : Personne atteinte de la maladie chronique MPOC (La Maladie Pulmonaire Obstructive Chronique)

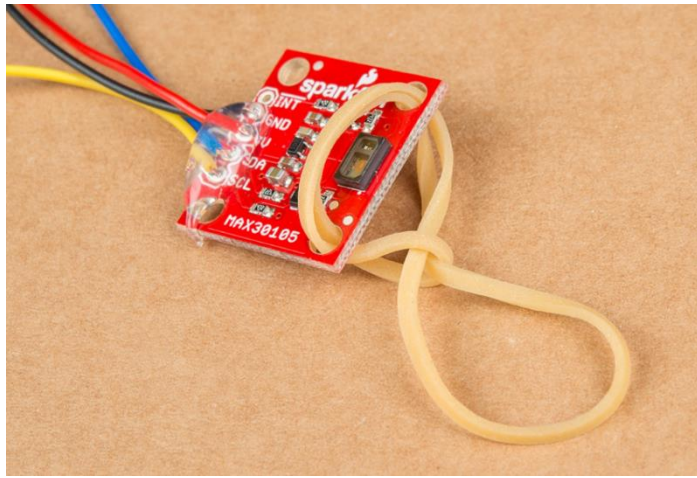
SpO2 entre 85% et 93% : La personne est atteinte de l'hypoxie

SpO2 inférieur à 85% : Cas extrême d'hypoxie.

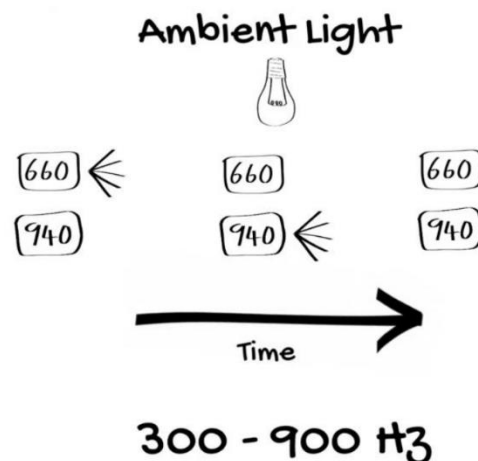
Précautions de mesure :

Afin d'obtenir une bonne lecture d'information depuis le doigt de l'utilisateur, quelques précautions sont à prendre à priori.

On a d'abord **la pression d'appui du doigt** sur la plateforme. En règle générale, une pression plus élevée entraîne une estimation de la SpO2 plus faible que prévu. Similairement, une pression faible n'aboutira pas à une bonne lecture de valeurs. C'est pour cela qu'on voit souvent des élastiques attachés aux capteurs de la SpO2 qui servent à maintenir une pression raisonnable qui permet le bon calcul pendant tout le temps de lecture.



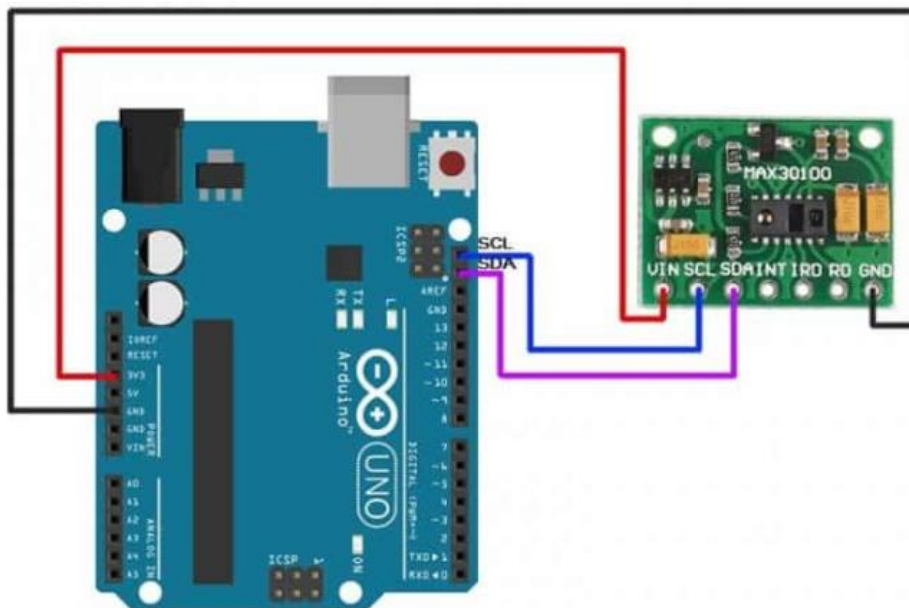
On a aussi à essayer d'éliminer les interférences des sources lumineuses ambiantes. L'oxymètre est conçu d'une façon à rejeter les lectures depuis les lumières ambiantes. Les faisceaux lumineux rouges et infrarouges sont émis un après l'autre mais pas en même temps, et ensuite s'éteignent simultanément, et le cycle recommence. Durant le temps où les deux LED sont éteintes, l'oxymètre calcule la quantité de la lumière ambiante reçue dans le photodétecteur. Après, l'oxymètre peut retirer la partie ambiante depuis l'intégralité du signal reçu.



La correction de la lumière ambiante effectuée par l'oxymètre est souvent insuffisante. C'est pour cela conseillé d'effectuer les tests dans un endroit sombre ou d'utiliser un outil pour couvrir la lumière extérieure d'intervenir.

b) Schéma de l'oxymètre

Voici le schéma qu'on doit suivre en construisant notre oxymètre :

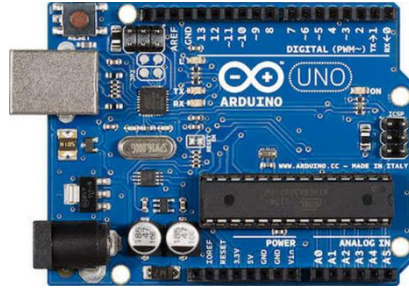


Comme c'est montré au schéma, le pin de 3V d'alimentation est connecté aux VIN du capteur, le GND au GND, le SDA avec le pin digital 4 et le SCL avec le pin digital 5. Le pin digital 4 est pour le passage d'information entre le capteur et l'Arduino UNO, alors que le pin digital 5 est le signal horloge.

c) Description du matériel

Notre oxymètre de pouls se composera de :

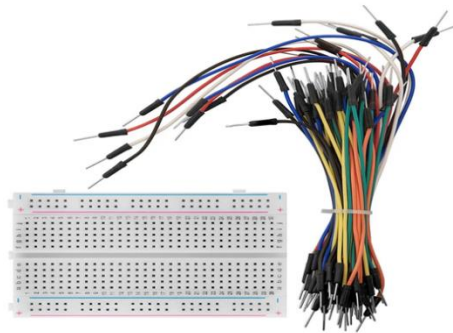
Carte Arduino UNO:



Capteur d'oxymétrie de pouls et de fréquence cardiaque : Le MAX30105



Notre platine d'expérimentation et les fils de connexion :



Afin de comprendre le fonctionnement des composants principales de l'oxymètre (Carte Arduino, Capteur MAX30100, LCD), faisons-en une étude détaillée.

Commençons par la carte Arduino UNO.

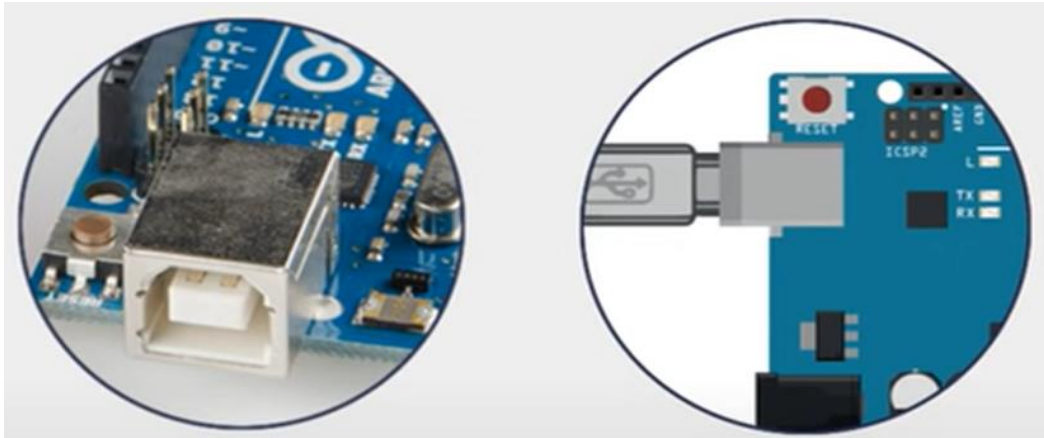
La carte Arduino UNO est une carte de prototypage électronique open source basée sur un microcontrôleur qu'on peut programmer facilement à l'aide d'un Arduino IDE relativement simple.

La carte Arduino possède à la fois un circuit physique programmable ainsi qu'une partie logicielle. L'IDE utilise une version simplifiée du langage C++.

La UNO est l'une des carte Arduino les plus populaire qui se présente comme un bon choix pour les débutants.

L'Arduino UNO consiste de 9 composants majeurs, avec chacun son rôle et son utilité :

❖ Le connecteur USB



Le connecteur USB est utilisé pour téléverser le programme depuis l'IDE vers la carte Arduino. Il peut également servir comme source d'alimentation pour la carte.

❖ Prise d'alimentation



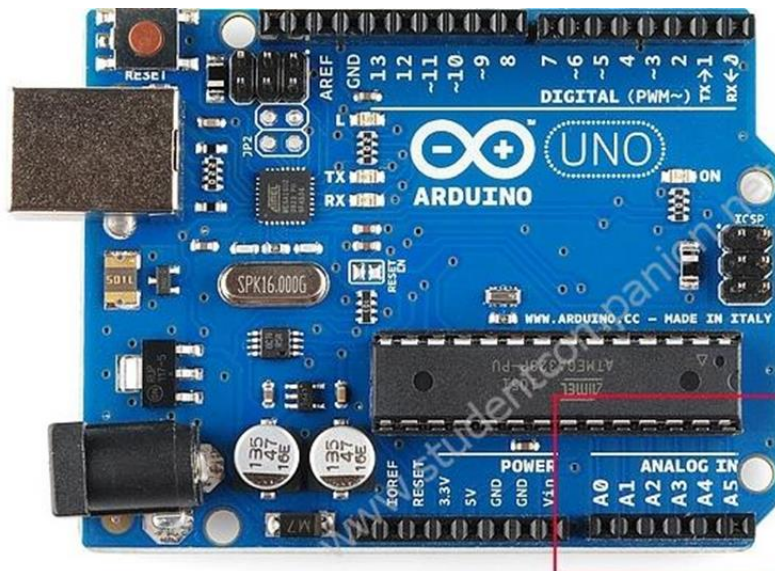
Comme son nom l'indique, cette prise est aussi dédiée à l'alimentation de la carte. Cette dernière ne supporte pas une tension de plus de 20 volts et possède un régulateur de tension qui la protège de l'épuisement.

❖ Microcontrôleur



Il s'agit du rectangle noir le plus visible dans la carte, se comportant comme le cerveau de l'Arduino. Il possède une mémoire dans laquelle le programme est stockée depuis l'IDE ainsi qu'une mémoire RAM qui aide le CPU à compléter les instructions du programme. Il possède aussi une mémoire morte qui garde les informations en protection même après redémarrage.

❖ Broches d'entrée analogique

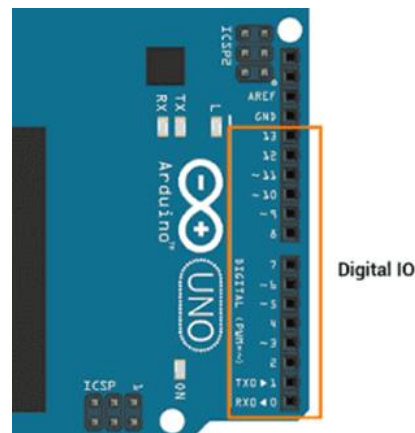


La carte Arduino UNO possède 6 broches d'entrée analogiques numérotés de A0 à A5. Ces broches servent à lire le signal depuis un capteur analogique tel le capteur de température, et le transformer ensuite en un signal numérique pour que le système le comprenne.

Ces broches peuvent aussi être utilisées comme des entrées et des sorties numérique (Digital Input/Output) même s'ils sont par défaut classés comme des entrées analogiques.

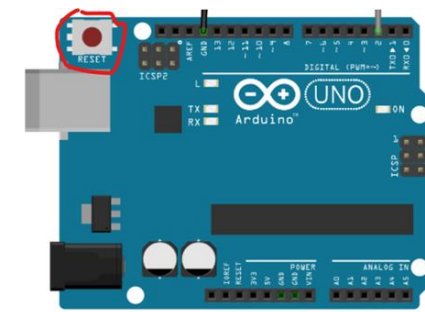
❖ Broches digitales

Ces broches sont numérotées de 0 à 13. Lorsqu'ils sont utilisés comme broches Output, ils agissent comme sources d'alimentation. S'ils sont utilisés comme broches Input, ils lisent le signal depuis les périphériques connectés.



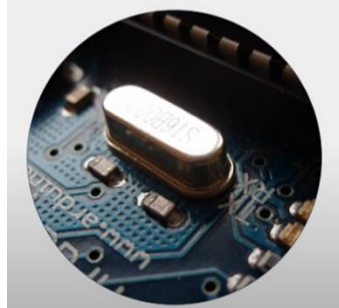
❖ Bouton de réinitialisation

Ce bouton réexécute le programme une deuxième fois lorsqu'il est cliqué. Cela peut être utile si on souhaite tester un programme qui ne se répète pas plusieurs fois.



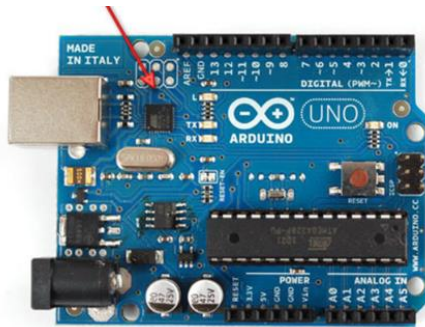
❖ Oscillateur à Crystal

Cet oscillateur oscille à une vitesse de 16 millions de fois par seconde. Le microcontrôleur fait une opération simple chaque oscillation.



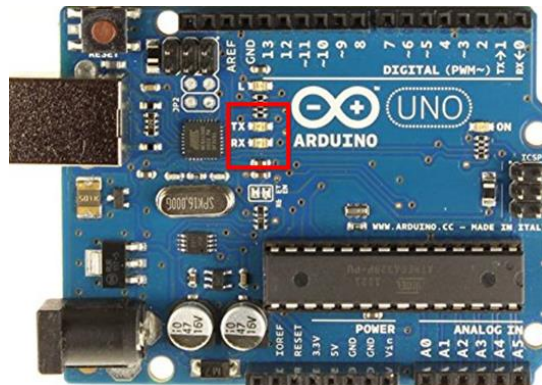
❖ Puce d'interface USB

Cette puce traduit le signal du niveau USB à un niveau que la carte Arduino comprend.

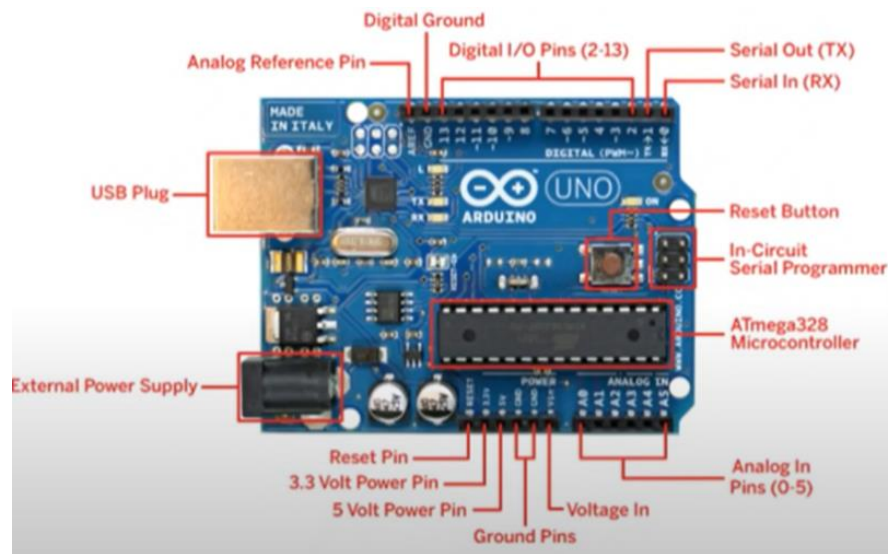


❖ Indicateur TX/RX

Ce sont deux LEDs qui clignotent. Une LED TX de transmission, clignote lorsque la carte Arduino transmet de l'information et une LED RX de réception qui clignote lorsque la carte Arduino reçoit de l'information.



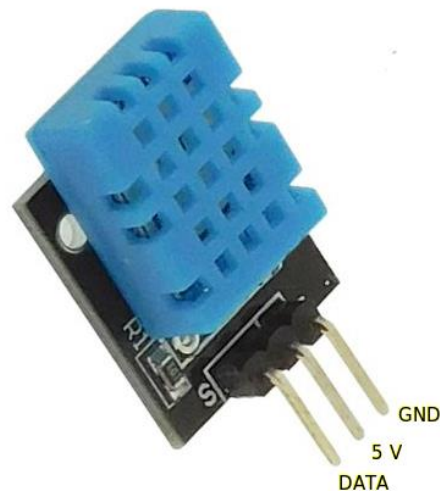
Voici un schéma de la répartition de ces composants à travers la carte :



2. Pratique

a) Tests

Capteur de température et d'humidité



Pour ce test, on n'aura pas besoin d'utiliser le Breadboard.

Le capteur de température possède 3 pins: Le pin positif, le pin de l'entrée et le pin négatif.

On connectera le pin positif avec le pin d'alimentation 5V de l'Arduino, le pin négatif au Ground et le pin d'entrée à un pin digital quelconque de l'Arduino. On utilisera le pin digital 2 dans ce cas.

Voici le code Arduino utilisé pour ce test:

```
#include <SimpleDHT.h>
#include <Wire.h>
#include <LiquidCrystal_I2C.h>

// for DHT11,
//      VCC: 5V or 3V
//      GND: GND
//      DATA: 2
int pinDHT11 = 2;
SimpleDHT11 dht11(pinDHT11);
void setup() {
    Serial.begin(9600);
}
void loop() {
    // start working...
    Serial.println("=====");
    Serial.println("Sample DHT11...");

    // read without samples.
    byte temperature = 0;
    byte humidity = 0;
    int err = SimpleDHTErrSuccess;
    if ((err = dht11.read(&temperature, &humidity, NULL)) != SimpleDHTErrSuccess) {
        Serial.print("Read DHT11 failed, err="); Serial.println(err);delay(1000);
        return;
    }
    Serial.print("Sample OK: ");
    Serial.print((int)temperature); Serial.print(" *C, ");
    Serial.print((int)humidity); Serial.print(" H ");

    delay(1500);
}
```

Résultat dans le serial monitor de l'Arduino IDE:

```
COM3
=====
Sample DHT11...
Sample OK: 32 *C, 73 H
=====
Sample DHT11...
Sample OK: 32 *C, 73 H
=====
Sample DHT11...
Sample OK: 32 *C, 73 H
=====
Sample DHT11...
Sample OK: 32 *C, 73 H
=====
Sample DHT11...
Sample OK: 32 *C, 73 H
=====
```

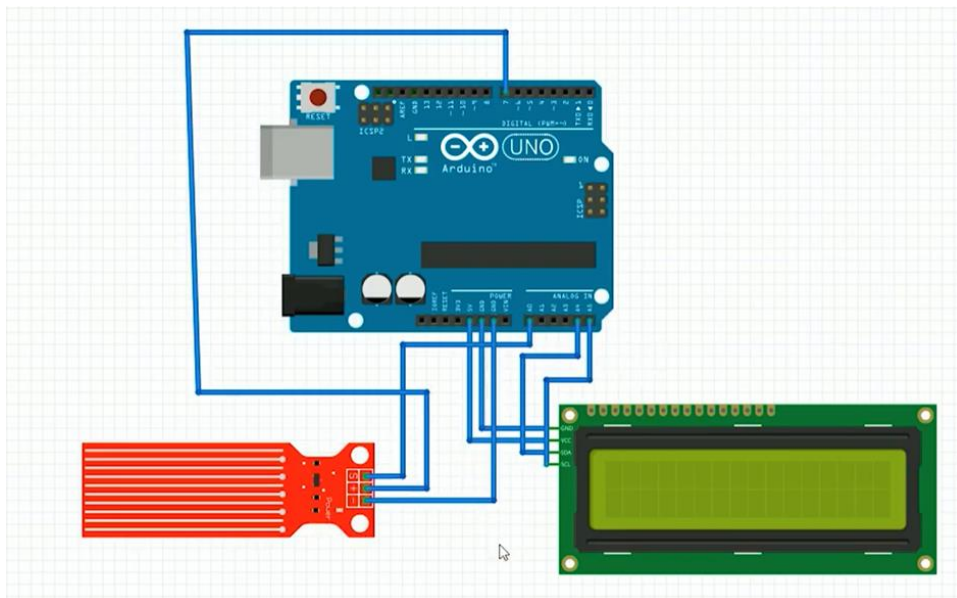
Résultat après avoir soufflé de l'air dans le capteur:

```
Sample OK: 32 *C, 74 H
=====
Sample DHT11...
Sample OK: 32 *C, 95 H
=====
Sample DHT11...
Sample OK: 33 *C, 95 H
=====
Sample DHT11...
Sample OK: 33 *C, 95 H
=====
Sample DHT11...
Sample OK: 34 *C, 95 H
=====
Sample DHT11...
Sample OK: 34 *C, 95 H
=====
Sample DHT11...
Sample OK: 34 *C, 95 H
=====
```

Capteur de niveau d'eau



Pour le câblage, on suit ce schéma :



Pour ce test, on n'aura pas besoin d'utiliser la Breadboard, mais on utilisera quand même puisque cette fois on introduira un LCD (Liquid Crsystal Display).

Voici le code adopté:

```

#include <Wire.h>
#include <LCD.h>
#include <LiquidCrystal_I2C.h>

LiquidCrystal_I2C lcd(0x3F, 2, 1, 0, 4, 5, 6, 7, 3, POSITIVE);

int lowerThreshold = 310;
int upperThreshold = 510;

// Sensor pins
#define sensorPower 7
#define sensorPin A0
int val = 0; // Value for storing water level

void setup() {
  Serial.begin(9600);
  lcd.begin(16,2);
  lcd.backlight();
  pinMode(sensorPower, OUTPUT);
  digitalWrite(sensorPower, LOW);
}

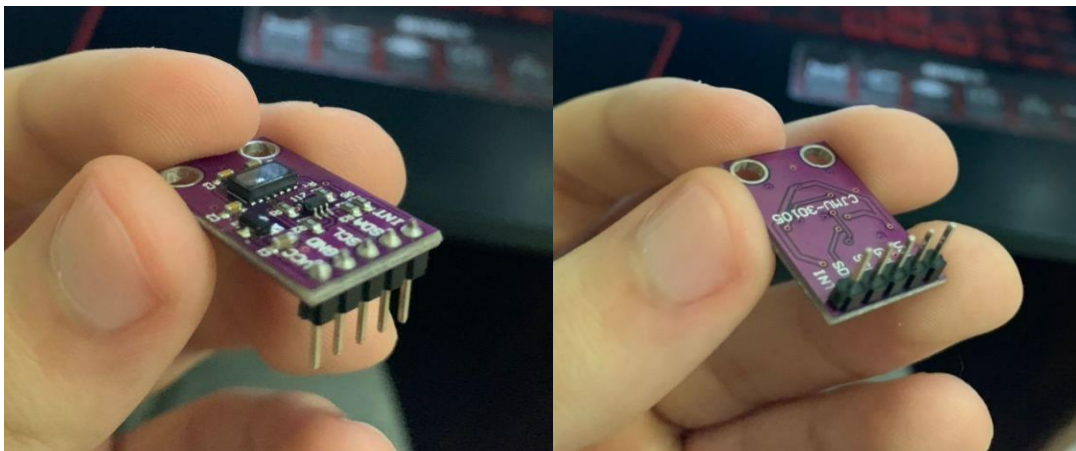
void loop() {
  int level = readSensor();

  if (level == 0) {
    Serial.println("Water Level: Empty");
    lcd.setCursor(0,0);
    lcd.print("  WATER LEVEL ");
    lcd.setCursor(0,1);
    lcd.print("    EMPTY    ");
  }
  else if (level > 0 && level <= lowerThreshold) {
    Serial.println("Water Level: Low");
  }
}

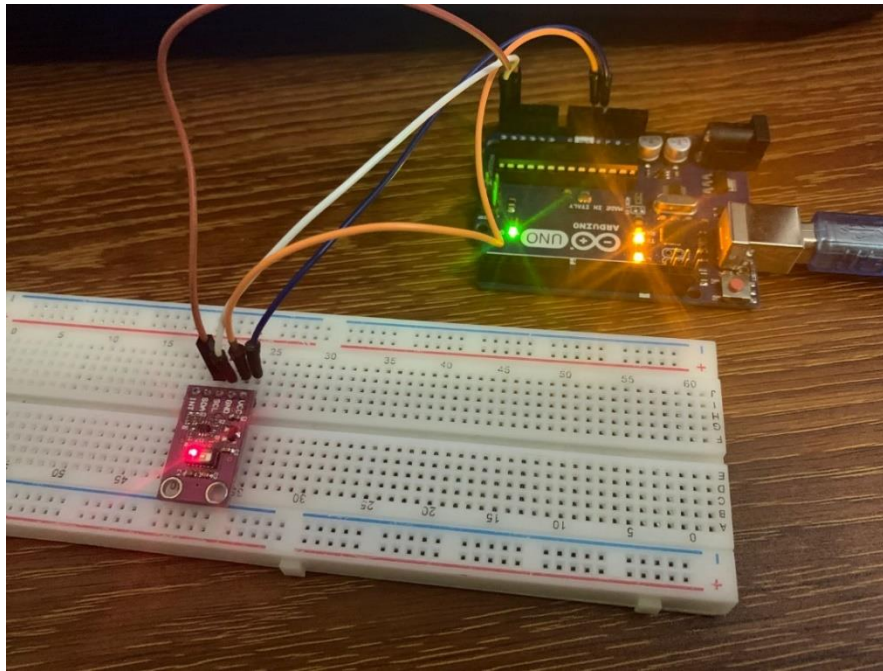
```

b) Montage de l'oxymètre

Voici le capteur MAX30105 avec lequel nous avons procédé :



Pour le câblage, on a suivi le même schéma indiqué avant malgré le fait que les deux capteurs sont légèrement différents. Le pin d'alimentation VCC avec le 3.3V de la carte, le GND avec le GND pour fermer le circuit, le signal d'horloge SCL avec le pin A5 et le signal de passage d'information SDA avec le pin A4.



On a utilisé le code nécessaire pour la mesure du rythme cardiaque et de la saturation d'oxygène. Après compilation et téléversement de ce dernier, le LED dans le capteur s'allume et commence à lancer les faisceaux lumineux.

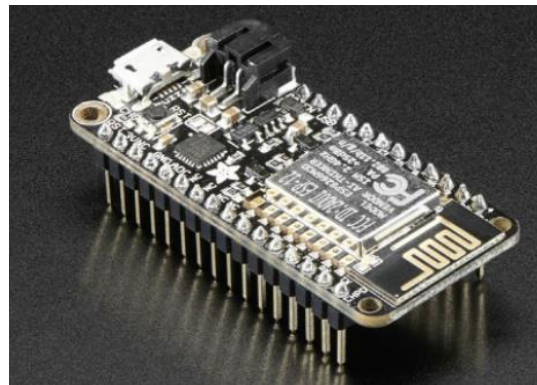
Après ouverture du Serial Monitor et en cliquant sur Send, les valeurs se lisent et on obtient cela comme résultat :

```
red=49280, ir=44158, HR=115, HRvalid=1, SPO2=100, SPO2Valid=1  
red=49281, ir=44158, HR=115, HRvalid=1, SPO2=100, SPO2Valid=1  
red=49270, ir=44168, HR=115, HRvalid=1, SPO2=100, SPO2Valid=1  
red=49301, ir=44176, HR=115, HRvalid=1, SPO2=100, SPO2Valid=1  
red=49249, ir=44152, HR=115, HRvalid=1, SPO2=100, SPO2Valid=1  
red=49115, ir=43764, HR=115, HRvalid=1, SPO2=100, SPO2Valid=1  
red=49014, ir=43787, HR=115, HRvalid=1, SPO2=100, SPO2Valid=1  
red=49040, ir=43912, HR=100, HRvalid=1, SPO2=100, SPO2Valid=1  
red=49020, ir=43932, HR=100, HRvalid=1, SPO2=100, SPO2Valid=1  
red=49040, ir=44027, HR=100, HRvalid=1, SPO2=100, SPO2Valid=1  
red=49031, ir=44090, HR=100, HRvalid=1, SPO2=100, SPO2Valid=1  
red=49025, ir=44113, HR=100, HRvalid=1, SPO2=100, SPO2Valid=1  
red=48903, ir=43707, HR=100, HRvalid=1, SPO2=100, SPO2Valid=1  
red=48923, ir=43873, HR=100, HRvalid=1, SPO2=100, SPO2Valid=1
```

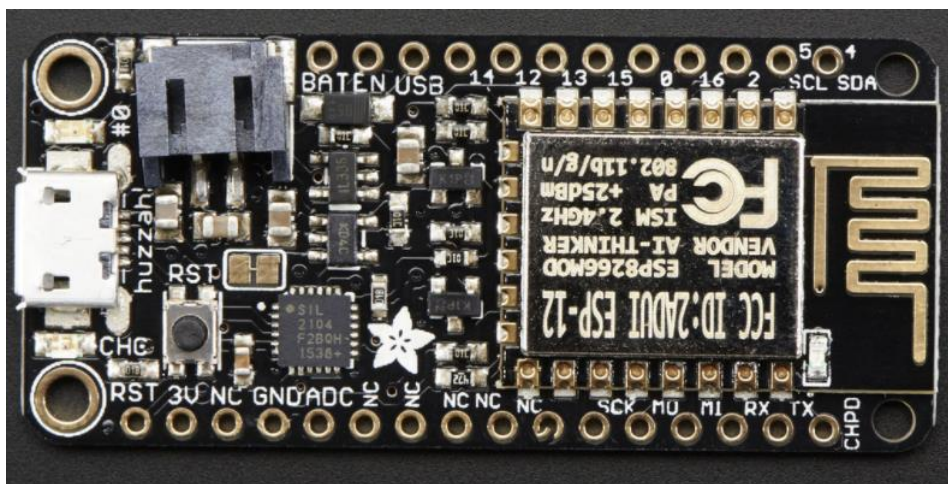

Les valeurs RED et IR ne nous intéressent pas puisqu'ils ont à avoir avec la lumière. On obtient donc des valeurs de rythme cardiaque et de SpO2 valides.

c) ESP8266 Feather HUZAH

Puisqu'il nous manque un équipement appelé "Ethernet Shield" pour envoyer l'information depuis l'Arduino vers Une base de données et faire marcher notre application, l'ESP8266 se présente comme un alternatif pour l'Arduino en possédant une carte Wi-Fi de plus permettant l'échange d'information sans fil.

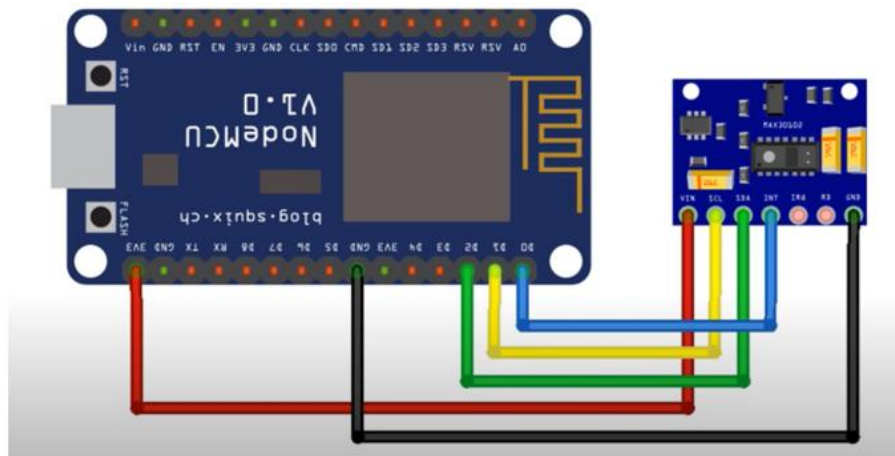


Voici une photo montrant les différents pins d'une carte microcontrôleur ESP8266 Feather Huzzah :



Au cœur du Feather HUZZAH se trouve un microcontrôleur WiFi ESP8266 cadencé à 80 MHz et en logique 3.3V. Ce microcontrôleur contient un cœur de puce Tensilica ainsi qu'une pile WiFi complète. On arrive à programmer le microcontrôleur à l'aide de l'IDE Arduino pour un noyau Internet des objets facile à exécuter.

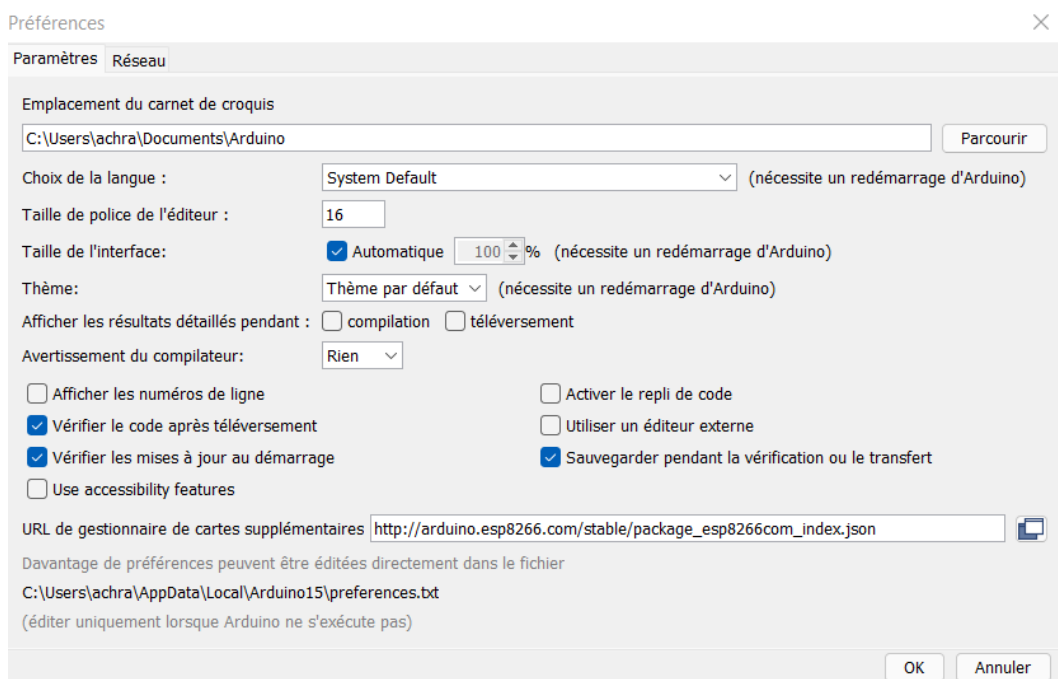
Pour le câblage du Feather HUZZAH avec l'oxymètre, on a procédé avec les mêmes pins que le montage avec la carte Arduino sauf que le in INT est dans ce cas utilisé.



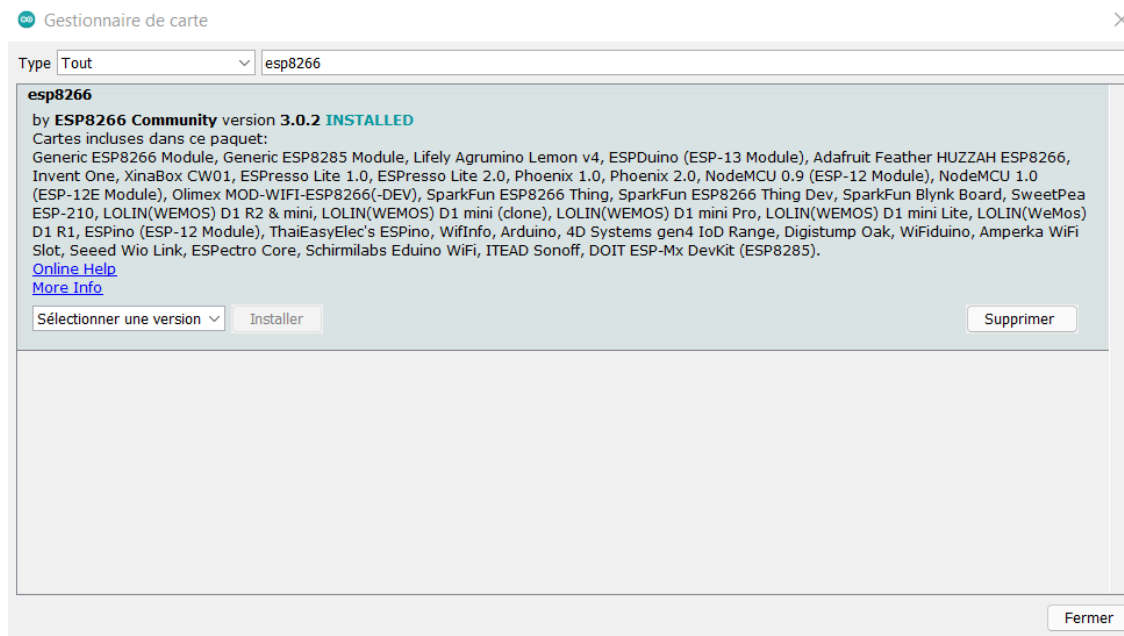
Le VIN/VCC se relie avec le 3.3V, le GND avec le GND de la carte, le signal d'horloge SCL avec D1, le signal d'information SDA avec D2 et le INT avec le D0.

En ce qui concerne la programmation par l'IDE Arduino, il y a eu quelques opérations à faire avant débuter de coder telles l'installation du driver SiLabs CP2104, du package de la carte ESP8266 et des librairies utiles.

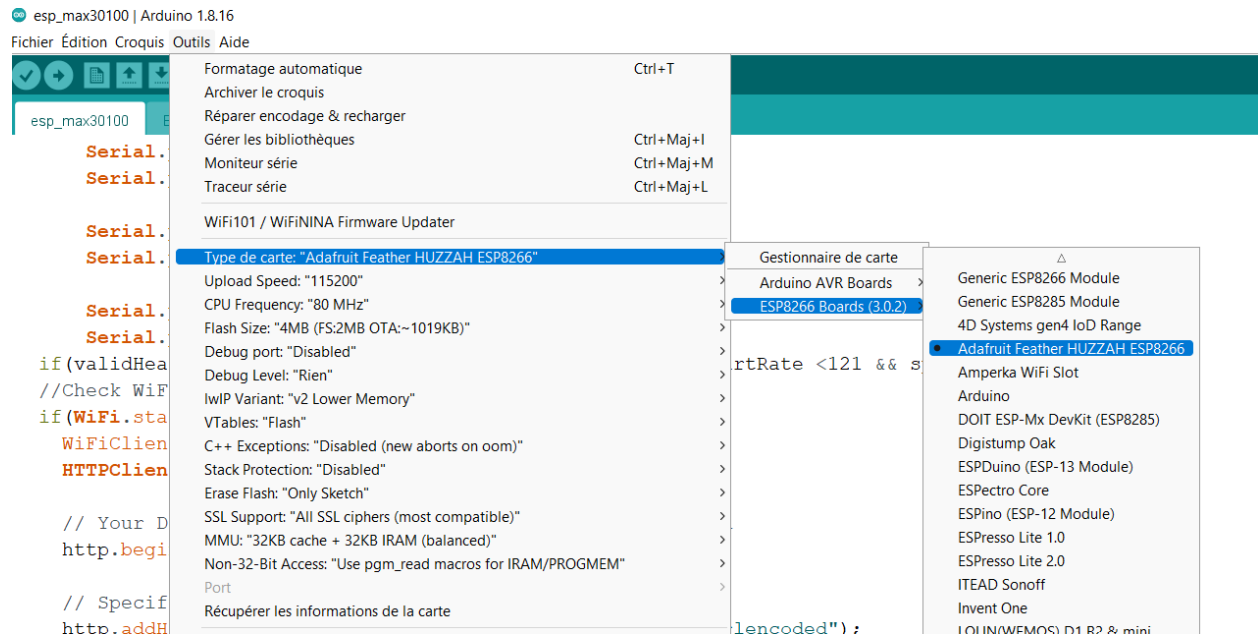
Pour installer le package, on entre son lien dans le champ : URL de gestionnaire de cartes supplémentaires.



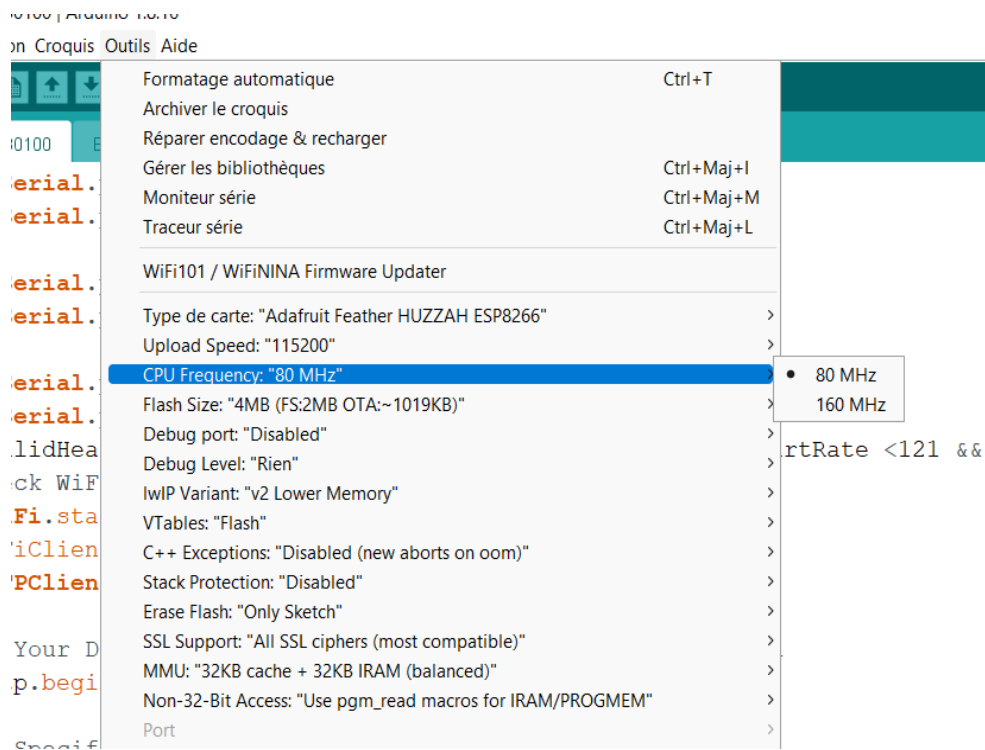
Ensuite, on utilise le gestionnaire des cartes pour installer le package en question.

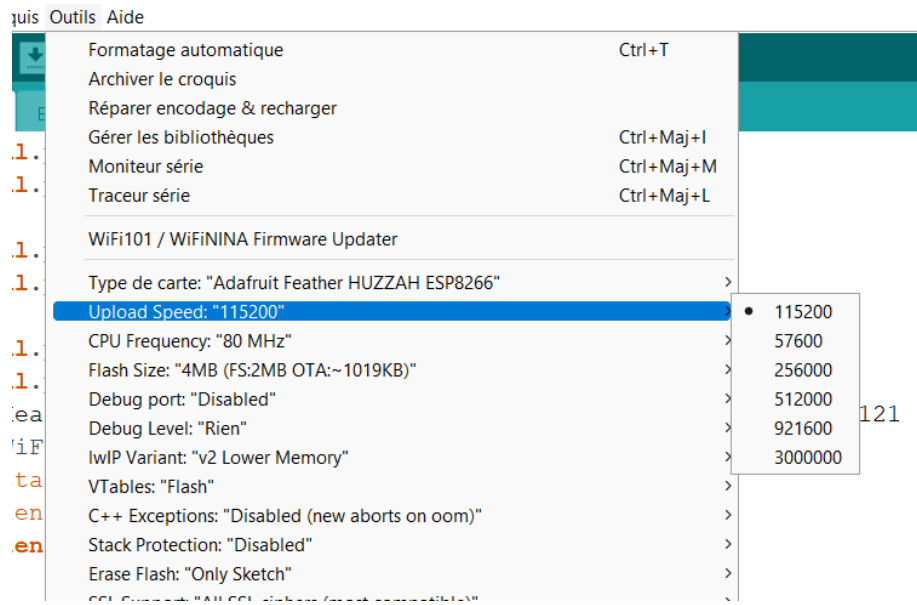


On sélectionne ensuite le nom exact de notre carte depuis les outils

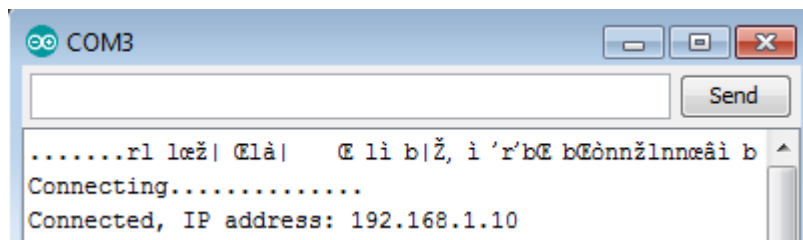


On vérifie ensuite quelques paramètres comme la fréquence du CPU et la vitesse du téléversement...





Note : La vitesse de téléversement est importante car elle détermine si on peut afficher les données ou pas avec notre port, au cas contraire , on aurait eu un problème similaire à :



On pourra maintenant commencer à coder notre programme et le téléverser à la carte.

Le code utilisé pour la carte Feather HUZZAH ressemble à celui utilisé pour la carte Arduino pour la mesure de l'SpO2 et du rythme cardiaque. Il a des lignes de codes supplémentaires qui servent à établir la connexion à un serveur pour pouvoir envoyer les informations vers un appareil appartenant à ce serveur.

Nous aborderons le code en plus de détails dans la partie Backend.

II. Backend

Dans le cadre de notre développement de projet IoT, on a opté pour utiliser le langage Java/JavaFx, qui est assez connue dans le domaine de développement des Graphic User Interfaces (GUI).

Note : Nous n'avons pas pris les dernières versions des logiciels, car plusieurs de nos librairies externes utilisées côté design ne sont pas mis à jour, et sont donc non-compatibles avec les versions récentes.

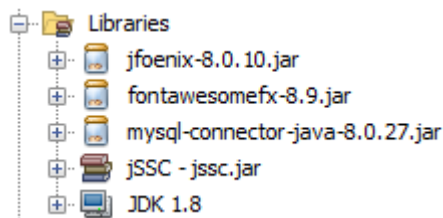
Bien évidemment, le Backend comprend aussi la partie de « réception des données du capteur », on a donc aussi utilisé du langage C (connexion entre l'ESP8266/Arduino), du PHP (envoi des données) et du SQL (Base de données utilisateurs + réception de données)

1. Outils utilisés

a) Logiciels

- Eclipse, NetBeans, ou IntelliJ sont tous des choix d'IDE compatibles avec le langage choisi, notre choix de NetBeans est uniquement par préférence.
- Entre SceneBuilder et Swing, on a trouvé que SceneBuilder était tout simplement plus intuitif, et tout simplement plus efficace avec ses fonctionnalités drag-and-drop.
- Arduino IDE (Communication entre capteur et ordinateur)
- Visual Studio Code (PHP, HTML)
- WAMPserver64 (Notre base SQL avec PHPMyadmin)

b) Librairies

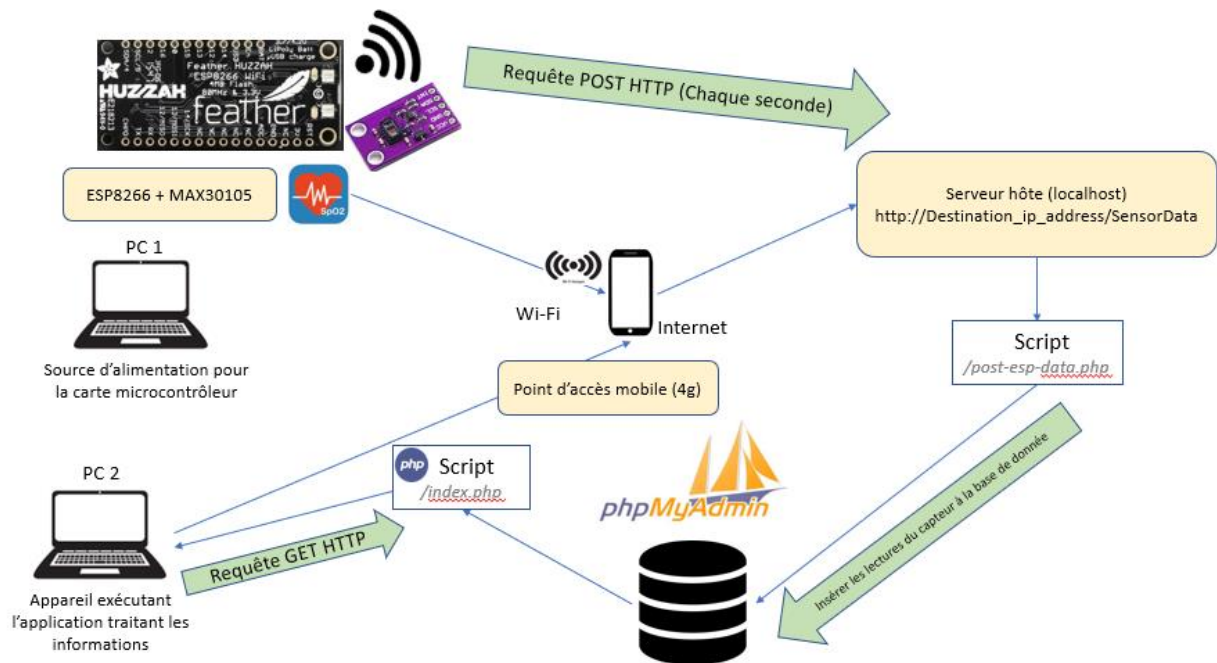


- Jfoenix comporte plusieurs sélections disponibles sur SceneBuilder, mais qui sont déjà était Stylisé en CSS, et sont donc très pratique pour optimiser le temps que cette personnalisation des Labels, buttons, etc. aurait pu prendre.

- Fontawesome est similaire, mais pour les icônes.
- Mysql-connector, comme son nom l'indique, permet la communication entre notre code avec la base de données.
- jSSC était une librairie qui aurait permis la communication entre code Java et Arduino, mais non utilisé dans le contexte IoT car elle dépend d'une connexion filaire.
- JDK 1.8 est notre librairie JAVA de choix, compatible avec les librairies précédentes.

2. Communication entre application et MAX30105

Avant de commencer , voici un schéma sur le parcours des données dans notre projet :

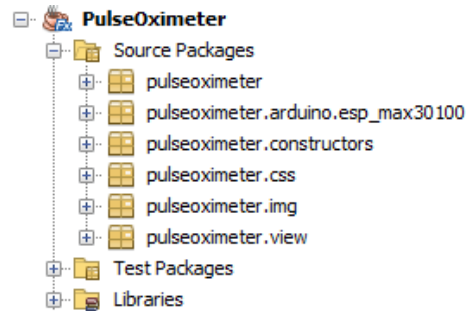


Par la suite on va essayer de décrire le plus pertinemment possible, les fonctions et bouts de codes utilisés.

Note : Pour tester l'application, ouvrir 'PulseOximeter\dist' du fichier zip, et sélectionner PulseOximeter.jar

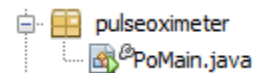
a) GUI (JAVA)

L'arborescence de notre projet final ressemble à ceci

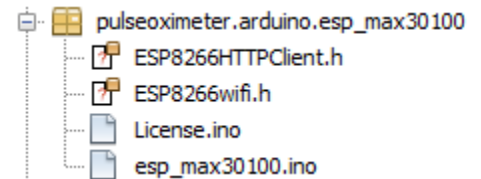


Les packages sont nommés intuitivement :

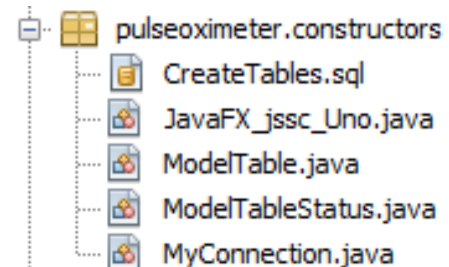
pulseoximeter : contient le main compilable qui lance notre application



pulseoximeter.arduino.esp_max30100 : Nommé ainsi car les fichiers .ino doivent appartenir à un dossier du même nom, le fichier est rajouté pour pouvoir accéder à l'IDE d'Arduino à partir de notre application.

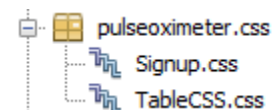


pulseoximeter.constructors : contient des fonctions qui seraient utilisées plusieurs fois dans un projet, dans quel cas il est judicieux de faire appel aux constructeurs pour éviter les répétitions et réduire le temps d'exécution.



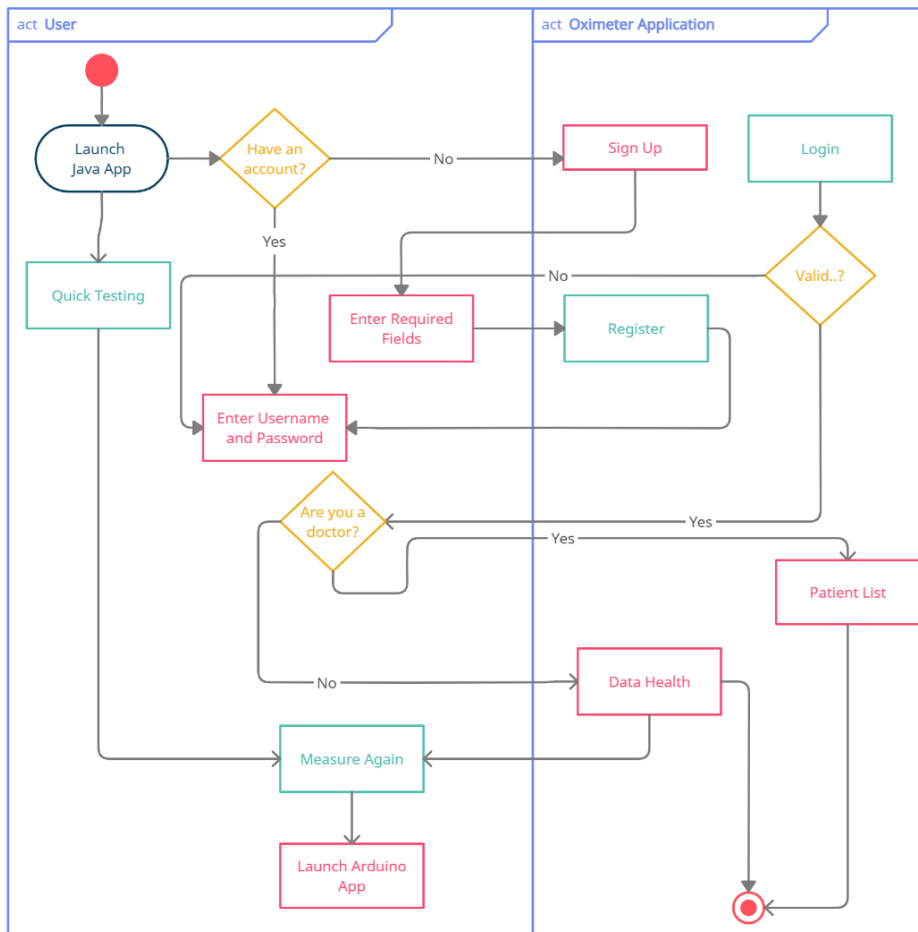
MyConnection.java : mention spéciale car c'est ce bout de code qui permet la connexion à notre base de données.

pulseoximeter.css : nos fichiers de stylisation (notamment pour les tableaux, buttons, textes, etc.)



pulseoximeter.img : images utilisés par notre GUI

pulseoximeter.view : où on a regroupé nos fichiers FXML, ouvrables par le logiciel SceneBuilder, ainsi que leur contrôleurs, qui permette de donner vie aux fonctionnalités insérer dans notre GUI, en associant chaque composant à une fonction.



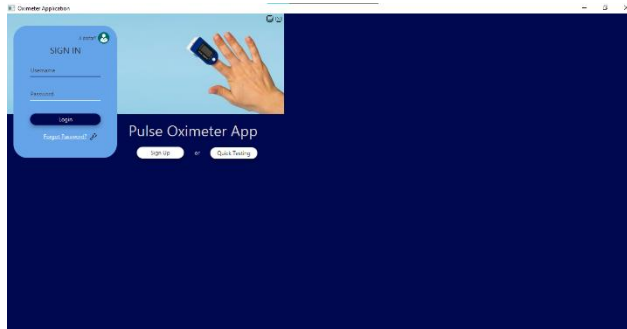
Ci-dessus est le diagramme d'activité qui éclaire l'utilisation principale de l'application.

On passe maintenant à la partie « exécution » du code, que se passe-t-il quand on lance l'application ?

```

public class PoMain extends Application {
    public static Stage stage = null;
    @Override
    public void start(Stage stage) throws Exception {
        Parent root = FXMLLoader.load(getClass().getResource("view/PoMain.fxml"));
        Scene scene = new Scene(root, 600, 332);
        stage.initStyle(StageStyle.UNDECORATED);
        stage.setTitle("Oximeter Application");
        stage.setScene(scene);
        stage.show();
    }
}
  
```

Permet de lancer notre application, Stage.Style.UNDECORATED nous permet d'éliminer les boutons classiques de fermeture, réduction, agrandissement, qui évitera directement le cas suivant d'arriver :



Ils vont donc mieux programmer nos propre buttons, dont les bout de codes se répètent pour chaque fenêtre :

```
@FXML
private void CloseApp(MouseEvent event) {
    System.exit(0);
}
@FXML
private void MinApp(MouseEvent event) {
    stage.setIconified(true);
}
```

Bonus : Ce bout de code permet de bouger notre application en cliquant sur n'importe quel point de la scène.

```
private void makeStageDragable() {
    HealthReport.setOnMousePressed((event) -> {
        xOffset=event.getSceneX();
        yOffset=event.getSceneY();
    });
    HealthReport.setOnMouseDragged((event)-> {
        stage = (Stage) ((Node)event.getSource()).getScene().getWindow();
        stage.setX(event.getScreenX()-xOffset);
        stage.setY(event.getScreenY()-yOffset);
        stage.setOpacity(1f);
    });
    HealthReport.setOnDragDone((event)-> {
        stage.setOpacity(1.0f);
    });
    HealthReport.setOnMouseReleased((event)->{
        stage.setOpacity(1.0f);
    });
}
```

Par la suite, pour un passage de fenêtre à une autre(des deux sens) , classiquement, ils ressemblent tous à ce bout de code :

```
Parent signup = FXMLLoader.load(getClass().getResource("Signup.fxml"));
Scene SignPage = new Scene(signup);
Stage signstage = (Stage)((Node) event.getSource()).getScene().getWindow();
signstage.setScene(SignPage);
signstage.show();

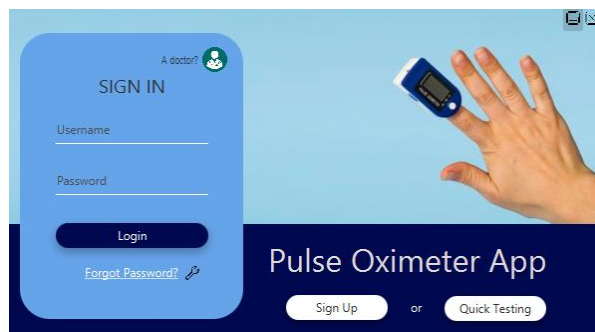
FXMLLoader loader = new FXMLLoader();
loader.setLocation(getClass().getResource("PatientList.fxml"));
Parent PatientList = loader.load();
Scene patientscene = new Scene(PatientList);
PatientListController plc = loader.getController();
plc.docname.setText(rs.getString("name"));
Stage patientstage = (Stage)((Node) event.getSource()).getScene().getWindow();
patientstage.setScene(patientscene);
patientstage.show();
```

Avec une petite différence, dans les cas où on a besoin de passer des informations d'une scène vers une autre, on modifie un petit peu le code :

En initialisant un loader, on peut maintenant utiliser la fonction `getController()` qui nous permettra d'utiliser les paramètres et fonctions d'un autre contrôleur.

Pour ce qui est de la partie SQL , je ne vais pas m'attarder sur le codage des requêtes (c'est du SQL classique), la partie intéressante est celle en JAVA qui est tout simplement la différence entre les requêtes d'affichage et les requêtes de modification : la première (SELECT par exemple) utilise généralement un `executeQuery()` puis une boucle qui permet de parcourir chaque ligne d'un tableau et les utiliser, alors que les modification(INSERT,UPDATE,ETC.) utilise plutôt un `executeUpdate()`.

De plus, un autre point intéressant à soulever, est ce bouton près du texte 'A doctor?' qui permet de changer l'interface du Login de celle du patient(bleue) à celle de docteur(vert), chaque interface est associée à une table de données différente, pour accéder à celle du docteur, cliquer sur le bouton.



La partie code sera jointe ultérieurement au rapport, et elle contient le reste du code car faire le point sur chaque fonctionnalité serait trop long et contre-productif.

Quick Testing is only for checking without follow up...
If you wish for doctor analysis, please go back and signup

[Measure Again](#)

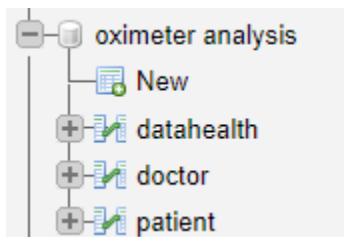
Time	User	Heart Rate (BPM)	Oxygen Saturation %
2022-01-31 01:04:11	Anonymous	115	100
2022-01-31 01:04:06	Anonymous	115	100
2022-01-31 01:04:01	Anonymous	115	100
2022-01-31	Anonymous	115	100

On va donc clôturer cette partie avec la fenêtre 'Quick Testing' ci-dessus qui permet d'afficher en temps réel les données reçues de notre détecteur vers notre base de données, et ce en utilisant du langage PHP.

b) Lecture des données(MYSQL)

La lecture de données se fait à partir de la base de données, pour récupérer celle utilisée, il suffit de créer une base de données vide, et d'importer le fichier oximeter_analysis.sql qui sera joint au projet, celui-ci va répliquer la base de données utilisée au cours de ce projet.

Notre base de données a été nommée : 'oximeter analysis', et elle comporte trois tableaux :



Le plus simple est le tableau 'doctor', qui agit ici comme fonction de 'admin', et donc par cette logique, seul un login et un mot de passe est nécessaire :

name	pass
Hatim	root1
Achraf	root2

Par la suite le tableau 'patient' :

	PatientID	Username	Password	Sex	DateOfBirth	Status
<input type="checkbox"/> Edit Copy Delete	0	Anonymous	NULL	NULL	NULL	healthy
<input type="checkbox"/> Edit Copy Delete	1	Jorja	itsjorja	FEMALE	27/05/1991	Healthy
<input type="checkbox"/> Edit Copy Delete	2	Mira	itsmira	FEMALE	22/09/2000	Unhealthy
<input type="checkbox"/> Edit Copy Delete	3	Rachid	itsrachid	MALE	09/08/1960	Healthy
<input type="checkbox"/> Edit Copy Delete	4	Patrick	itspatrick	MALE	06/01/1996	Healthy
<input type="checkbox"/> Edit Copy Delete	7	Achraf	itsachraf	MALE	27/02/2001	Healthy

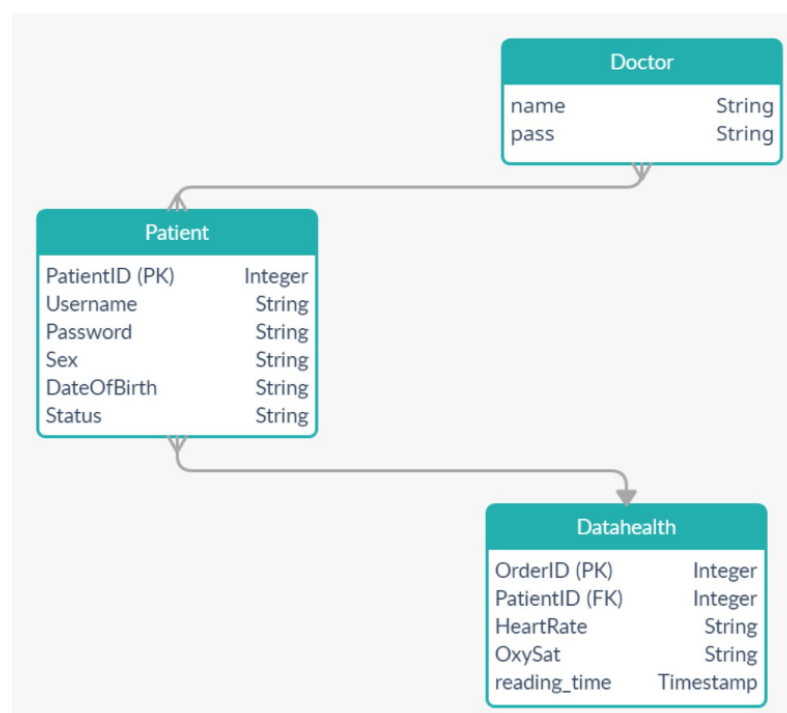
Ici le PatientID de valeur nulle(0) existe afin de pouvoir faire des requêtes de sélection qui nécessite une jointure, il sera uniquement référé quand l'utilisateur accèdera au 'Quick Testing'.

D'ailleurs, PatientID agit ici comme notre clé primaire, de type AUTO_INCREMENT, elle permet d'ignorer l'insertion dans cette colonne permettant ainsi l'ajout du reste des patients qui sont insérés au fur et à mesure que des nouveaux utilisateurs sont enregistrés(Dans l'interface 'Sign Up').

Finalement, la table 'datahealth, tout aussi simple, juste un tout petit peu différent, ici, PatientID est une clé étrangère qui fait référence à celle dans le tableau de 'patient' , notre clé primaire est une autre clé de type AUTO_INCREMENT, cette fois ci nous servant de colonne de tri afin de récupérer uniquement les dernières valeurs prises par le capteur.

				OrderID	PatientID	HeartRate	OxySat	reading_time
<input type="checkbox"/>	Edit	Copy	Delete	781	1	93	100	2022-01-31 01:09:58
<input type="checkbox"/>	Edit	Copy	Delete	780	1	93	100	2022-01-31 01:09:53
<input type="checkbox"/>	Edit	Copy	Delete	779	1	93	100	2022-01-31 01:09:48
<input type="checkbox"/>	Edit	Copy	Delete	778	1	93	100	2022-01-31 01:09:43
<input type="checkbox"/>	Edit	Copy	Delete	777	1	93	100	2022-01-31 01:09:38
<input type="checkbox"/>	Edit	Copy	Delete	776	1	93	100	2022-01-31 01:09:32
<input type="checkbox"/>	Edit	Copy	Delete	775	1	93	100	2022-01-31 01:09:27
<input type="checkbox"/>	Edit	Copy	Delete	774	1	93	100	2022-01-31 01:09:22

Pour résumer voici un diagramme de la base de données :



Ces tables suivent quelques règles prédéfinies :

- Un docteur peut avoir un ou plusieurs patients.
- Un patient peut avoir un ou plusieurs docteurs.
- Les docteurs agissent comme administrateurs dans le système.
- Le patient ne peut avoir qu'un seul dossier médical (Datahealth).
- Datahealth peut concerner plusieurs patients.

c) Envoi de données(C & PHP)

Notre application marche, notre base SQL est prête...mais où sont les données ? Pour ce faire, on divisera cette partie en deux étapes, la première du côté de l'Arduino IDE, où on récupérera les informations captées de notre MAX30105, qu'on enverra vers notre page PHP qui sera mise en charge d'insérer les données lues dans notre base de données.

Arduino ID(C)

```
#include <ESP8266WiFi.h>
#include <ESP8266HTTPClient.h>
#include <WiFiClient.h>
```

Ces trois bibliothèques contiennent les bouts de code nous permettant de nous connecter avec un autre appareil(adresse IP). On initialise par la suite nos identifiants réseau et serveur.

```
const char* ssid      = "Vesperiam";
const char* password = "Hatim@284";

const char* serverName = "http://192.168.43.87/SensorData/post-esp-data.php";

String apiKeyValue = "tPmAT5Ab3j7F9";
```

Note : l'apiKeyValue est là pour s'assurer qu'un n'importe quelle insertion de données a définitivement été confirmé.

Avec nos données entrées juste avant, on teste notre connexion :

```
WiFi.begin(ssid, password);
Serial.println("Connecting");
while(WiFi.status() != WL_CONNECTED) {
  delay(500);
  Serial.print(".");
}
Serial.println("");
Serial.print("Connected to WiFi network with IP Address: ");
Serial.println(WiFi.localIP());
```

Wifi.localIP() retourne notre adresse IP de connexion.

On démarre notre MAX30105 et on teste si tout démarre sans souci.

```
//read the first 100 samples, and determine the signal range
for (byte i = 0 ; i < bufferLength ; i++)
{
  while (particleSensor.available() == false) //do we have new data?
    particleSensor.check(); //Check the sensor for new data

  redBuffer[i] = particleSensor.getRed();
  irBuffer[i] = particleSensor.getIR();
  particleSensor.nextSample(); //We're finished with this sample so move to next sample

  Serial.print(F("red="));
  Serial.print(redBuffer[i], DEC);
  Serial.print(F(", ir="));
  Serial.println(irBuffer[i], DEC);
}

// Initialize sensor
if (!particleSensor.begin(Wire, I2C_SPEED_FAST)) //Use default I2C port, 400kHz speed
{
  Serial.println(F("MAX30105 was not found. Please check wiring/power."));
  while (1);
}
Serial.println(F("Attach sensor to finger with rubber band. Press any key to start conversion"));
while (Serial.available() == 0) ; //wait until user presses a key
Serial.read();
```

On prend un échantillon de rayons pour déterminer les intervalles de fonctionnement de notre capteur (on teste simplement que tout fonctionne correctement).

```
//calculate heart rate and SpO2 after first 100 samples (first 4 seconds of samples)
maxim_heart_rate_and_oxygen_saturation(irBuffer, bufferLength, redBuffer, &spo2, &validSPO2, &heartRate, &validHeartRate);
```

La librairie de "spo2_algorithm.h" nous permet d'utiliser une fonction prédéfinie qui suit les lois et formules vu dans la partie théorique.

Après avoir fini la détection, on doit envoyer les valeurs, pour ce faire, on fait appel à la librairie importée au début <ESP8266HTTPClient.h> . Celle-ci nous permettra de faire une requête post à notre fichier post-esp-data.php qui par la suite fera une insertion dans la base de données.

Tout d'abord, on met en place quelques conditions :

```
if(validHeartRate==1 && validSPO2==1 && heartRate>39 && heartRate <121 && spo2 >80 && spo2 <101 )
```

Après tout, on ne veut pas prendre en compte des valeurs négatives ni de valeurs incohérentes.

```
//Check WiFi connection status
if(WiFi.status()== WL_CONNECTED){
    WiFiClient client;
    HTTPClient http;

    // Your Domain name with URL path or IP address with path
    http.begin(client,serverName);

    // Specify content-type header
    http.addHeader("Content-Type", "application/x-www-form-urlencoded");

    // Prepare your HTTP POST request data
    String httpRequestData = "api_key=" + apiKeyValue + "&Oxysat=" + String(spo2)+ "&HeartRate=" + String(heartRate)+" ";
    Serial.print("httpRequestData: ");
    Serial.println(httpRequestData);
    //String httpRequestData = "api_key=tPmAT5Ab3j7F9&Oxysat=60&HeartRate=99";
    // Send HTTP POST request
    int httpResponseCode = http.POST(httpRequestData);
    Serial.print("httpRequestData: ");
    Serial.println(httpRequestData);
```

httpRequestData est la requête qui sera reçue et traitée côté PHP.

Et à la suite de l'envoi de ce code, le httpResponseCode nous aidera à déterminer si l'opération a bien été exécutée.

```

if (httpResponseCode >0) {
    Serial.print("HTTP Response code:");
    Serial.println(httpResponseCode);
}
else {
    Serial.print("Error code: ");
    Serial.println(httpResponseCode);
}

```

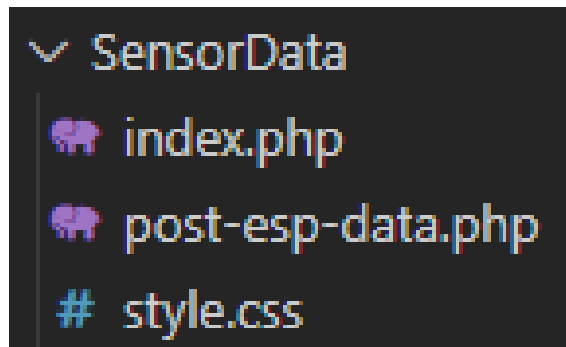
httpResponseCode = -1 : Il y'a un souci de connexion, il faut vérifier que l'appareil est bien connecté(idéalement dans un réseau privé).

httpResponseCode = 403 : Forbidden, L'accès au serveur a été refusé, il faut vérifier que rien ne bloque l'accès des adresses IP provenant de l'extérieur.



httpResponseCode = 200 : OK , La requête POST a bien été exécutée.



Il y'a évidemment d'autre réponses possibles mais ces trois ont été les plus communes durant notre projet.

SensorData (PHP)




Dans cette partie, notre index.php nous affiche la page suivante (vu dans le 'Quick Testing'), et ce en faisant une simple requête de connexion, sélection et d'affichage :

 Quick Testing is only for checking without follow up...
If you wish for doctor analysis, please go back and signup 


  Measure Again


Time	User	Heart Rate (BPM)	Oxygen Saturation %
2022-01-31 01:04:11	Anonymous	115	100
2022-01-31 01:04:06	Anonymous	115	100
2022-01-31 01:04:01	Anonymous	115	100
2022-01-31	Anonymous	115	100

Welcome, Jorja 

Your latest test showed, you are therefore : **Healthy**

HeartRate	Oxygen Saturation
93	100

 Display Results

 Measure Again

Time	User	Heart Rate (BPM)	Oxygen Saturation %
2022-01-31 01:09:58	Jorja	93	100
2022-01-31 01:09:53	Jorja	93	100

Notre application arrive à reconnaître l'utilisateur connecté et affiche les données pertinentes, pourquoi ? Ceci est grâce à une requête GET qu'on a mis du côté JAVA et PHP.

```
qtc.engine.load("http://" + this.ip_address + "/SensorData/?pid="+0);
```



```
hrc.engine.load("http://" + this.ip_address + "/SensorData/?pid=" + rs.getInt("PatientID"));
```

```
if ($_SERVER["REQUEST_METHOD"] == "GET") {
    $pid = $_GET['pid'];
    $Patient_ID = (int)$pid;
```

Et puis cette requête pour afficher le tableau :

```
$sql = "SELECT Username,HeartRate,OxySat, reading_time FROM datahealth, patient
WHERE datahealth.PatientID=patient.PatientID AND datahealth.PatientID =' " .
$Patient_ID . "' ORDER BY OrderID DESC ";
```

```
echo '<table cellpadding="5" cellspacing="5">
<tr>
    <th>Time</th>
    <th>User</th>
    <th>Heart Rate (BPM)</th>
    <th>Oxygen Saturation &#37;</th>
</tr>';

if ($result = $conn->query($sql)) {
    while ($row = $result->fetch_assoc()) {
        $row_username = $row["Username"];
        $row_reading_time = $row["reading_time"];
        $row_HeartRate = $row["HeartRate"];
        $row_OxySat = $row["OxySat"];

        echo '<tr>
            <td>' . $row_reading_time . '</td>
            <td>' . $row_username . '</td>
            <td>' . $row_HeartRate . '</td>
            <td>' . $row_OxySat . '</td>
        </tr>';
    }
    $result->free();
}

$conn->close();
```

Le post-esp-data.php quant à lui, est plus important, car il permet de recevoir les données de notre Arduino IDE, et les insérer dans notre bases de données.

- Connexion base de données :

```
$servername = "localhost";
$username = "root";
$password = "";
$dbname = "oximeter analysis";
$conn = mysqli_connect($servername, $username, $password, $dbname);
```

- Initialisation des paramètres et de la clé de sécurité :

```
$api_key_value = "tPmAT5Ab3j7F9";
$api_key= $Oxysat = $HeartRate = "";
```

- Fonction test_input (enlever les caractères spéciaux, etc.) :

```
function test_input($data) {
    $data = trim($data);
    $data = stripslashes($data);
    $data = htmlspecialchars($data);
    return $data;
}
```

- Récupérer les valeurs de l'ESP8266 :

```
if ($_SERVER["REQUEST_METHOD"] == "POST") {
    $api_key = test_input($_POST["api_key"]);
    if($api_key==$api_key_value) {
        $Oxysat = test_input($_POST["Oxysat"]);
        $HeartRate = test_input($_POST["HeartRate"]);
        $conn = new mysqli($servername, $username, $password, $dbname);
        if ($conn->connect_error) {
            die("Connection failed: " . $conn->connect_error);
        }
    }
}
```

- Requête sql (on change l'ID manuellement selon l'utilisateur) :

```
$sql = "INSERT INTO datahealth(PatientID,HeartRate,OxySat) VALUES('" . 1 .
",'" . $HeartRate . "','" . $Oxysat . "')";
```

- Les autres cas :

```
if ($conn->query($sql) === TRUE) {
    echo "New record created successfully";
}
else {
    echo "Error: " . $sql . "<br>" . $conn->error;
}

$conn->close();
}
else {
    echo "Wrong API Key provided.";
}
}
else {
    echo "No data posted with HTTP POST.";
}
```

Le style.css, tout comme dans notre application JAVA, sert simplement à décorer et modifier le style de notre page HTML.

Conclusion

Le travail sur ce projet a été pour nous très enrichissant en plusieurs domaines de connaissances. Ce projet a servi comme une initiation vers la mise en place de circuits électroniques à base de carte microcontrôleur. Nous avons appris à monter les circuits électroniques relativement complexes, tout en développant des programmes exécutés par les microcontrôleurs. On a également pu fortifier nos connaissances en C,PHP,JAVA et SQL. Nous avons eu l'opportunité d'appliquer nos connaissances antérieures de programmation orientée objet avec Java pour développer l'application qui traite les données reçues de la part de l'oxymètre. Et non seulement Orienté Objet, mais les cours étudiés cette année en parallèle de HTML, et Linux embarqué, ont cultivé des compétences qui se sont montrés pertinentes durant ce projet.

Nous ne sommes bloqués plusieurs fois lors de notre avancement sur le projet, en rencontrant toutes sortes d'erreurs à la fois au niveau du code Arduino, Java et l'établissement de la connexion via Internet entre notre oxymètre et notre application(PHP). C'est là où notre esprit de résolution de problèmes et de « Troubleshooting » a fait preuve d'utilité. Avec de la recherche en ligne, relecture des codes et révision des méthodes adoptées, on a pu surmonter la majorité des obstacles rencontrés.

Bibliographie

<https://create.arduino.cc/projecthub/SurtrTech/measure-heart-rate-and-spo2-with-max30102-c2b4d8> (source initiale)

<https://how2electronics.com/max30100-pulse-oximeter-with-esp8266/>

https://electronicsinnovation.com/solved-max30100-not-working-initializing-pulse-oximeter-failed/#Circuit_Connection_with_ESP32_ESP8266

<https://how2electronics.com/interfacing-max30100-pulse-oximeter-sensor-arduino/>

<https://www.circuitschools.com/diy-pulse-oximeter-by-interfacing-max30100-sensor-with-arduino/>

<https://morf.lv/implementing-pulse-oximeter-using-max30100>

<https://www.youtube.com/watch?v=itmGK3rBtF8>

https://www.electronicclinic.com/max30100-pulse-oximeter-arduino-code-circuit-and-programming/#Arduino_program_to_Display_the_Oxygen_percentage_and_BPM_on_the_LCD

<https://www.youtube.com/watch?v=5WI2lXr9CVE>

<https://www.youtube.com/watch?v=4RNhPZJ84P0> (Comment utiliser setCellValueFactory() proprement)

<https://www.youtube.com/watch?v=LoiQVoNil9Q> (Table.css provient d'ici, avec un peu de modifications selon nos besoins)

<https://github.com/oxullo/Arduino-MAX30100/issues/13> (Diagnosticuer MAX30100+Arduino IDE qui ne marche pas)

<https://github.com/oxullo/Arduino-MAX30100/issues/23> (Diagnosticuer MAX30100+Arduino IDE qui ne marche pas) Solution trouvé, utiliser la librairie suivante :

https://github.com/sparkfun/SparkFun_MAX3010x_Sensor_Library

<https://www.medisafe.fr/blog/comment-fonctionne-un-oxymetre-de-pouls/> pour des définitions (et l'intervalle pour les battements de cœur et la saturation d'O2)

https://mytectutor.com/how-to-connect-esp8266-to-mysql-database-using-php-and-arduino-ide/#Code_for_ESP8266_Nodemcu_with_BME_280_sensor

<https://www.youtube.com/watch?v=vxd-gu9nXpY> pour signer mon projet et utiliser le pulseoximeter.jar

<https://netbeans.apache.org/download/index.html> IDE Java utilisé

<https://code.visualstudio.com/download> IDE PHP utilisé

<https://sourceforge.net/projects/wampserver/> PHPMYADMIN + Localhost

<https://www.arduino.cc/en/software> Arduino IDE

<http://www.jfoenix.com/documentation.html> (Librairie JFoenix)

<https://dev.mysql.com/downloads/connector/j/> (Librairie JDBC connector)

<https://www.youtube.com/watch?v=6hpljx8d15s>

<https://www.youtube.com/watch?v=y3dqJrcKqEo&t=70s> Câblage correct pour notre Feather Huzzah + MAX30105