

CHU
Patrick
COLSON
Eliott

RAPPORT DE PROJET

Tube de Kundt

Lycée Dorian
2022-2023

SOMMAIRE

INTRODUCTION

REPARTITION DES TÂCHES

BASE DE DONNEES

ETUDE UML

LE TUBE DE KUNDT

PARTIE PERSONELLE ETUDIANT 1

CREATION DE L'IHM

MISE EN PLACE DU BANC DE MESURE

MESURES MANUELLES

CODER LES MODULES DE PILOTAGE DES APPAREILS DU BANC
DE MESURE

MODULE D’AFFICHAGE DE LA MESURE

CREER LA PAGE WEB

REGROUPER TOUT LES MODULES

PARTIE PERSONELLE ETUDIANT 2

INTRODUCTION

Les particuliers, les professionnels de la musique ou les industriels utilisent parfois des matériaux acoustiques pour leur confort, voire leur santé. Pour se les procurer et trouver le matériau qui leur conviendra le plus, ils se tournent vers des fabricants, des distributeurs ou des vendeurs de matériaux acoustiques.

Un fabricant et distributeur de matériaux acoustiques veut pouvoir mesurer le coefficient d'absorption d'énergie acoustique des différents matériaux qu'il propose à la vente.

Pour pouvoir répondre à ses attentes nous allons utiliser un tube de Kundt afin de pouvoir relever les coefficients d'absorption d'énergie acoustique de ses matériaux en fonction de différentes fréquences, normalisées à tous les tiers d'octave entre 200Hz et 3150Hz.

L'utilisation de ce tube soulève un problème, il n'est pas simple à manipuler pour une personne qui n'est pas qualifiée. Il faut donc trouver un moyen pour rendre la manipulation du tube accessible par des personnes non qualifiées.

Nous avons choisi d'automatiser le tube afin de le rendre pilotable facilement à l'aide d'une application qui communiquera avec les différents appareils présents sur le banc de mesure.

Le tube sera piloté à l'aide d'un moteur pas-à-pas hybride que l'on contrôlera à l'aide d'un driver de moteur pas-à-pas A4988 de la marque Polulu et d'une carte Arduino Mega 2560. Ce moteur sera fourni en énergie par une alimentation elc AL 991s.

Nous récupérerons les mesures grâce à un oscilloscope Rigol MSO5074 qui servira également de générateur basses fréquences pour le haut-parleur du tube.

Répartition des tâches

Étudiant 1 : Patrick CHU

- Concevoir l'IHM
- Mettre en place le banc de mesure
- Réaliser des tests manuels
- Coder le module de pilotage des appareils du banc de mesure
- Coder un module d'affichage de la mesure
- Définition et création de la base de données pour archiver les mesures et les autres paramètres
- Création d'une page Web de consultation de matériaux et de ses caractéristiques
- Regrouper tous les modules dans l'IHM

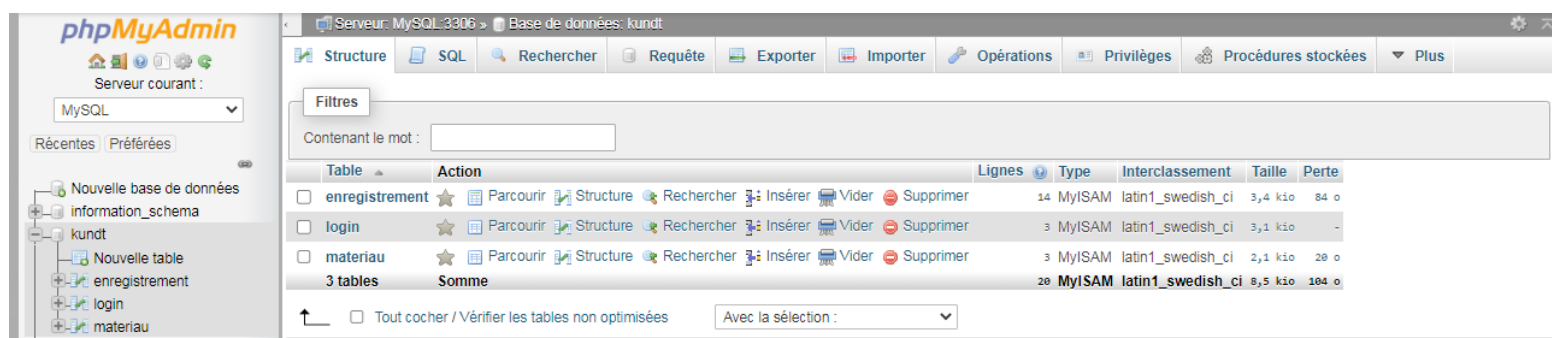
Étudiant 2 : Eliott COLSON

- Installer le système de déplacement
- Coder le module de pilotage du moteur par l'Arduino
- Coder le module d'affichage des informations importantes par l'Arduino
- Coder une classe de communication avec l'IHM
- Mise en place du serveur WEB et de la base de données
- Définition et création de la base de données pour archiver les mesures et les autres paramètres
- Coder un module logiciel pour lire et écrire dans la base de données les informations pertinentes
- Coder un module pour tracer la courbe du coefficient d'absorption du matériau testé

BASE DE DONNÉES

Nous avons d'abord pensé à une base de données dans laquelle une nouvelle table est créée , portant le nom de chaque matériau pour lequel on aurait mesuré au moins un coefficient d'absorption. Jugée trop compliquée nous sommes donc partis sur une autre structure de base de données.

Structure de la base de données :



The screenshot shows the phpMyAdmin interface for a MySQL database named 'kundt'. The left sidebar shows the database structure with three tables: 'enregistrement', 'login', and 'matériau'. The main panel displays the 'Structure' tab for the 'enregistrement' table. The table has the following columns: 'id_enregistrement' (PK), 'id_materiel' (FK), 'frequence', and 'absorption'. The table is MyISAM, latin1_swedish_ci, and has 14 rows. The 'matériau' table has 3 rows, and the 'login' table has 3 rows. The 'Somme' row shows a total of 20 rows across all tables.

Table	Action	Lignes	Type	Interclassement	Taille	Perte
<input type="checkbox"/> enregistrement	Parcourir Structure Rechercher Insérer Vider Supprimer	14	MyISAM	latin1_swedish_ci	3,4 kio	84 o
<input type="checkbox"/> login	Parcourir Structure Rechercher Insérer Vider Supprimer	3	MyISAM	latin1_swedish_ci	3,1 kio	-
<input type="checkbox"/> matériau	Parcourir Structure Rechercher Insérer Vider Supprimer	3	MyISAM	latin1_swedish_ci	2,1 kio	20 o
3 tables	Somme	20	MyISAM	latin1_swedish_ci	8,5 kio	104 o

La base de données comporte trois tables :

- *matériau* : qui contient le nom des matériaux déjà testés et pour lesquels un *id_materiel* leur sera lié ;
- *enregistrement* : qui répertorie les fréquences et leur coefficient d'absorption respectif de chaque matériau. Cette table sera liée à la table *matériau* par la clé étrangère *id_materiel* ;
- *login* : qui permet de stocker les noms des utilisateurs et leur mot de passe respectif ;



Structure de la table « materiau » :

Serveur: MySQL:3306 » Base de données: kundt » Table: materiau										
Parcourir Structure SQL Rechercher Insérer Exporter Importer Privileges Opérations Déclencheurs										
#	Nom	Type	Interclassement	Attributs	Null	Valeur par défaut	Commentaires	Extra	Action	
<input type="checkbox"/> 1	id_materiau	int(11)			Non	Aucun(e)		AUTO_INCREMENT	Modifier	Supprimer Plus
<input type="checkbox"/> 2	nom	text	latin1_swedish_ci		Non	Aucun(e)			Modifier	Supprimer Plus

Structure de la table « enregistrement » :

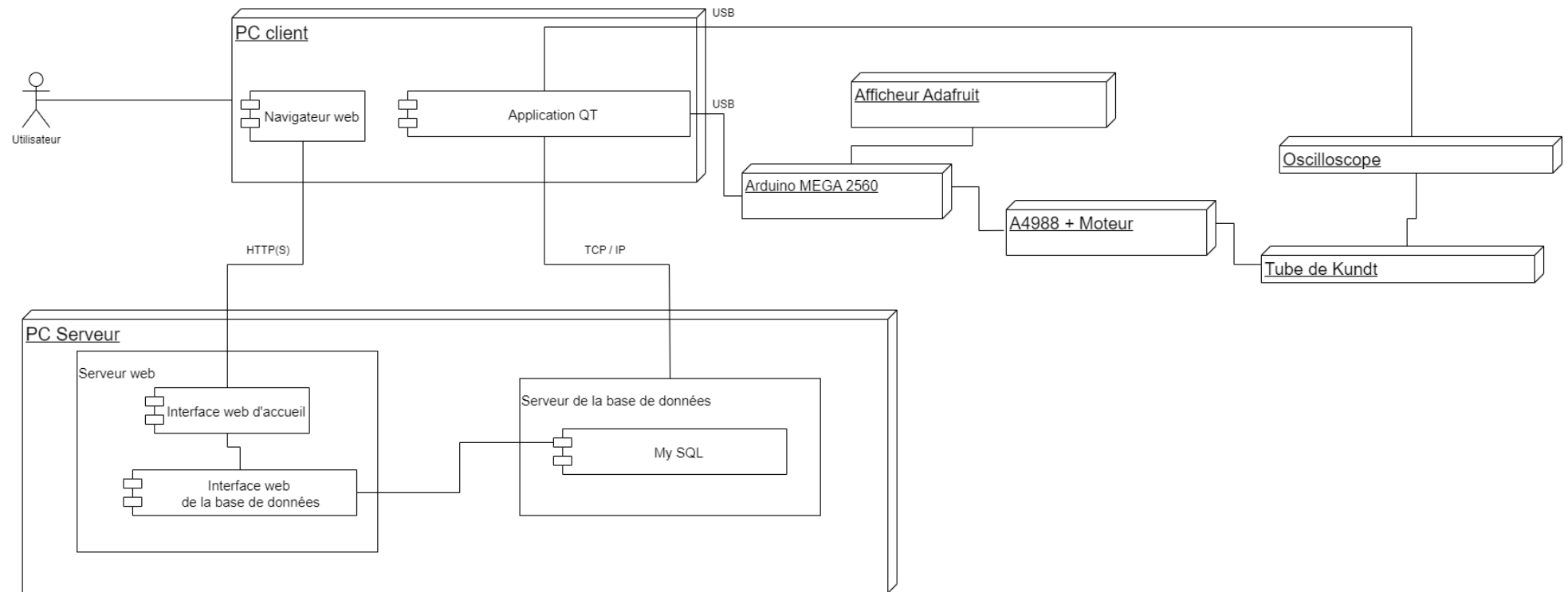
Serveur: MySQL:3306 » Base de données: kundt » Table: enregistrement										
Parcourir Structure SQL Rechercher Insérer Exporter Importer Privileges Opérations Déclencheurs										
#	Nom	Type	Interclassement	Attributs	Null	Valeur par défaut	Commentaires	Extra	Action	
<input type="checkbox"/> 1	id_enregistrement	int(11)			Non	Aucun(e)		AUTO_INCREMENT	Modifier	Supprimer Plus
<input type="checkbox"/> 2	frequence	int(11)			Oui	NULL			Modifier	Supprimer Plus
<input type="checkbox"/> 3	absorption	double			Oui	NULL			Modifier	Supprimer Plus
<input type="checkbox"/> 4	id_materiau	int(11)			Oui	NULL			Modifier	Supprimer Plus

Structure de la table « login » :

Serveur: MySQL:3306 » Base de données: kundt » Table: login										
Parcourir Structure SQL Rechercher Insérer Exporter Importer Privileges Opérations Déclencheurs										
#	Nom	Type	Interclassement	Attributs	Null	Valeur par défaut	Commentaires	Extra	Action	
<input type="checkbox"/> 1	id_login	int(11)			Non	Aucun(e)		AUTO_INCREMENT	Modifier	Supprimer Plus
<input type="checkbox"/> 2	user	varchar(30)	latin1_swedish_ci		Non	Aucun(e)			Modifier	Supprimer Plus
<input type="checkbox"/> 3	password	varchar(30)	latin1_swedish_ci		Non	Aucun(e)			Modifier	Supprimer Plus

Etude UML

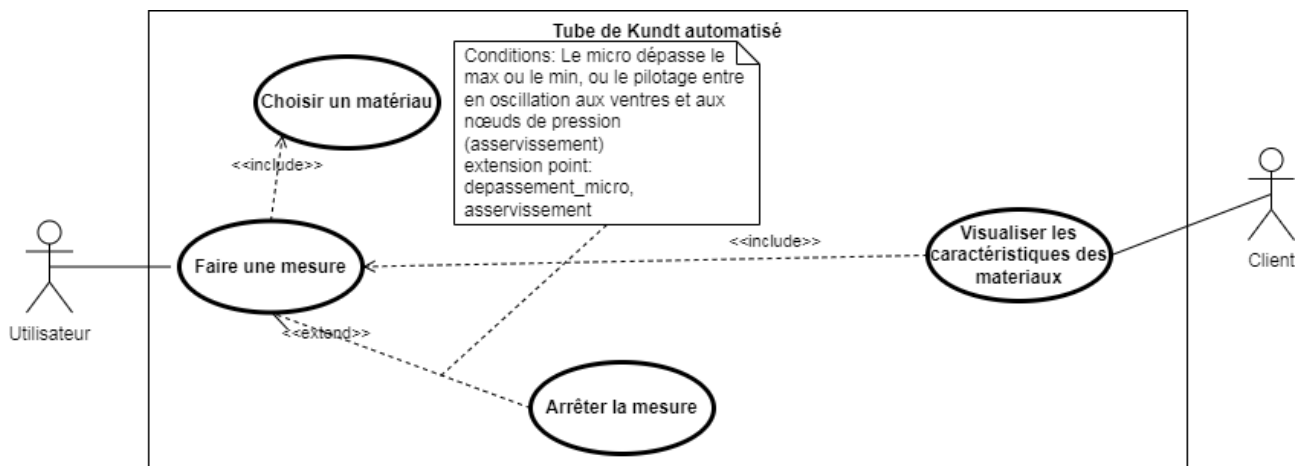
1. Déploiement :



L'utilisateur aura un PC sur lequel se trouvera l'application et un navigateur web.

L'application devra contrôler l'oscilloscope et communiquer avec l'Arduino pour pouvoir contrôler le tube. Les résultats obtenus via l'application seront ensuite enregistrés sur la base de données. Les données seront ensuite visibles sur un site web et sur l'afficheur connecté à l'Arduino.

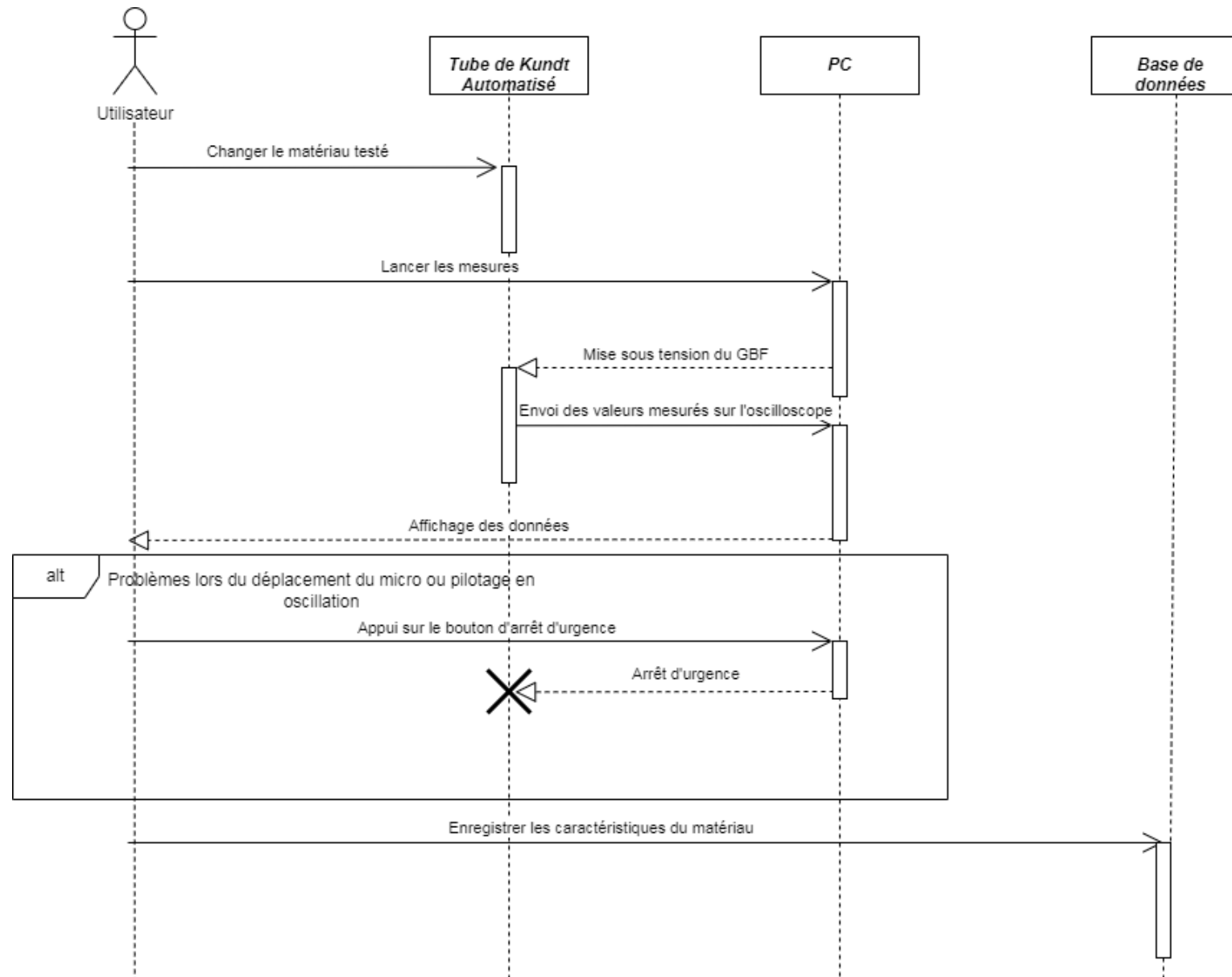
2. Cas d'utilisation :



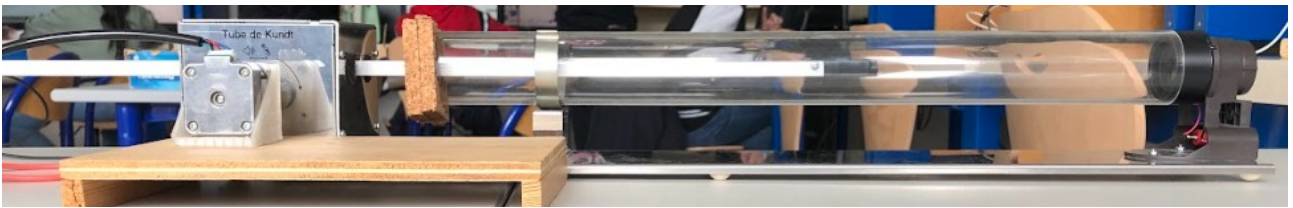
L'utilisateur devra choisir un matériau pour pouvoir faire une mesure. Il pourra aussi arrêter la mesure en cas de problèmes (si le micro bute contre une extrémité ou si le pilotage rentre en oscillation aux ventres ou aux nœuds de pression).

Le client, quant à lui, pourra visualiser les caractéristiques des matériaux qui auront été mesurées puis enregistrées dans la base de données.

3. Séquence :

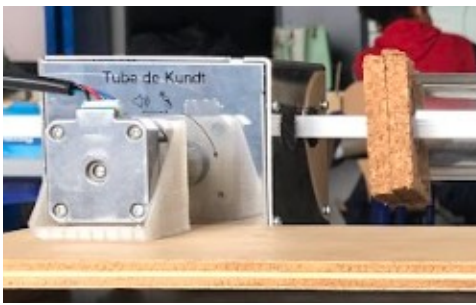


LE TUBE DE KUNDT

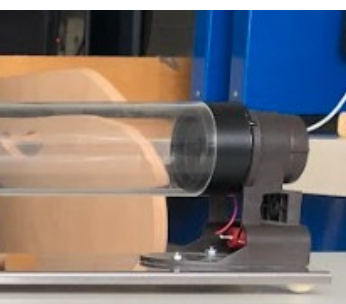


Le tube de Kundt, inventé en 1866 par August Kundt, est un dispositif expérimental qui permet de mettre en évidence les ondes stationnaires sonores dans un tube rempli d'air. Il permet également la mesure du coefficient d'absorption d'énergie acoustique de différents matériaux.

A) Principe de fonctionnement

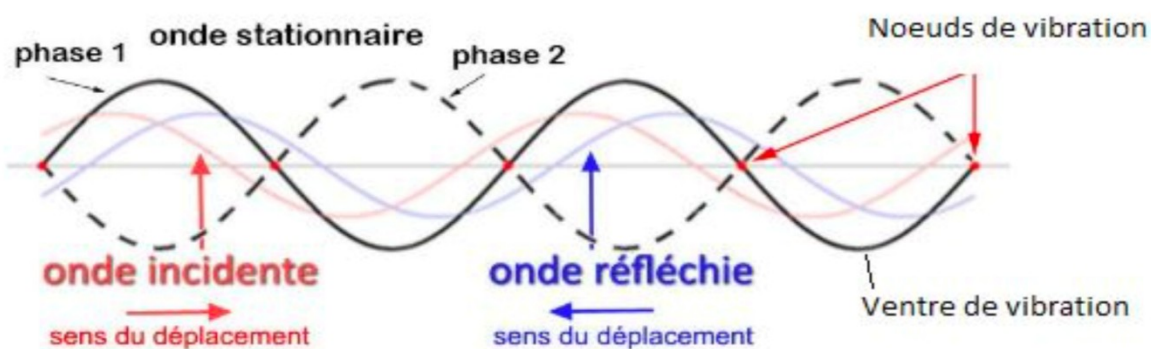


Dans un premier temps le tube est fermé à l'extrémité par le matériau que l'on veut tester.



Ensuite, le haut-parleur qui est situé à l'autre bout est alimenté par un générateur basses fréquences. Le haut-parleur émet ainsi une onde sonore qui est réfléchiée par le matériau placé à l'autre bout du tube.

Les ondes qui voyagent alors dans des directions opposées, mais avec la même fréquence, se superposent composant alors une onde stationnaire.



Cette onde est captée par un micro placé sur une tige mobile et visualisée sur un oscilloscope en sortie.

B) Mesure du coefficient d'absorption

La mesure se fait en relevant les ventres et les nœuds de pression de l'onde créée dans le tube.

Les ventres de pressions sont ainsi relevés aux points où l'amplitude de l'onde stationnaire est maximale tandis que les nœuds de pressions sont mesurés aux points où l'amplitude de l'onde est minimale.

Le coefficient d'absorption est ensuite calculé avec la formule suivante :

$$\alpha = 1 - \left(\frac{n-1}{n+1} \right)^2$$

Calcul de n :

$$n = \frac{P_{max}}{P_{min}}$$

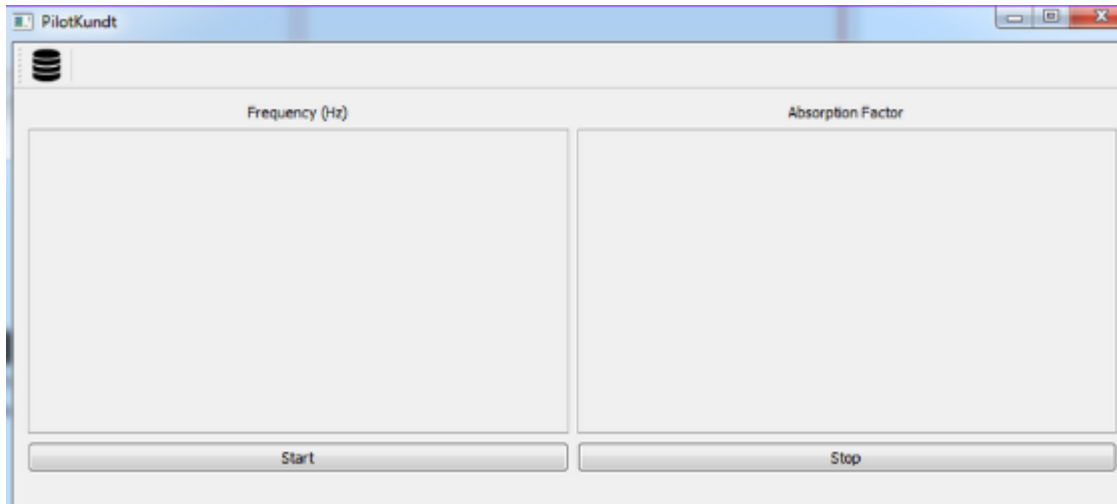
P_{max} est mesuré aux ventres de pressions.

P_{min} est mesuré aux nœuds de pressions.

PARTIE PERSONELLE : PATRICK (ETUDIANT 1)

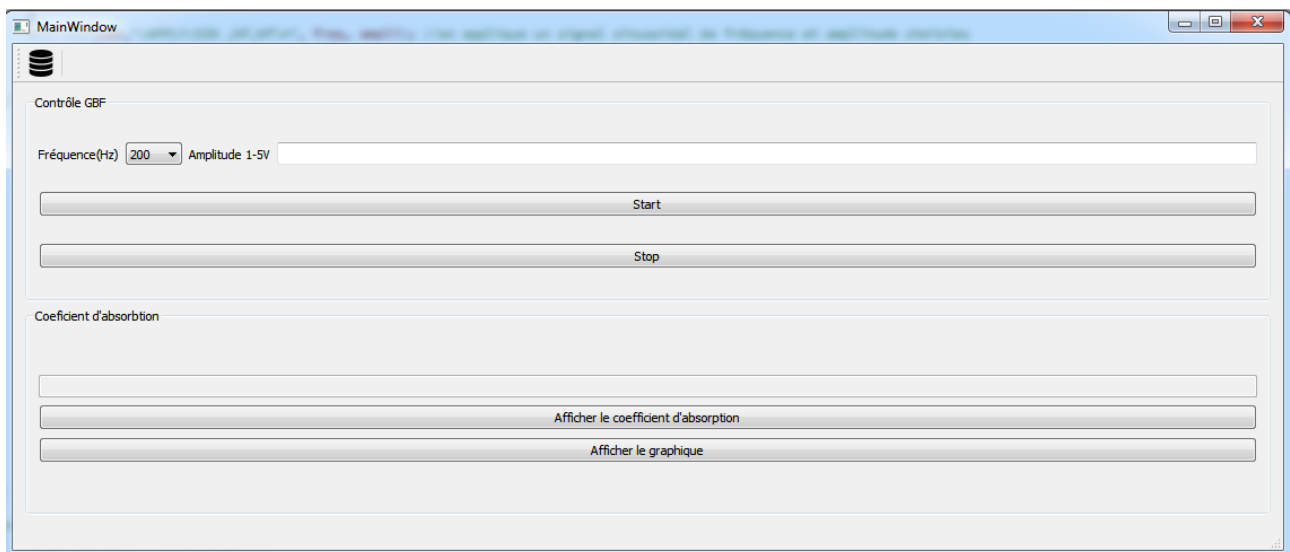
CRÉATION DE L'IHM

Première version de l'IHM :

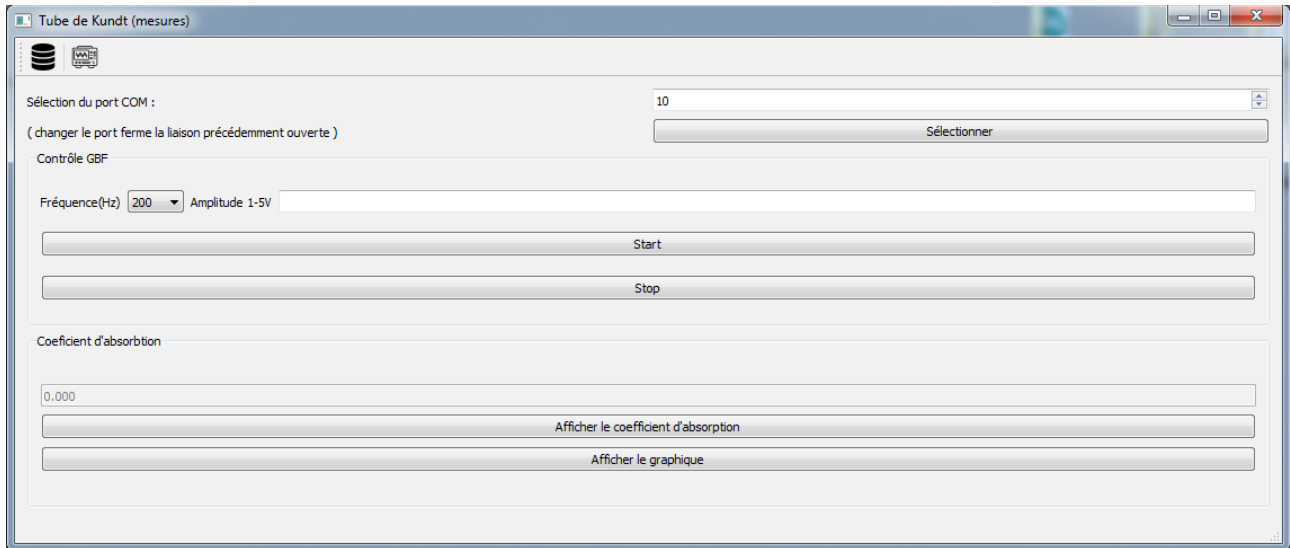


Cette première version très simple permettait de visualiser la fréquence du signal émis et le coefficient d'absorption mesuré, de lancer la mesure, d'arrêter la mesure et d'accéder à la base de données.

Cette version de l'IHM ne répondant plus à nos attentes nous avons dû la modifier pour qu'elle puisse être adaptée à différentes situations. Elle s'est ainsi composée de deux parties, une partie pour contrôler le GBF de l'oscilloscope et une autre pour visualiser les résultats obtenus.



Cette IHM nous paraissait complète avant de nous rendre compte qu'on pouvait la rendre encore plus flexible, notamment en permettant de choisir le port COM sur lequel l'Arduino est connecté ainsi qu'en ajoutant un bouton permettant de se reconnecter à l'oscilloscope en cas de problème. Nous nous retrouvons ainsi avec l'IHM suivante sur laquelle nous avons toutes les options nécessaires à la complétion d'une mesure.



MISE EN PLACE DU BANC DE MESURE



Pour réaliser les mesures manuelles nous avons utilisé un oscilloscope Rigol MSO5074.

La mesure se fait en reliant les deux sorties « signal brut » et « enveloppe » aux entrées de l'oscilloscope.

Il ne reste plus qu'à régler la fréquence sonore du haut-parleur à l'aide d'un GBF, nous avons dans un premier temps utilisé un GBF Rigol DG1022 que nous n'utiliserons plus dans le projet car l'oscilloscope a une fonction GBF intégrée.



GBF Rigol DG1022

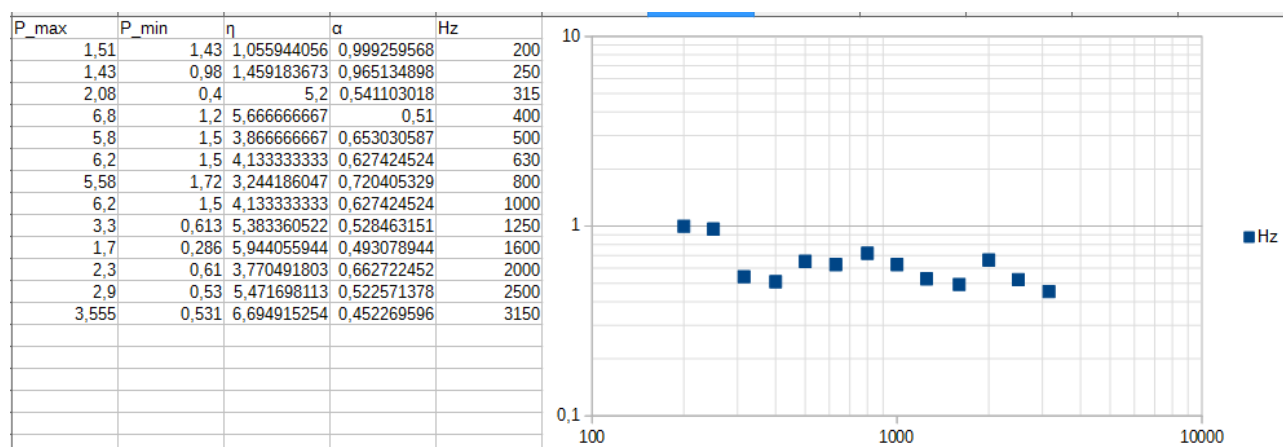


Oscilloscope MSO 5074

Nous relierons la sortie signal brut à la voie 1 de l'oscilloscope, la sortie enveloppe à la voie 2 et la sortie position du micro à la voie 3 de l'oscilloscope.

MESURES MANUELLES

Une fois le banc de mesure mis en place, nous pouvons commencer nos mesures. Seul le liège a pu être testé car ces dernières étaient très longues à faire.



Nous relevons alors ces mesures pour les 13 fréquences différentes.

Le choix du matériau a été mauvais car le liège est l'un des seuls matériaux ayant des coefficients différents à cause du type de grain utilisé.

Néanmoins, les mesures effectuées avec le liège qui a été mis à notre disposition sont proches des valeurs que l'on peut retrouver sur internet.

Ces mesures nous ont fait nous rendre compte à quel point le relevé de la courbe d'absorption d'un matériau était longue, mais aussi que cette mesure pouvait être très facilement perturbée lors d'un relevé. En effet, le tube est très sensible aux bruits extérieurs. C'est une information que nous devons prendre en compte lors de l'automatisation du tube car le moteur génère des bruits parasites lors du déplacement de la tige du tube.

CODER LES MODULES DE PILOTAGE DES APPAREILS DE MESURE

L'utilisation de la librairie ni-visa nous permet de contrôler les instruments Ethernet/LXI, GPIB, série, USB, PXI et VXI dans des environnements de développement.



Nous avons eu accès à cette librairie en téléchargeant l'application d'*Agilent Technologies* qui nous propose une interface simple qui utilise des commandes en IEEE 488.

La librairie Visa nous offre différentes fonctions qui nous permettent de communiquer facilement avec l'appareil.

Pour piloter le GBF on utilisera :

- - `viFindRsrc()` pour trouver l'instrument ;
- - `viOpen` pour ouvrir la connexion avec l'instrument ;
- - `viPrintf()` pour envoyer une chaîne de caractères à l'instrument ;
- - `viScanf()` pour récupérer la chaîne de caractères renvoyée par l'instrument ;
- - `viClose()` pour fermer la connexion avec l'instrument ;

L'oscilloscope sera piloté grâce aux commandes suivantes :

Pour toutes les commandes, les caractères en minuscules sont facultatifs. La commande fonctionne en ne mettant que les caractères en majuscule.

- `:SOURce[<n>]:APPLY :SINusoid[<freq>|DEFault|MINimum|MAXimum[,<amp>|DEFault|MINimum|MAXimum[,<offset>|DEFault|MINimum|MAXimum[,<phase>|DEFault|MINimum|MAXimum]]]]`

Name	Type	Range	Default
[<n>]	Discrete	1 2	1
<freq>	Real	1uHz to 60MHz	1kHz
<amp>	Real	Refer to the "Explanation"	5Vpp
<offset>	Real	Refer to the "Explanation"	0V _{DC}
<phase>	Real	0° to 360°	0°

Cette commande permet d'appliquer un signal sinusoïdal au GBF spécifié.

- :OUTPut[<n>][:STATe] {ON|1|OFF|0}

Name	Type	Range	Default
[<n>]	Discrete	1 2	1
{ON 1 OFF 0}	Bool	ON 1 OFF 0	OFF

Cette commande permet de couper le GBF spécifié.

- :CHANnel<n> :SCALE <scale>

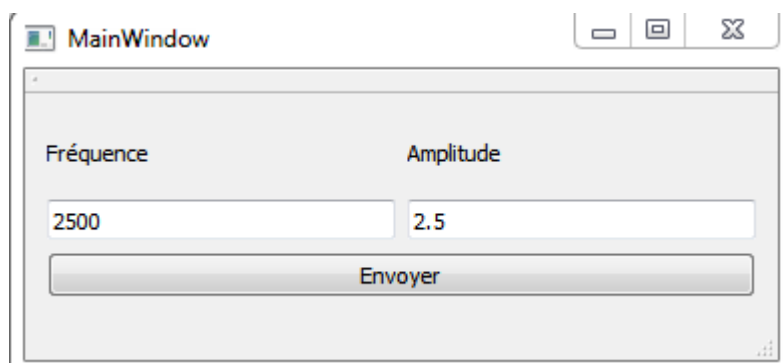
Name	Type	Range	Default
<n>	Discrete	{1 2 3 4}	1
<scale>	Real	The probe ratio is 1X: 500µV to 10 V	100 mV

Cette commande permet d'ajuster l'échelle d'une voie.

- :AUToscale

Cette commande permet de faire un autoset de l'oscilloscope.

En ayant pris connaissance de ces commandes, j'ai ainsi pu créer un module pour piloter le GBF de l'oscilloscope.



Nous pouvons ainsi récupérer la fréquence et l'amplitude que nous voulons appliquer.

Pour faire cela il faut tout simplement récupérer les valeurs indiquées dans les textEdit qu'on convertis en double pour ensuite les envoyer vers l'oscilloscope.

```
freq = ui->freqEdit->text().toDouble();  
ampli = ui->amplEdit->text().toDouble();  
viPrintf(osc,":APPLY:SIN ,%f,%f\n", freq, ampli); //on applique un signal sinusoidal de fréquence et amplitude choisies
```

MODULE D’AFFICHAGE DE LA MESURE

Pour récupérer les valeurs visibles sur l’oscilloscope, on utilise les commandes suivantes :

- MEASure :ITEM <item>[,<src>[,<src>]]

Name	Type	Range	Default
<item>	Discrete	{VMAX VMIN VPP VTOP VBASE VAMP VAVG VRMS OVERshoot PRESshoot MAREa MPAREa PERiod FREQuency RTIME FTIME PWIDTH NWIDTH PDUTy NDUTy TVMAX TVMIN PSLe wrate NSLewrate VUPPer VMID VLOWer VAR iance PVRMs PPULses NPULses PEDGes NED Ges RRDelay RFDelay FRDelay FFDelay RRP Hase RFPHase FRPHase FFPHase}	—
<src>	Discrete	Refer to Remarks	—

<src> sera remplacé par CHANnel <n>

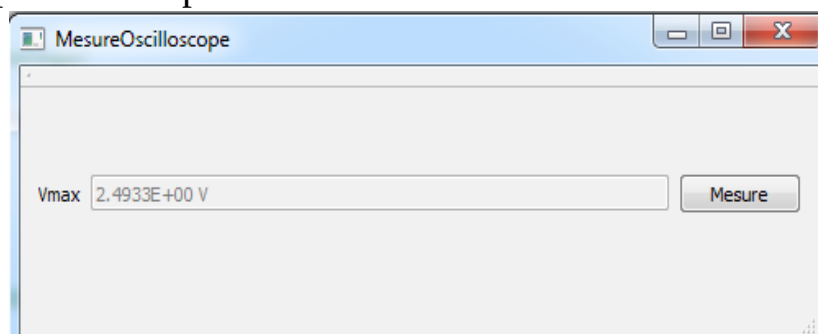
Cette commande permet de relever la valeur que l’on veut sur le canal spécifié.

On utilisera ensuite la fonction viScanf() pour récupérer la valeur mesurée avec l’option %t pour pouvoir récupérer toute la chaîne de caractères, même si elle est séparée par des espaces.

Ci-après, un code de test pour afficher la tension maximale mesurée sur la voie 1 de l’oscilloscope.

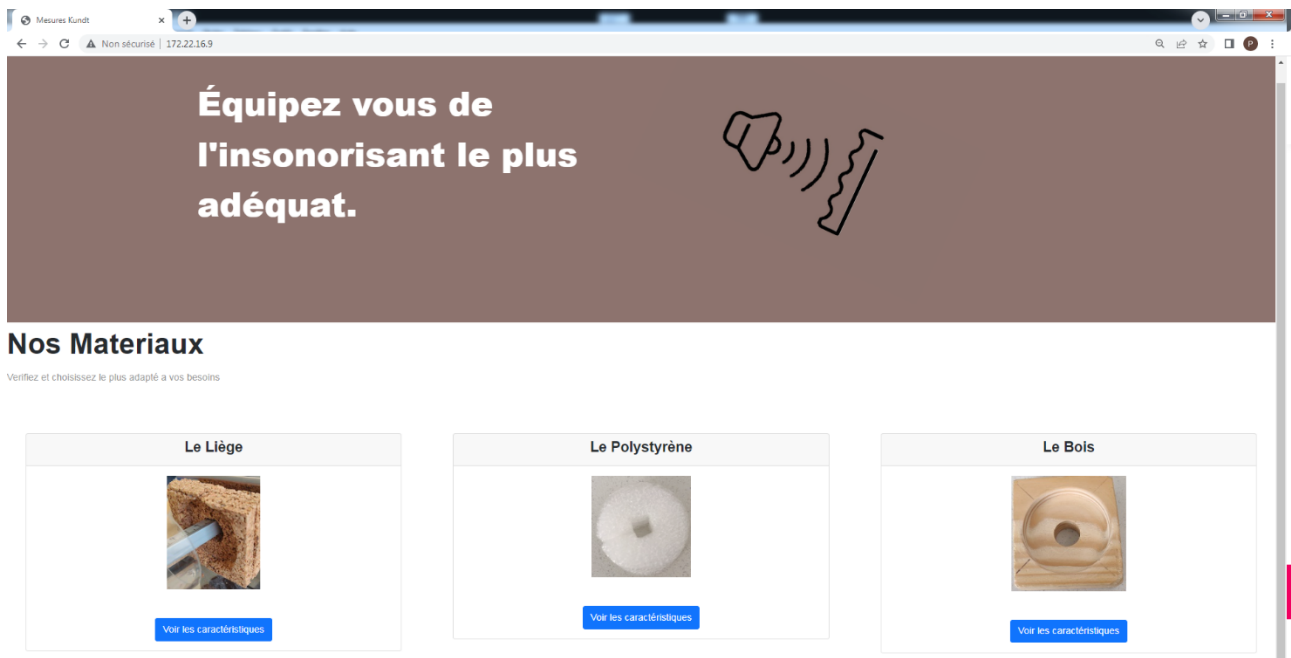
```
viPrintf(osc, (ViString)"MEAS:ITEM? VMAX,CHAN1\n"); //tension mesurée sur le channel 1
viScanf(osc, (ViString)"%t", &buf); //Lecture du resultat %t récupère toute la chaîne de caractere si separé par un espace
qDebug() << buf;
QString essai = QString(buf);
essai = essai + "V";
qDebug() << essai;
ui->EdtOscillo->setText(essai);
```

Le résultat obtenu est en valeur scientifique qu’il faudra convertir en valeur décimale pour pouvoir ensuite simplifier le calcul du coefficient d’absorption et simplifier la lecture des résultats.

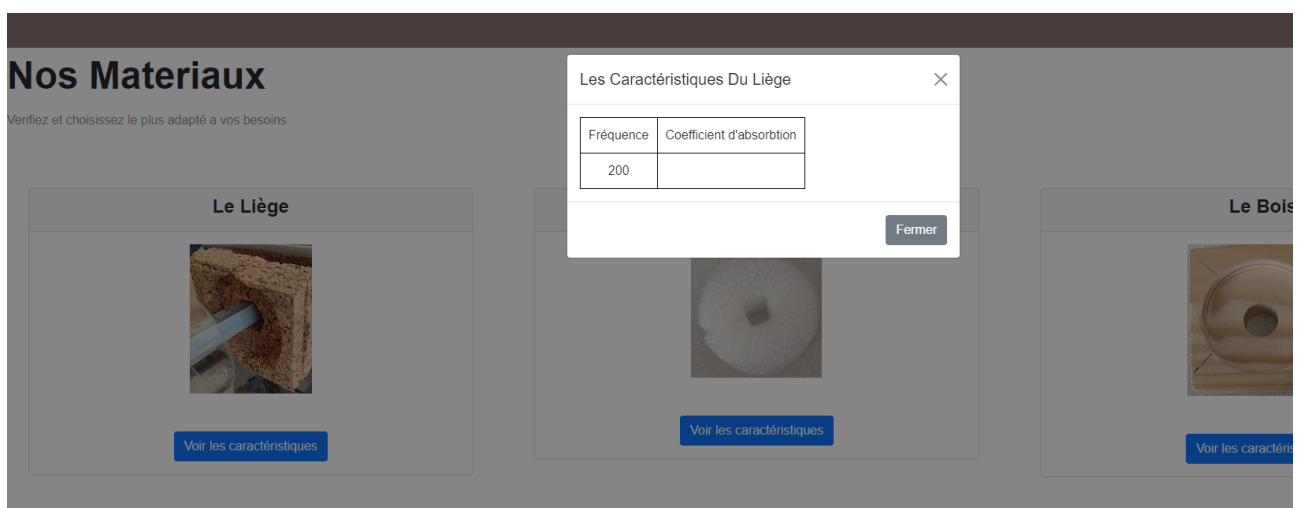


CREATION DE LA PAGE WEB

La page web est présente dans l'unique but d'afficher les coefficients d'absorption déjà enregistrés dans la base de données. Elle aura donc un style très simple.



Cette page à dans un premier temps été créée en HTML/CSS pour pouvoir avoir un aperçu rapide de la page que nous étions en train de créer. Nous avons ensuite utilisé Bootstrap qui est une collection d'outils utiles à la création du design de sites et d'applications web pour faire l'affichage des caractéristiques des matériaux.



Les caractéristiques sont ainsi affichées dans des modal.

Il a ensuite fallu passer la page HTML en PHP afin que notre page web puisse communiquer avec la base de données.

Nous avons ainsi utilisé une extension PHP appelée PDO (PHP Data Objects) qui permet d'accéder à une base de données avec PHP.

```
1  <?php
2
3  try{
4      //connection a la bdd
5      $db = new PDO('mysql:host=localhost;dbname=kundt;charset=latin1','root','');
6  }
7  catch(Exception $e)
8  {
9      die ('Erreur:'. $e->getMessage());
10 }
11 ?>
```

Nous devons ensuite mélanger PHP et HTML afin de pouvoir récupérer nos données et les afficher dans notre page Web.

```
1  <?php
2      $stabfreq = $db->prepare("SELECT `frequence` FROM `enregistrement` WHERE `id_materiau`=1");
3      $stabfreq->execute();
4
5      $coefliege = $db->prepare("SELECT `absorption` FROM `enregistrement` WHERE `id_materiau`=1");
6      $coefliege->execute();
7
8      for ($i=0; $i<13; $i++)
9      {
10
11
12      $frequences= $stabfreq->fetch();
13      $lieges= $coefliege->fetch();
14
15      ?>
16      <tr>
17          <td><?php echo $frequences['frequence'];?></td>
18          <td><?php echo($lieges['absorption']);?></td>
19      </tr>
20  <?php
21  }
22  ?>
23  </table>
```

Ce qui nous permet au final d'afficher nos caractéristiques à tout moment même si une valeur est changée dans la base de données.

Mesures Kuntt


Non sécurisé | 172.22.16.9

Équipez vous de l'insonorisant le adéquat.

Nos Materiaux


entifiez et choisissez le plus adapté à vos besoins

Le Liège



Voir les caractéristiques

Le Bois



Voir les caractéristiques

Les Caractéristiques Du Liège

Fréquence	Coefficient d'absorption
200	0.0198
250	0.01587
315	0.01261
400	0.00995
500	0.007968
630	0.006329
800	0.004987
1000	0.003992
1250	0.0031948
1600	0.0024968
2000	0.001998
2500	0.001590721
3150	0.001269

Fermer

REGROUPER TOUS LES MODULES

Les différents modules créés ont ensuite été mis sous la forme de fonctions car elles sont beaucoup utilisées lors d'une mesure.

On a d'abord fait une fonction qui récupère la position du micro dans le tube avec la fonction `checkPosition`

```
double MainWindow::checkPosition()//retourne la tension qui donne la position du micro
{
    for(int i = 0; i < 256; i++)
        buf[i] = 0;
    viPrintf(osc, (ViString)":MEAS:ITEM? VMAX,CHAN3\n"); //tension mesurée sur le channel 3
    viScanf(osc, (ViString)"%t",&buf);
    tensionPos = QString(buf).toDouble();
    return tensionPos;
}
```

Dans cette fonction on commence par vider le buffer qui va contenir le résultat de la mesure puis nous récupérons la valeur de cette position qui est en valeur scientifique que nous convertissons en double pour ensuite pouvoir l'utiliser dans le programme principal.

Ensuite la fonction `mesureTension` qui nous permet de récupérer la tension mesurée sur la voie 3 de l'oscilloscope.

```
double MainWindow::mesureTension()// mesure de la tension
{
    double tension_mesuree = 0;
    viPrintf(osc, (ViString)":MEAS:ITEM? VMAX,CHAN1\n"); //tension mesurée sur le channel 1
    viScanf(osc, (ViString)"%t",&buf); //Lecture du resultat %t récupère toute la chaine de caractere si separé par un espace
    tension_mesuree = QString(buf).toDouble();
    return tension_mesuree;
}
```

Cette fonction récupère la tension maximum mesurée sur la voie 1 qu'on convertit en double puis on retourne cette valeur.

Cette première version du code ne fonctionnait pas. Le problème venait du fait que la vitesse qu'on indiquait sur la liaison série n'était pas la même que celle qui était configurée sur l'Arduino.

Le problème n'étant pas réglé nous avons ensuite supposé que programme envoyait trop de données à l'Arduino ce qui faisait planter l'application.

Nous avons ensuite essayé de voir comment régler ce problème en essayant d'optimiser le code en créant d'autres fonctions pour éviter les parties qui se répétaient.

Comme la fonction pour faire bouger la tige dans la direction qu'on voulait.

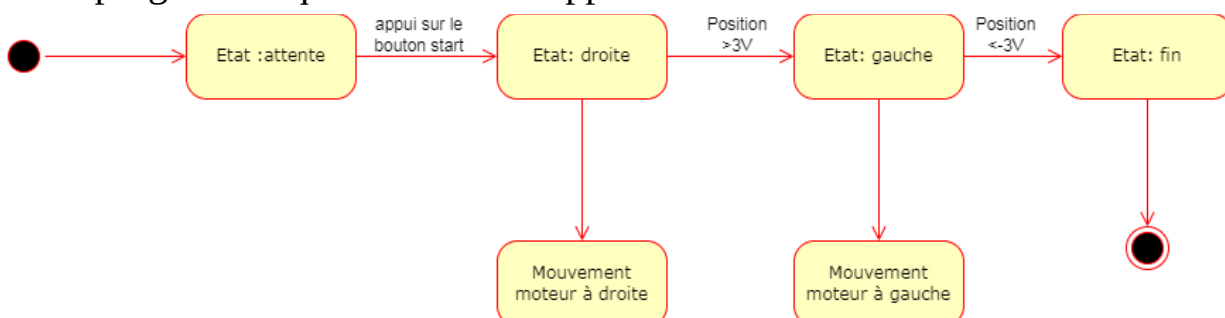
```
void MainWindow::movePosition(bool droite = true, double tensionPos)
{
    char caractere = "d";
    if(!droite)
        caractere = "g";
    for(double pos=tensionPos; pos>limite; pos=tensionPos)
    {
        if(arduino->isWritable())
            arduino->write(caractere); //mouvement micro vers le HP
        ui->Editcoef->setText("vers la droite");
        else
        {
            QMessageBox::critical(this,"Attention","L'écriture a échoué",QMessageBox::Ok);
            qDebug() << "Couldn't write to serial!";
            arduino->openConnection();
        }
        sleep(1);
    }
}
```

Cette fonction fait bouger dans la direction qu'on veut en écrivant « g » ou « d » sur la liaison série.

Les problèmes n'étant pas réglés nous avons utilisé le déboguer pour vérifier que nos variables avaient les bonnes valeurs.

En faisant cela nous avons remarqué que lorsque l'application est éteinte de façon brusque l'application le buffer qu'on utilise pour récupérer les valeurs de l'oscilloscope ne s'actualisait plus. Nous avons essayé de régler ce problème en remettant l'oscilloscope avec ses valeurs d'usines mais cela ne fonctionnait pas non plus. La seule solution que nous avons pour pallier ce problème est de redémarrer manuellement l'oscilloscope.

Nous avons ensuite créé une machine d'état pour synchroniser l'Arduino avec notre programme que nous allons appeler toutes les millisecondes.



Cette machine d'état a 4 états possibles :

- Etat attente, dans cet état le programme attend un appui sur le bouton start.
- Etat droite, dans cet état on vérifie que la position du micro est inférieure à 3V puis on envoie un "d" sur la liaison série qui permettra de faire un déplacement de la tige de 10 pas vers la droite pour ensuite faire la mesure de la tension sur la voie 1. On compare ensuite la valeur mesurée avec la valeur qu'on avait mesuré avant pour pouvoir récupérer la tension maximale.
- Etat gauche, dans cet état on vérifie que la position du micro est supérieure à -3V puis on envoie un "g" sur la liaison série qui permettra de faire un déplacement de la tige de 10 pas vers la gauche pour ensuite faire la mesure de la tension sur la voie 1. On compare ensuite la valeur mesurée avec la valeur qu'on avait mesuré avant pour pouvoir récupérer la tension minimale.
- Etat fin, dans cet état on coupe le GBF et le timer de qtimer et on enregistre ensuite le coefficient et la fréquence pour pouvoir afficher le graphique.

PARTIE PERSONNELLE : ELIOTT (Étudiant 2)

Arduino (1.8.13) :

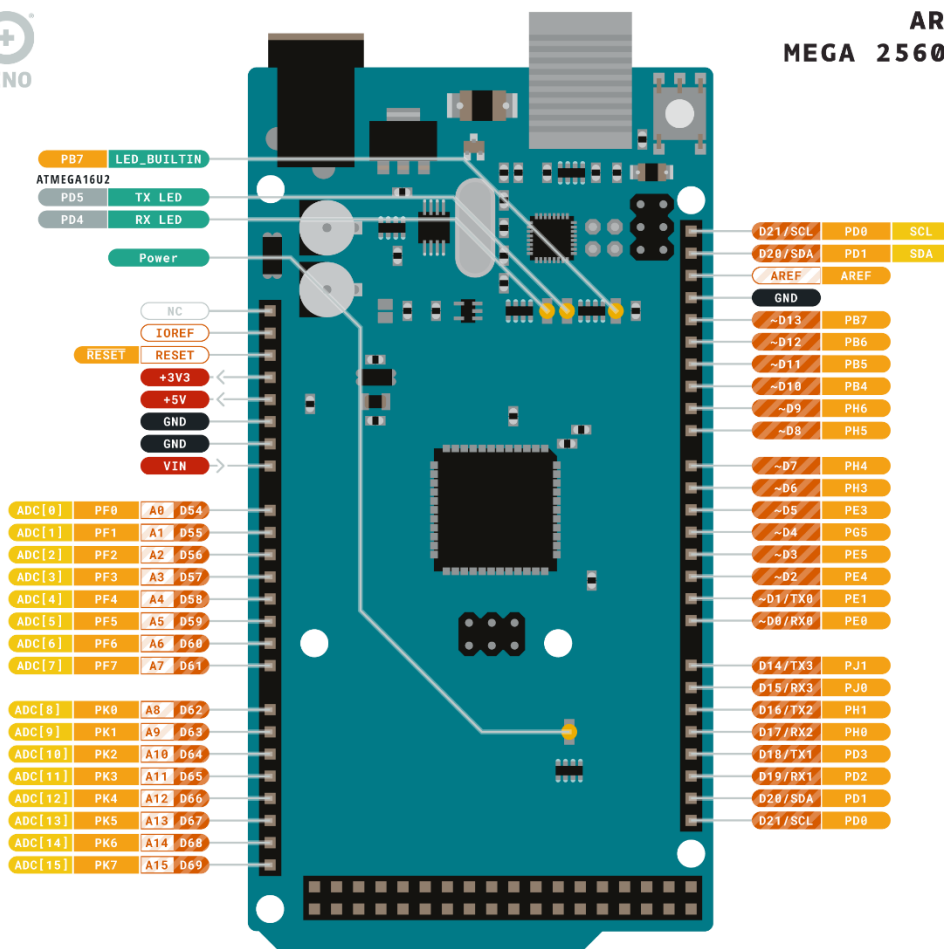


Présentation du montage :

Brochage de la carte Arduino Mega 2560 :



ARDUINO
MEGA 2560 REV3



Ground	Internal Pin	Digital Pin	Microcontroller's Port
Power	SWD Pin	Analog Pin	
LED	Other Pin	Default	

ARDUINO.CC

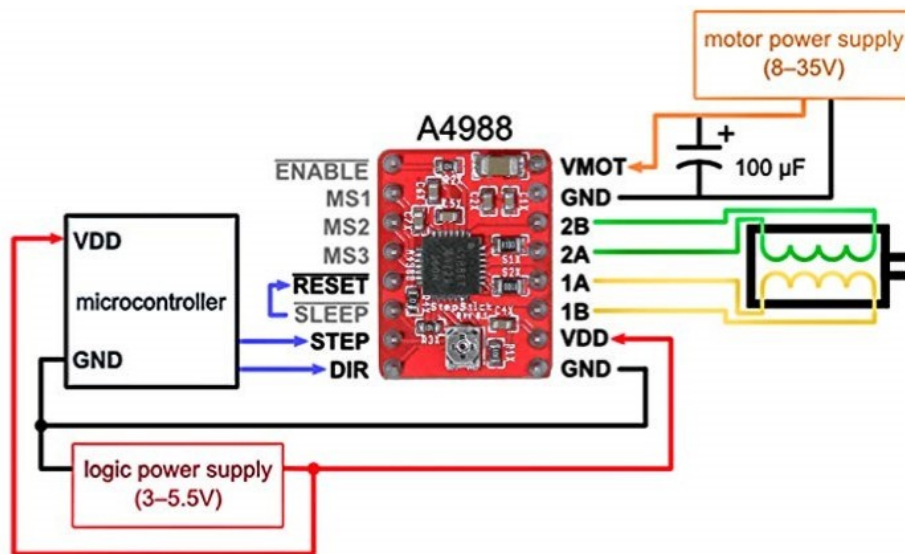


This work is licensed under the Creative Commons Attribution-ShareAlike 4.0 International License. To view a copy of this license, visit <http://creativecommons.org/licenses/by-sa/4.0/> or send a letter to Creative Commons, PO Box 1866, Mountain View, CA 94042, USA.

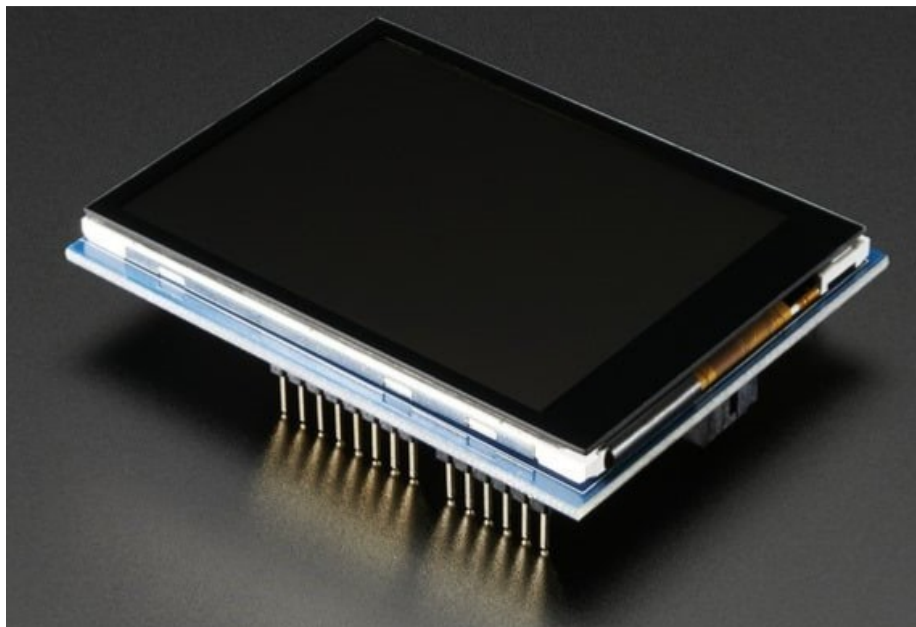
Liaison entre l'Arduino et le driver :

```
// brochage de la carte
#define EN_A4988 22
#define MS1_A4988 24
#define MS2_A4988 26
#define MS3_A4988 28
#define RST_A4988 30
#define SLP_A4988 32
#define STEP_A4988 34
#define DIR_A4988 36
```

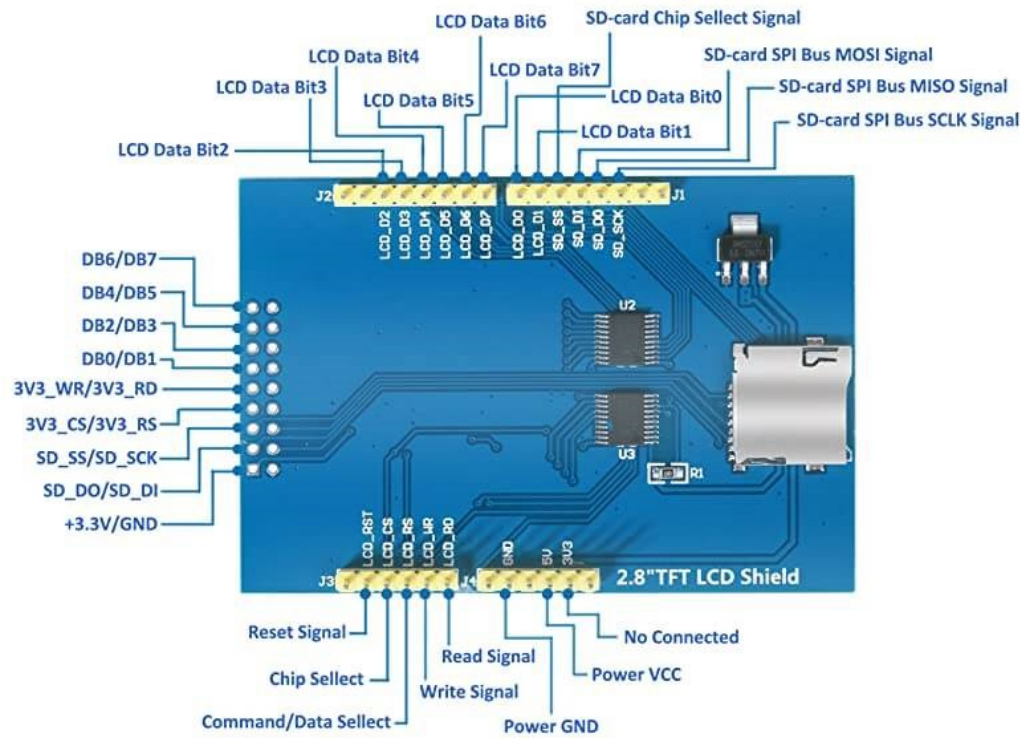
Brochage du driver Polulu A4988 :



Écran Adafruit 2.8 TFT LCD Shield :



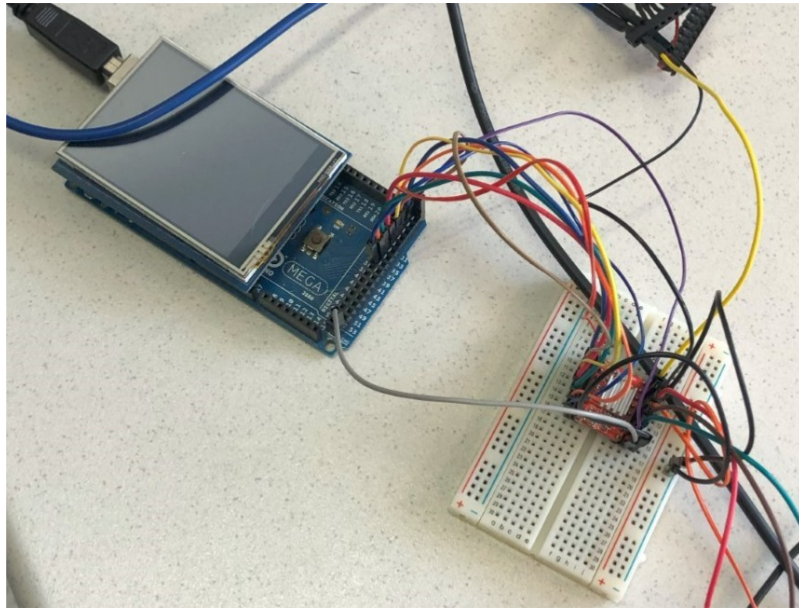
Brochage de l'écran :



Montage V1 :



Montage V2 :



Montage final :



Séparation du code en plusieurs fichiers :

pilote_arduino.ino pilotage_ecran.cpp pilotage_ecran.h pilotage_moteur.cpp pilotage_moteur.h

« pilotage_moteur.h » contient les déclarations des fonctions de pilotage du moteur.

```
1  #ifndef PILOTAGE_MOTEUR_H
2  #define PILOTAGE_MOTEUR_H
3
4  #include "SPI.h"
5
6  void setupSerialMoteur();
7  void avanceGauche();
8  void avanceDroite();
9  void arretMoteur();
10
11 #endif
```

« pilotage_moteur.cpp » contient les définitions des fonctions de pilotage du moteur (ex : la fonction *arretMoteur()*).

```
void arretMoteur() {
    // Broche ENABLE inactive
    digitalWrite( EN_A4988, HIGH );
}
```

```
1  #ifndef PILOTAGE_ECRAN_H
2  #define PILOTAGE_ECRAN_H
3
4  #include "SPI.h"
5  #include "Adafruit_GFX.h"
6  #include "Adafruit_ILI9341.h"
7
8  // For the Adafruit shield, these are the default.
9  #define TFT_DC 9
10 #define TFT_CS 10
11
12 void affichageMesureEnCours();
13 void setupEcran();
14 void affichageMesures(String data);
15 void affichageStopManuel(bool arret_manuel);
16 void affichageLiaison(bool is_open);
17
18 #endif
```

« pilotage_ecran.h » contient les bibliothèques et les déclarations des fonctions de pilotage et d'affichage sur l'écran.

```
void affichageMesures(String data) {
    tft.setRotation(4);
    tft.setCursor(0, 0);
    tft.fillScreen(ILI9341_BLACK);
    tft.setTextColor(ILI9341_WHITE);
    tft.setTextSize(2);
    tft.println(data);
}
```

« pilotage_ecran.cpp » contient les définitions des fonctions de pilotage de l'écran (ex : la fonction *affichageMesures()*).

tft est une instance de la classe *Adafruit_ILI9341*.

```
5  // Use hardware SPI (on Uno, #13, #12, #11) and the above for CS/DC
6  Adafruit_ILI9341 tft = Adafruit_ILI9341(TFT_CS, TFT_DC);
```

« pilote_arduino » contient les fonctions *setup()* et *loop()* nécessaires à la carte.

```
8 void setup() {
9     setupSerialMoteur();
10    setupEcran();
11 }
```

Dans la fonction *setup()* on retrouve l'initialisation des broches contrôlant l'écran et le moteur.

Dans la fonction *loop()* on retrouve un switch permettant de gérer l'affichage et les déplacements du moteur en fonction du caractère reçu via la liaison série.

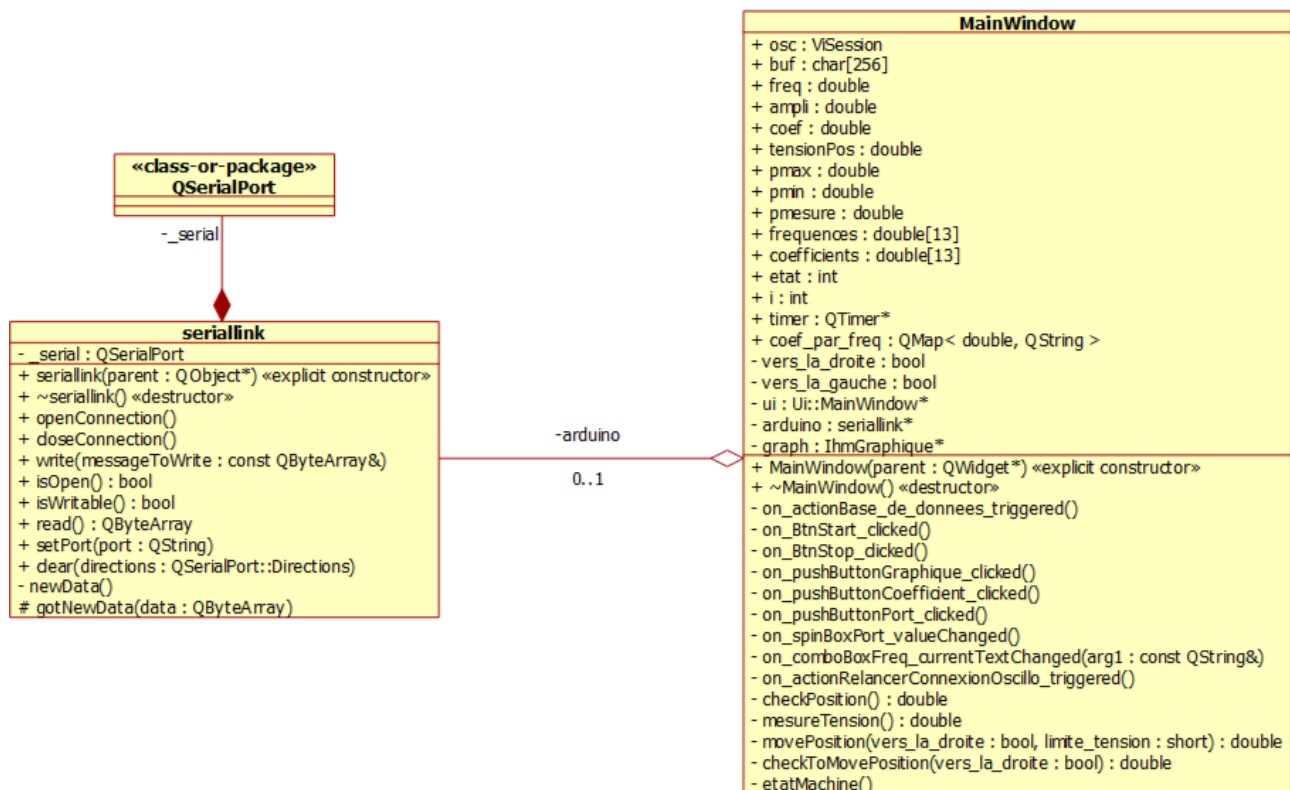
```
13 void loop() {
14     while (Serial.available())
15     {
16         data = Serial.read(); //Lecture des données sur le port série
17
18         switch (data) {
19             //Ouverture liaison
20             case 'o':
21                 affichageLiaison(true);
22                 break;
23
24             //Fermeture liaison
25             case 'f':
26                 affichageLiaison(false);
27                 break;
28
29             //Demande de déplacement à droite
30             case 'd':
31                 avanceDroite();
32                 affichageMesureEnCours();
33                 break;
34
35             //Demande de déplacement à gauche
36             case 'g':
37                 avanceGauche();
38                 affichageMesureEnCours();
39                 break;
40
41             //Demande d'arrêt d'urgence
42             case 's':
43                 arretMoteur();
44                 affichageStopManuel(true);
45                 break;
46         }
47         arretMoteur();
48     }
49     while (Serial.available()) {
50         donnees_a_afficher = Serial.readString();
51         delimitateur = donnees_a_afficher.indexOf("$");
52         delimitateur_1 = donnees_a_afficher.indexOf("$", delimitateur + 1);
53         donnees_a_afficher = donnees_a_afficher.substring(delimitateur + 1, delimitateur_1);
54         donnees_a_afficher.trim();
55         affichageMesures(donnees_a_afficher);
56         //affichageStopManuel(false);
57     }
58 }
```

Qt Creator (5.11.1) :



Qt Creator m'a été utile à plusieurs reprises notamment pour créer des interfaces et des modules permettant de communiquer à la fois avec Arduino mais aussi avec la base de données.

Création de la classe *seriallink* pour que l'IHM puisse communiquer avec Arduino :




```

seriallink::seriallink(QObject *parent) : QObject(parent)
{
    _serial.setPortName("COM10");// COM10 Elliott COM5 Patrick
    _serial.setBaudRate(QSerialPort::Baud115200);
    _serial.setDataBits(QSerialPort::Data8);
    _serial.setParity(QSerialPort::NoParity);
    _serial.setStopBits(QSerialPort::OneStop);
    _serial.setFlowControl(QSerialPort::NoFlowControl);
}

```

La liaison série est initialisée comme tel dans le constructeur de la classe *seriallink* :

- le port COM par défaut est le numéro 10 ;
- le débit binaire est de 115200 bauds ;
- 8 bits de données ;
- aucun bit de parité ;
- 1 bit de stop ;
- aucun bit de contrôle de flux ;

Cette classe est composée de diverses fonctions permettant de configurer et de manipuler la liaison série. On peut notamment l'ouvrir, la fermer, écrire dessus, vérifier si elle est ouverte ou encore lire les données présentes dessus.

Cette classe est utilisée comme suit dans l'application :

```

seriallink *arduino;

```

On l'utilise alors principalement pour écrire divers caractères sur la liaison série. Ces caractères sont alors reçus et traités dans le switch vu précédemment par Arduino.

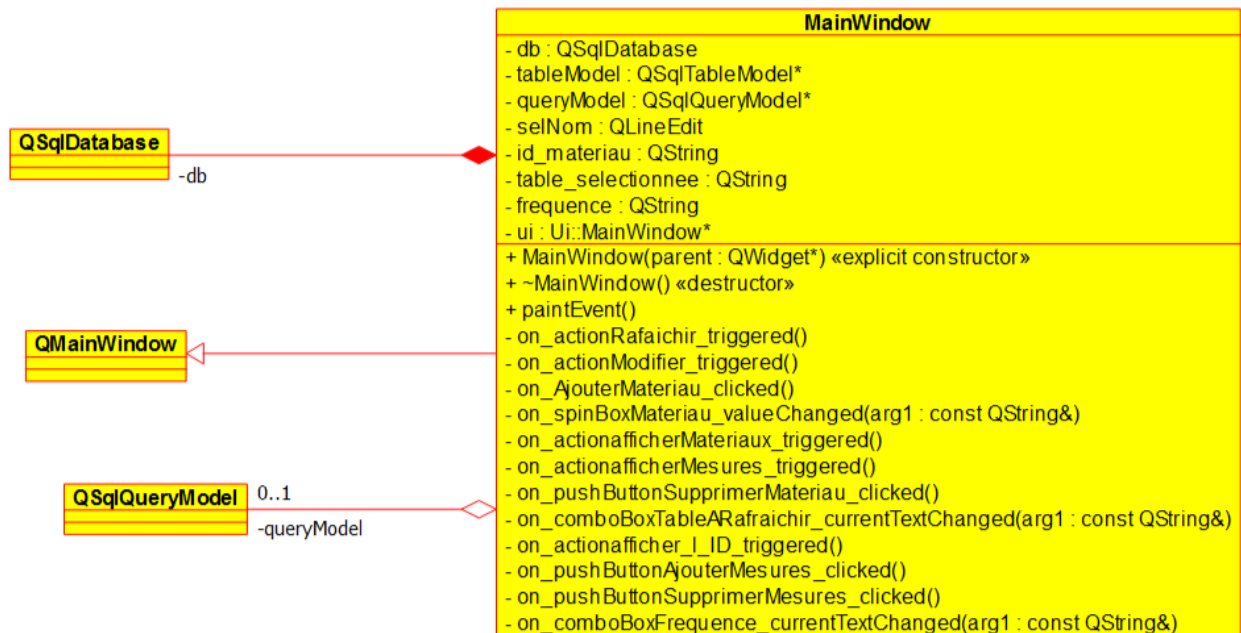
```

arduino = new seriallink;
arduino->openConnection();
arduino->write("o");|

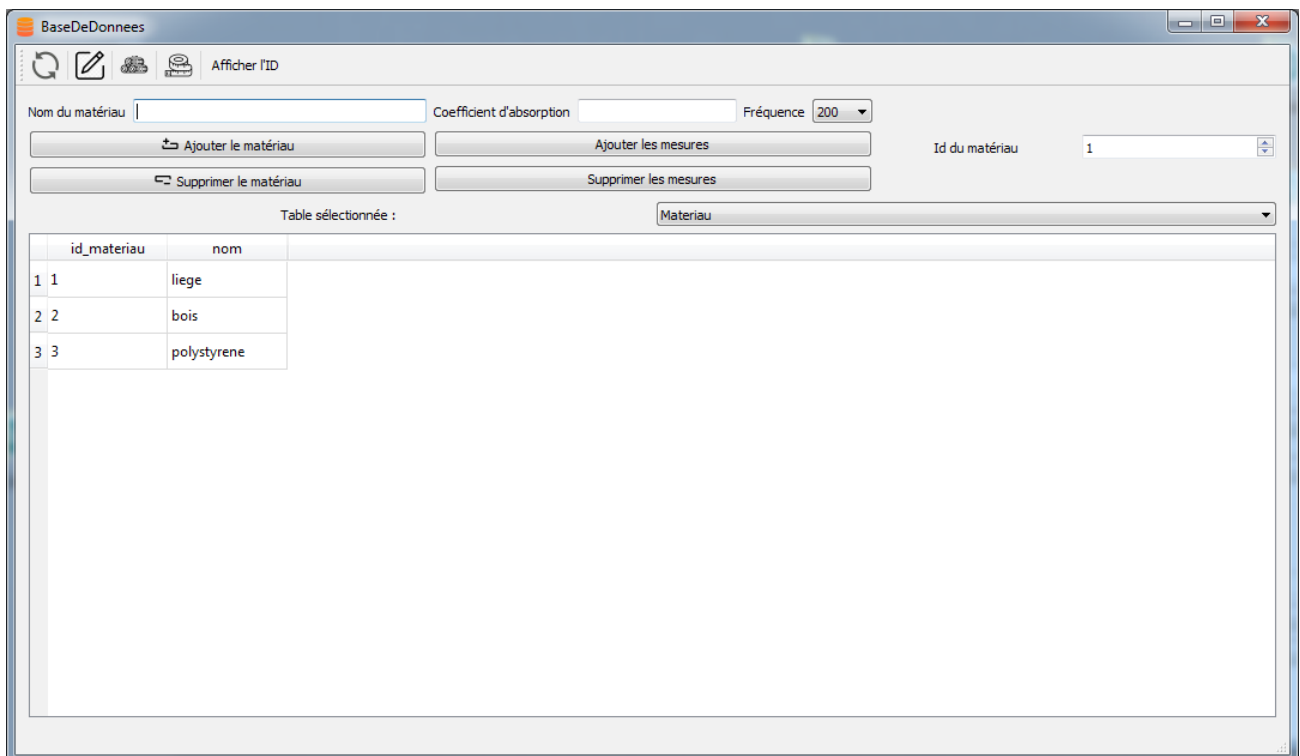
```

Création d'une application permettant de lire et d'écrire les informations pertinentes dans la base de données :

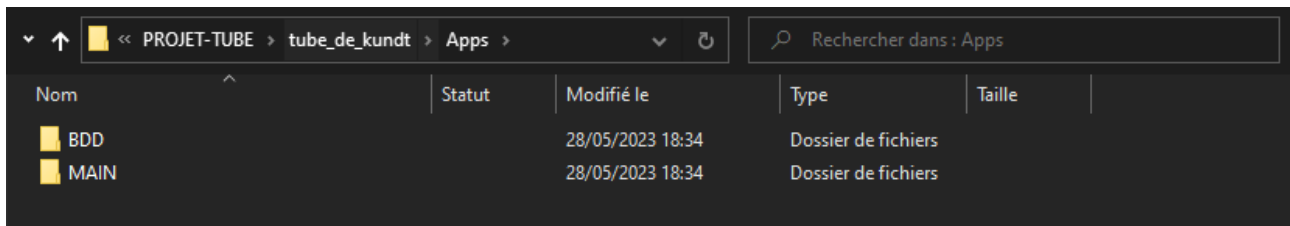
Diagramme de classe de la fenêtre principale d'accès à la base de données :



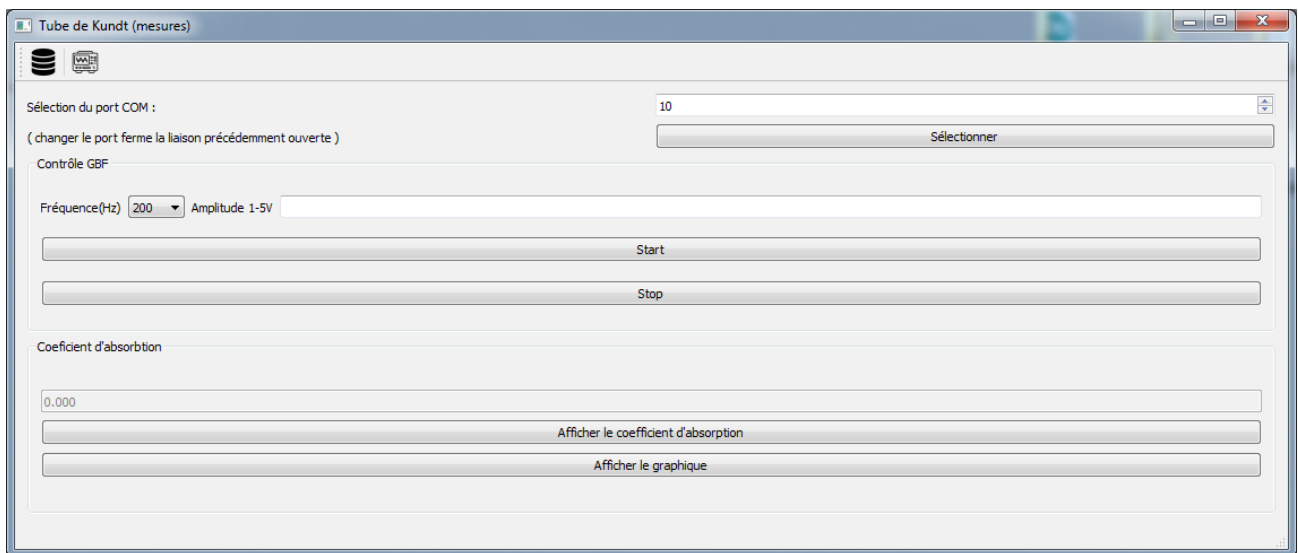
Interface principale d'interaction avec la base de données :



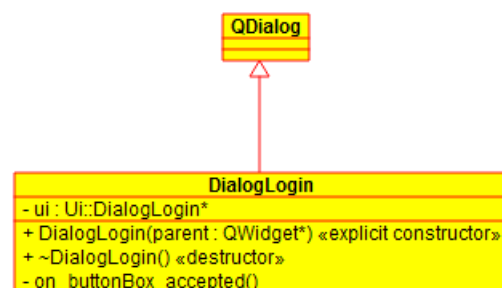
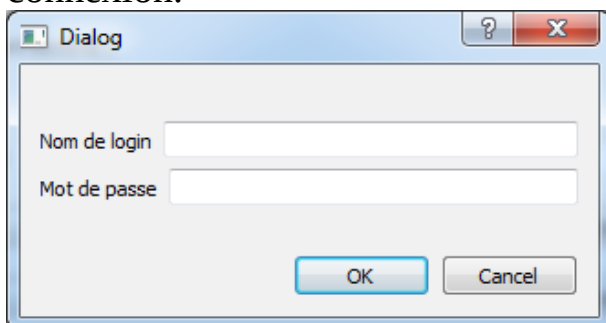
Il est possible d'accéder à cette application par deux biais. Soit en lançant le .exe lui étant attribué et disponible dans le répertoire « Apps/BDD » du projet :



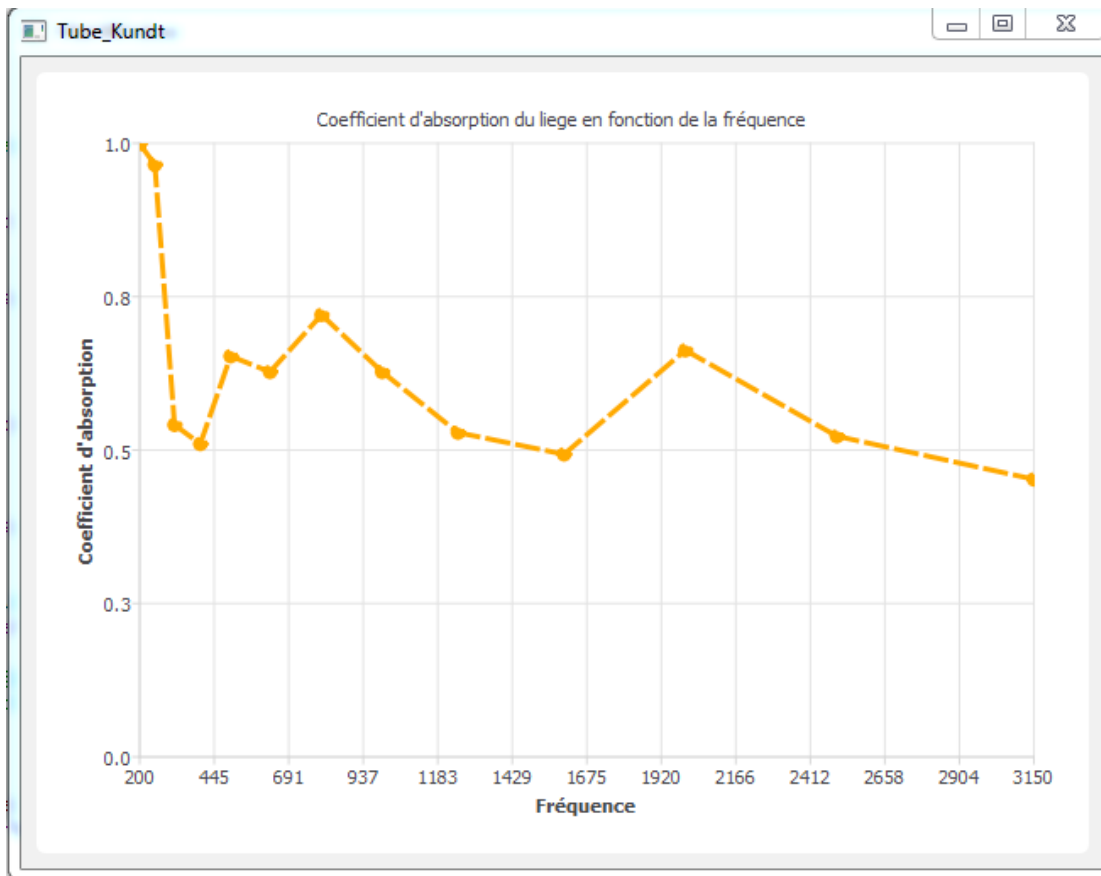
Soit depuis l'IHM en cliquant sur le bouton tout en haut à gauche qui lui est associé :



On accède à cette interface grâce à un nom d'utilisateur et un mot de passe. Lorsque le .exe ou le bouton sont cliqués, une boîte de dialogue s'ouvre demandant à l'utilisateur de renseigner ces informations permettant la connexion.



Création d'un module permettant de tracer la courbe du coefficient d'absorption du matériau testé :



Cette fenêtre s'ouvre grâce au bouton « Afficher le graphique de l'IHM ». Il s'agit d'un widget qui utilise les classes *Qwidget* et *QChart*. Elle récupère les mesures et les affiche dans ce graphique.

```
void IhmGraphique::dessinerGraphique(double frequences[13], double coefs[13])
{
    courbe = new QLineSeries();
    for(int i =0; i < 13; i++)
        *courbe << QPointF(frequences[i],coefs[i]);
}
```

Apache (2.4.35) :



J'ai dû modifier les fichiers suivants pour permettre à mon collègue d'accéder à la page web hébergée sur le serveur apache présent sur mon ordinateur :

- httpd.conf
- httpd-vhosts.conf

Dans httpd.conf :

```
DocumentRoot "C:/wamp64/www/tube-de-kundt"
<Directory "C:/wamp64/www/tube-de-kundt/">
#
# Possible values for the Options directive are "None", "All",
# or any combination of:
#   Indexes Includes FollowSymLinks SymLinksifOwnerMatch ExecCGI MultiViews
#
# Note that "MultiViews" must be named *explicitly* --- "Options All"
# doesn't give it to you.
#
# The Options directive is both complicated and important. Please see
# http://httpd.apache.org/docs/2.4/mod/core.html#options
# for more information.
#
Options +Indexes +FollowSymLinks +Multiviews

#
# AllowOverride controls what directives may be placed in .htaccess files.
# It can be "All", "None", or any combination of the keywords:
#   AllowOverride FileInfo AuthConfig Limit
#
AllowOverride all

#
# Controls who can get stuff from this server.
#
# onlineoffline tag - don't remove
Require local
</Directory>
```

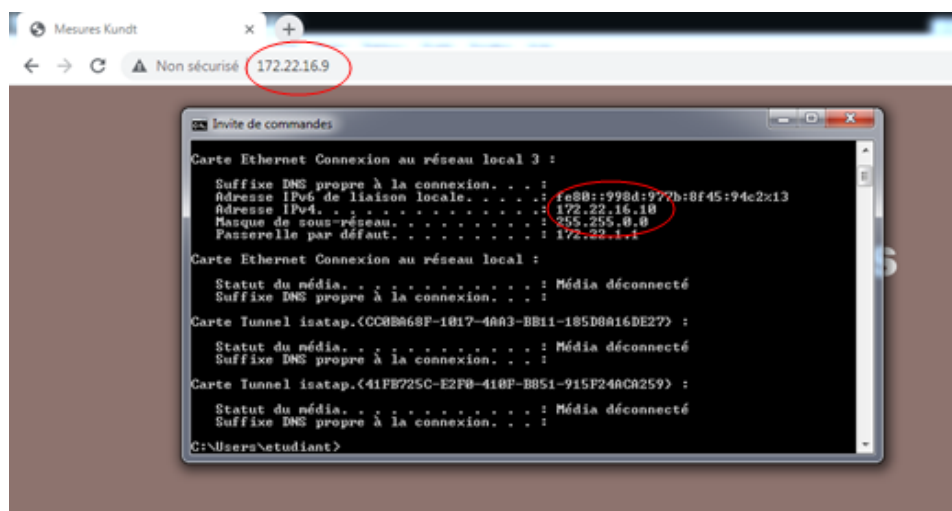
La ligne « DocumentRoot » permet de définir le fichier qui sera ouvert par défaut lorsqu'un client entre l'adresse IP de ma machine dans son navigateur. Les autres lignes sont expliquées avec les commentaires présents dans le code.

La seule autre ligne que j'ai eu à modifier était la ligne « Require » qui permet de définir qui a accès à ma machine. La valeur « local » y donne accès à toutes les machines sur le même réseau local que la mienne.

Dans httpd-vhosts.conf :

```
1  # Virtual Hosts
2  #
3  <VirtualHost *:80>
4      ServerName localhost
5      ServerAlias localhost
6      DocumentRoot "C:/wamp64/www/tube-de-kundt"
7      <Directory "C:/wamp64/www/tube-de-kundt/">
8          Options +Indexes +Includes +FollowSymLinks +MultiViews
9          AllowOverride All
10         Require all granted
11     </Directory>
12 </VirtualHost>
13
```

Ce fichier possède sensiblement les mêmes paramètres que le précédent, seulement ici, la valeur « all granted » de la ligne « Require » indique qu'aucune adresse n'est bloquée, autrement dit elles sont toutes autorisées et ont toutes accès à ma machine (tant qu'elles sont dans le même réseau local).



On voit ici que mon collègue (172.22.16.10) à accès au serveur présent sur ma machine (172.22.16.9).

CONCLUSION

Le projet de création d'un système automatisé pour mesurer le coefficient d'absorption d'énergie acoustique des matériaux à l'aide d'un tube de Kundt a été une expérience enrichissante. Nous avons réussi à concevoir et mettre en place un banc de mesure fonctionnel, qui utilise un oscilloscope et un générateur de basses fréquences pour contrôler le tube et enregistrer les mesures.

L'interface utilisateur développée permet de piloter facilement le système, de sélectionner les matériaux à tester et d'afficher les résultats des mesures. De plus, nous avons créé une base de données pour stocker les mesures et les autres paramètres pertinents.

Cependant, nous avons rencontré quelques défis lors de la réalisation des mesures manuelles, notamment en ce qui concerne la sensibilité du tube aux bruits extérieurs. Cela souligne l'importance de l'automatisation du tube pour minimiser les perturbations lors des mesures.

Pour améliorer ce projet à l'avenir, nous recommandons d'automatiser complètement le tube de Kundt en utilisant un moteur pas-à-pas contrôlé par une carte Arduino. Cela permettrait d'éliminer les erreurs humaines et de garantir des mesures plus précises et reproductibles.

De plus, il serait bénéfique d'explorer d'autres matériaux acoustiques et d'étendre la plage de fréquences testées. Cela permettrait d'obtenir une meilleure compréhension des propriétés d'absorption acoustique des différents matériaux et d'améliorer la précision des résultats.

En conclusion, ce projet nous a permis d'acquérir des connaissances approfondies sur le tube de Kundt et les mesures d'absorption acoustique. Il a également mis en évidence l'importance de l'automatisation pour simplifier le processus de mesure et obtenir des résultats fiables. Nous espérons que ce système automatisé pourra être utilisé par les fabricants et distributeurs de matériaux acoustiques afin de faciliter leur sélection et leur commercialisation.

MainWindow :

```
#include "mainwindow.h"
#include "ui_mainwindow.h"
#include <QProcess>
#include <QDebug>
#include <QMessageBox>
#include <unistd.h>
#include <QThread>
MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MainWindow)
{
    timer = new QTimer(this);
    timer->interval();
    QObject::connect(timer,&QTimer::timeout, [this]() { etatMachine(); });

    // ouverture de la session ressource manager
    ViSession rscmng;
    ViStatus stat = viOpenDefaultRM(&rscmng);

    // Recherche de l'oscilloscope
    ViChar viFound[VI_FIND_BUFLen];
    ViUInt32 nFound;
    ViFindList listOfFound;
    stat = viFindRsrc(rscmng, (ViString)"USB?*", &listOfFound, &nFound,
viFound);

    stat = viOpen(rscmng, viFound, VI_NULL, VI_NULL, &osc);
    if (stat < VI_SUCCESS)
    {
        QMessageBox::critical(this,"Attention","Echec connexion
oscilloscope",QMessageBox::Ok);
        qDebug() << "Echec connexion oscilloscope";
    }
    ui->setUpUi(this);
    viPrintf(osc, (ViString)":CHAN3:SCAL 4\n");
    arduino = new seriallink;
    arduino->openConnection();
    arduino->write("o");

    for(int i = 0; i < 13; i++)
    {

        coefficients[i] = {0.01};
    }
    frequences[0] = {200};
    frequences[1] = {250};
    frequences[2] = {315};
    frequences[3] = {400};
    frequences[4] = {500};
    frequences[5] = {630};
    frequences[6] = {800};
    frequences[7] = {1000};
    frequences[8] = {1250};
    frequences[9] = {1600};
    frequences[10] = {2000};
    frequences[11] = {2500};
    frequences[12] = {3150};
}

MainWindow::~MainWindow()
```



```

{
    delete ui;
}

void MainWindow::on_actionBase_de_donnees_triggered()
{
    QProcess *process = new QProcess(this);
    process-
>start("\"C:\\Users\\etudiant\\Desktop\\tube_de_kundt\\Apps\\BDD\\MySqlQt.exe\"")
);
    //chemin Elliott et Patrick
    qDebug() << process->errorString();
}

void MainWindow::on_BtnStart_clicked()
{
    freq = ui->comboBoxFreq->currentText().toDouble();
    ampli = ui->EditAmpli->text().toDouble();
    if(ampli > 5 || ampli < 1)
    {
        QMessageBox::critical(this, "Attention", "amplitude supérieure à 5V ou
inférieure à 1V", QMessageBox::Ok);
        return;
    }
    viPrintf(osc, ":APPLY:SIN ,%f,%f\n", freq, ampli); //on applique un signal
sinusoidal de fréquence et amplitude choisies
    viPrintf(osc, ":AUT\n"); // autoset oscillo
    viPrintf(osc, (ViString)":CHAN3:SCAL 10\n");
    viPrintf(osc, (ViString)":CHAN3:SCAL 10\n");
    tensionPos = checkPosition();
    // double n, tension_max_mesuree = 1, tension_min_mesuree = 1;

    if(tensionPos > -5 && tensionPos < 5) // -5 et 5 sont les tension max et
min pour la position du micro

    {

        etat = 1;
        timer->start(1);

        QByteArray recu = arduino->read();
        qDebug() << recu << tensionPos;

        if (recu == "d_ACK")
            etat = 2;
        else
            if(recu == "g_ACK")
                etat = 4;
        }
        else
            QMessageBox::critical(this, "Attention", "Verifiez la position du
micro", QMessageBox::Ok);
    }

}

void MainWindow::on_BtnStop_clicked() //Arret d'urgence
{
    while(!(arduino->isWritable()));
    arduino->write("s");

    viPrintf(osc, (ViString)":OUTP1 :0\n"); //COUPER LE GBF DE L'OSCILLO
    etat = 0;
}

```

```

}

double MainWindow::checkPosition()//retourne la tension qui donne la position du
micro
{
    for(int i = 0; i < 256; i++)
        buf[i] = 0;
    viPrintf(osc, (ViString)":MEAS:ITEM? VMAX,CHAN3\n"); //tension mesurée sur
le channel 3
    viScanf(osc, (ViString)"%t",&buf);
    tensionPos = QString(buf).toDouble();
    return tensionPos;
}

double MainWindow::mesureTension()// mesure de la tension
{
    double tension_mesuree = 0;
    viPrintf(osc, (ViString)":MEAS:ITEM? VMAX,CHAN1\n"); //tension mesurée sur
le channel 1
    viScanf(osc, (ViString)"%t",&buf); //Lecture du resultat %t récupère
toute la chaine de caractere si separé par un espace
    tension_mesuree = QString(buf).toDouble();
    return tension_mesuree;
}

void MainWindow::etatMachine()
{
    tensionPos = checkPosition();
    switch (etat)
    {
    case 0:
        break;
    case 1:
        qDebug() << "demande deplacement droite" << "tensionPos : " <<
tensionPos;
        if(tensionPos >= 3)
            etat = 2;
        arduino->write("d");
        pmesure = mesureTension();
        if((pmax<pmesure) && pmesure < 10)
            pmax = pmesure;
        qDebug() << pmax;
        sleep(1);
        break;

    case 2:
        qDebug()<<"demande deplacement gauche" << "tensionPos : " << tensionPos;
        if(tensionPos <= -3)
            etat = 3;
        pmesure = 0;
        arduino->write("g");
        pmesure=mesureTension();
        if(pmesure < pmin)
            pmin = pmesure;
        qDebug() << pmin;
        sleep(1);
        break;
    case 3:
        viPrintf(osc, (ViString)":OUTP1 :0\n");
        timer->stop();
        frequences[i] = freq;
        coefficients[i] = coef;
    }
}

```

```

        i++;
        break;
    }
}

void MainWindow::on_pushButtonGraphique_clicked()
{
    coefficients[0] = 0.74851;
    coefficients[1] = 0.51234;
    coefficients[2] = 0.39541;
    coefficients[3] = 0.28743;
    graph = new IhmGraphique(0);
    graph->dessinerGraphique(frequences, coefficients);
    graph->show();
}

void MainWindow::on_pushButtonCoefficient_clicked()
{
    double n = pmax / pmin;
    coef = pow((n-1)/(n+1), 2);
    coef = 1 - coef;

    qDebug() << coef;

    ui->Editcoef->setText(QString::number(coef, 'f', 3));
    if(arduino->isOpen())
    {
        arduino->write("f");
        arduino->closeConnection();
    }
    arduino->openConnection();
    arduino->write("o");
    arduino->clear();
    QByteArray donnees_a_afficher = ("$frequence : " + QByteArray::number(freq)
+ "      coef : " + QByteArray::number(coef));
    arduino->write(donnees_a_afficher);
}

void MainWindow::on_pushButtonPort_clicked()
{
    QString port = ui->spinBoxPort->text();
    arduino->setPort(port);
    arduino->openConnection();
    if(!arduino->isOpen())
        qDebug() << "Problème à l'ouverture du port sélectionné";
    else
        qDebug() << "connexion au nouveau port COM ok!";
    arduino->write("o");

    qDebug() << "port demandé : " + port;
    tensionPos=checkPosition();
    while(tensionPos>0)
        arduino->write("g");
}

void MainWindow::on_spinBoxPort_valueChanged()
{
    arduino->write("f");
    arduino->closeConnection();
}

void MainWindow::on_comboBoxFreq_currentTextChanged(const QString &arg1)

```

```

{
    freq = arg1.toDouble();
    qDebug() << "changement de frequence : " << freq;
}

void MainWindow::on_actionRelancerConnexionOscillo_triggered()
{
    // ouverture de la session ressource manager
    ViSession rscmng;
    ViStatus stat = viOpenDefaultRM(&rscmng);

    // Recherche de l'oscilloscope
    ViChar viFound[VI_FIND_BUFLN];
    ViUInt32 nFound;
    ViFindList listOfFound;
    stat = viFindRsrc(rscmng, (ViString)"USB?*", &listOfFound, &nFound,
viFound);

    stat = viOpen(rscmng, viFound, VI_NULL, VI_NULL, &osc);
    if (stat < VI_SUCCESS)
    {
        QMessageBox::critical(this, "Attention", "Echec connexion
oscilloscope", QMessageBox::Ok);
        qDebug() << "Echec connexion oscilloscope";
    }
}

```

Serial link

```
#include "seriallink.h"
#include <QDebug>
#include <QMessageBox>

seriallink::seriallink(QObject *parent) : QObject(parent)
{
    _serial.setPortName("COM10");// COM10 Elliott COM5 Patrick
    _serial.setBaudRate(QSerialPort::Baud115200);
    _serial.setDataBits(QSerialPort::Data8);
    _serial.setParity(QSerialPort::NoParity);
    _serial.setStopBits(QSerialPort::OneStop);
    _serial.setFlowControl(QSerialPort::NoFlowControl);

    connect(&_amp_serial, &QSerialPort::readyRead, this, &seriallink::newData);
}

seriallink::~seriallink(){
    closeConnection();
}

void seriallink::closeConnection(){
    _serial.close();
}

void seriallink::openConnection(){
    if(!_serial.isOpen())
        if(!_serial.open(QIODevice::ReadWrite))
        {
            qDebug() << "connexion avec la carte Arduino impossible";
        }
        else
            qDebug() << "connexion ok";
    else
        qDebug() << "connexion ok";
    this->clear();
}

void seriallink::write(const QByteArray &messageToWrite){
    _serial.write(messageToWrite);
}

bool seriallink::isOpen(){
    return _serial.isOpen();
}

bool seriallink::isWritable(){
    return _serial.isWritable();
}

QByteArray seriallink::read(){
    return _serial.readAll();
}

void seriallink::newData(){
    emit gotNewData(_serial.readAll());
}
```

```
void seriallink::setPort(QString port){  
    _serial.setPortName("COM" + port);  
}  
  
void seriallink::clear(QSerialPort::Directions directions)  
{  
    _serial.clear(directions);  
}
```

ihmgraphique

```
#include "ihmgraphique.h"
IhmGraphique::IhmGraphique(QWidget *parent)
    : QMainWindow(parent)
{

}

IhmGraphique::~IhmGraphique()
{

}

void IhmGraphique::dessinerGraphique(double frequences[13], double coefs[13])
{
    courbe = new QLineSeries();

    /*courbe << QPointF(200, 0.9993) << QPointF(250, 0.9651) << QPointF(315,
0.5411) << QPointF(400, 0.5100) << QPointF(500, 0.6530) << QPointF(630, 0.6274)
<< QPointF(800, 0.7204) << QPointF(1000, 0.6274) << QPointF(1250, 0.5285) <<
QPointF(1600, 0.4931) << QPointF(2000, 0.6627) << QPointF(2500, 0.5226) <<
QPointF(3150, 0.4523);

    for(int i =0; i < 13; i++)
        *courbe << QPointF(frequences[i],coefs[i]);

    QPen pen(0x059605);
    pen.setColor("orange");
    pen.setWidth(3);
    pen.setStyle(Qt::DashLine);
    courbe->setPen(pen);

    // Le graphe
    graphe = new QChart();
    graphe->addSeries(courbe);
    graphe->setTitle("Coefficient d'absorption en fonction de la fréquence");
    // Légende
    graphe->legend()->hide();
    graphe->legend()->setAlignment(Qt::AlignBottom);

    // axe abscisses
    QValueAxis *axeX = new QValueAxis;
    axeX->setTickCount(13);
    axeX->setRange(200, 3150);
    axeX->setLabelFormat("%i");
    axeX->setTitleText("Fréquence");
    graphe->addAxis(axeX, Qt::AlignBottom);
    courbe->attachAxis(axeX);

    // axe ordonnées
    QValueAxis *axeY = new QValueAxis;
    axeY->setRange(0, 1);
    axeY->setLabelFormat("%.1f");
    axeY->setTitleText(QString::fromUtf8("Coefficient d'absorption"));
    graphe->addAxis(axeY, Qt::AlignLeft);
    courbe->setPointsVisible(true);
    //courbe->setPointLabelsFormat("@yPoint");
    //courbe->setPointLabelsVisible(true);
    courbe->attachAxis(axeY);
```

```
// Le widget
graphique = new QChartView(graphe);
graphique->setRenderHint(QPainter::Antialiasing);

setCentralWidget(graphique);
resize(640, 480);
}
```


Pilotage écran

```
#include "pilotage_ecran.h"

#define SD_CS 4

// Use hardware SPI (on Uno, #13, #12, #11) and the above for CS/DC
Adafruit_ILI9341 tft = Adafruit_ILI9341(TFT_CS, TFT_DC);

void setupEcran() {
  tft.begin();
  tft.setRotation(3);
  tft.fillScreen(ILI9341_BLACK);
  affichageLiaison(false);
}

void affichageMesureEnCours() {
  String texteMesure = "Mesure en cours";

  tft.fillScreen(ILI9341_BLACK);
  tft.setCursor(0, 0);
  tft.setTextColor(ILI9341_WHITE);
  tft.setTextSize(2);
  tft.println(texteMesure);
  tft.setCursor(0, 0);
  for (int i = 0; i < 3; i++) {
    tft.println(texteMesure += ".");
    tft.setCursor(0, 0);
    delay(500);
  }
  tft.setCursor(0, 0);
  texteMesure = "Mesure en cours ";
}

void affichageMesures(String data) {
  tft.setRotation(4);
  tft.setCursor(0, 0);
  tft.fillScreen(ILI9341_BLACK);
  tft.setTextColor(ILI9341_WHITE);
  tft.setTextSize(2);
  tft.println(data);
}

void affichageStopManuel(bool arret_manuel) {
  tft.fillScreen(ILI9341_BLACK);
  tft.setRotation(4);
  tft.setTextColor(ILI9341_RED);
  tft.setTextSize(5);
  if (arret_manuel == true)
    tft.println("Arret du moteur en cours");
  else
    tft.println("Moteur a l'arret");
}
```

```
}
```

```
void affichageLiaison(bool is_open) {  
  String liaison;  
  if (is_open)  
    liaison = "Liaison    ouverte";  
  else  
    liaison = "Liaison    fermee";  
  tft.setRotation(1);  
  tft.fillScreen(ILI9341_BLACK);  
  tft.setCursor(0, 0);  
  tft.setTextColor(ILI9341_WHITE);  
  tft.setTextSize(4);  
  tft.println(liaison);  
}
```

pilotage moteur

```
#include "pilotage_moteur.h"

// brochage de la carte moteur pas à pas (A4988)
#define EN_A4988 22
#define MS1_A4988 24
#define MS2_A4988 26
#define MS3_A4988 28
#define RST_A4988 30
#define SLP_A4988 32
#define STEP_A4988 34
#define DIR_A4988 36

#define nb_pas 10

void setupSerialMoteur() {
  //Initialisation de la liaison série à 115200 bauds
  Serial.begin(115200);

  //DEFINITION DE TOUTES LES BROCHES EN SORTIE
  pinMode( EN_A4988, OUTPUT );
  pinMode( DIR_A4988 , OUTPUT );
  pinMode( STEP_A4988 , OUTPUT );
  pinMode( MS1_A4988, OUTPUT );
  pinMode( MS2_A4988 , OUTPUT );
  pinMode( MS3_A4988 , OUTPUT );
  pinMode( RST_A4988, OUTPUT );
  pinMode( SLP_A4988 , OUTPUT );
  pinMode(LED_BUILTIN, OUTPUT);

  //Initialisation des broches MS1, MS2 et MS3 à LOW = 1 pas entier
  digitalWrite( MS1_A4988, LOW );
  digitalWrite( MS2_A4988, LOW );
  digitalWrite( MS3_A4988, LOW );

  //Initialisation des broches ENABLE, RESET et SLEEP
  digitalWrite( EN_A4988, HIGH );
  digitalWrite( RST_A4988, LOW );
  digitalWrite( SLP_A4988, HIGH );
}

void avanceGauche() {

  digitalWrite( DIR_A4988 , LOW); // Direction GAUCHE

  // Initialisation des broches STEP et RESET
  digitalWrite( STEP_A4988 , LOW);
  digitalWrite( RST_A4988, HIGH );

  //Enable actif
  digitalWrite( EN_A4988, LOW );
```

```

// Avance de 30 pas = 1 cm

for (int i = 0; i < 10; i++) {
    digitalWrite( STEP_A4988, HIGH );
    delay( 20 );
    digitalWrite( STEP_A4988, LOW );
    delay( 20 );
}

// Broche ENABLE inactive
digitalWrite( EN_A4988, HIGH );

//Vérification visuelle de la bonne réception de l'ordre avec le clignotement de la DEL de la carte
digitalWrite(LED_BUILTIN, digitalRead(LED_BUILTIN) ^ 1);

}

void avanceDroite() {

    digitalWrite( DIR_A4988 , HIGH); // Direction DROITE

    // Initialisation des broches STEP et RESET
    digitalWrite( STEP_A4988 , LOW);
    digitalWrite( RST_A4988, HIGH );

    //Enable actif
    digitalWrite( EN_A4988, LOW );

    // Avance de 30 pas = 1 cm
    for ( int i = 0; i < 10; i++) {
        digitalWrite( STEP_A4988, HIGH );
        delay( 20 );
        digitalWrite( STEP_A4988, LOW );
        delay( 20 );
    }

    // Broche ENABLE inactive
    digitalWrite( EN_A4988, HIGH );

    //Vérification visuelle de la bonne réception de l'ordre avec le clignotement de la DEL de la carte
    digitalWrite(LED_BUILTIN, digitalRead(LED_BUILTIN) ^ 1);
}

void arretMoteur() {
    // Broche ENABLE inactive
    digitalWrite( EN_A4988, HIGH );
}

```

DATABASE

MAINWINDOW

```
#include "mainwindow.h"
```

```
#include "ui_mainwindow.h"  
#include <QMessageBox>  
#include <QDebug>  
#include <QScrollBar>  
#include <QSqlField>  
#include <QSqlRecord>  
#include <QSqlQuery>  
#include <QSqlError>
```

```
MainWindow::MainWindow(QWidget *parent) :  
    QMainWindow(parent),  
    ui(new Ui::MainWindow)  
{  
    ui->setupUi(this);  
    queryModel=0;  
    tableModel= 0;  
    //ui->mainToolBar->insertWidget(0,&selNom);  
    db = QSqlDatabase::addDatabase("QMYSQL", "MaDB");  
    db.setDatabaseName("Kundt");  
    db.setHostName("172.22.16.9");  
    db.setUserName("Kundt");  
    db.setPassword("Tube");  
    if(!db.open())  
        QMessageBox::critical(this, "Attention", "Pb d'accès 1", QMessageBox::Ok);  
    else  
        qDebug()<<"MaBase ouverte";  
  
    this->on_actionRafaichir_triggered();  
}
```

```
MainWindow::~MainWindow()  
{  
    delete ui;  
    delete tableModel;  
    if(db.isOpen())  
        db.close();  
}
```

```
void MainWindow::paintEvent()  
{  
    int w= ui->tableView->width()-ui->tableView->verticalHeader()->width()-ui->tableView->verticalScrollBar()->width();  
    ui->tableView->setColumnWidth(0,w/4);  
    ui->tableView->setColumnWidth(1,w/4);  
    ui->tableView->setColumnWidth(2,w/4);  
    ui->tableView->setColumnWidth(3,w/4);  
}
```

```
void MainWindow::on_actionRafaichir_triggered()  
{  
    if(selNom.text().isEmpty()){  
        if(tableModel)  
            delete tableModel;  
        tableModel= new QSqlTableModel(this, db);  
        tableModel->setTable(table_selectionnee);  
        tableModel->setEditStrategy(QSqlTableModel::OnManualSubmit);  
    }
```

```

        ui->tableView->hideColumn(0);
        if(tableModel->select())
        {
            ui->tableView->setModel(tableModel);
        }
    }
else
{
    if(queryModel)
        delete queryModel;
    queryModel= new QSqlQueryModel(this);
    queryModel->setQuery("SELECT * FROM login WHERE user LIKE
    '"+selNom.text()+"%' ",db);
    ui->tableView->setModel(queryModel);
}
ui->lineEditCoef->clear();
}

void MainWindow::on_actionModifier_triggered()
{
    int ligne = ui->tableView->currentIndex().row();
    if(ligne < 0) return;
    QSqlRecord sqlRecord = tableModel->record(ligne);
    QString id = sqlRecord.field("id_enregistrement").value().toString();
    QString coef_abs = sqlRecord.field("absorption").value().toString();
    QString frequence = sqlRecord.field("frequence").value().toString();
    id_materiau = sqlRecord.field("id_materiau").value().toString();

    QSqlQuery query(db);
    query.prepare(
        "UPDATE enregistrement SET frequence = " + frequence + " ,
absorption = " + coef_abs
        + " , id_materiau = " + id_materiau + " WHERE id_enregistrement
= " + id
    );
    if(!query.exec())
        QMessageBox::critical(this,"Attention","Pb Req",QMessageBox::Ok);
    qDebug() << id;
    qDebug() << query.lastError();
}

void MainWindow::on_AjouterMateriau_clicked()
{
    QString materiau = ui->lineEditNom->text();

    QSqlQuery query(db);
    query.prepare("INSERT INTO materiau (nom) VALUES " + materiau);
    if(!query.exec())
    {
        QString error = query.lastError().text();
        QMessageBox::critical(this,"Attention","Impossible d'ajouter le
matériau. ERREUR : " + error,QMessageBox::Ok);
    }
    qDebug() << query.lastError();
    ui->lineEditNom->clear();
    this->on_actionRafaichir_triggered();
}

//void MainWindow::on_actionCreerTableMateriau_triggered()
//{
//    QString table = ui->lineEditNom->text();

```

```

//      QSqlQuery query(db);
//      query.prepare("CREATE TABLE `kundt`." + table + " ( `id_freq` INT(50) NOT
//      NULL AUTO_INCREMENT , `frequence` INT(50) NOT NULL , `coefficientAbsorption`
//      DOUBLE NOT NULL , PRIMARY KEY (`id_freq`), UNIQUE `UNIQUE` (`frequence`)) ENGINE
//      = MyISAM;");
//      if(!query.exec())
//      QMessageBox::critical(this,"Attention","Pb Req",QMessageBox::Ok);
//      this->on_actionRafaichir_triggered();
//}

//void MainWindow::on_actionSupprimerTable_triggered()
//{
//      QString table = ui->lineEditNom->text();

//      QSqlQuery query(db);
//      query.prepare("DROP TABLE " + table);
//      if(!query.exec())
//      QMessageBox::critical(this,"Attention","Pb Req",QMessageBox::Ok);

//      this->on_actionRafaichir_triggered();
//}

void MainWindow::on_spinBoxMateriau_valueChanged(const QString &arg1)
{
    id_materiau = arg1;
}

void MainWindow::on_actionafficherMateriaux_triggered()
{
    if(selNom.text().isEmpty()){
        if(tableModel)
            delete tableModel;
        tableModel= new QSqlTableModel(this,db);
        tableModel->setTable("materiau");
        tableModel->setEditStrategy(QSqlTableModel::OnManualSubmit);
        ui->tableView->hideColumn(0);
        if(tableModel->select())
        {
            ui->tableView->setModel(tableModel);
        }
    }
    else
    {
        if(queryModel)
            delete queryModel;
        queryModel= new QSqlQueryModel(this);
        queryModel->setQuery("SELECT * FROM login WHERE user LIKE
""+selNom.text()+"%",db);
        ui->tableView->setModel(queryModel);
    }
    ui->lineEditCoef->clear();
}

void MainWindow::on_actionafficherMesures_triggered()
{
    if(selNom.text().isEmpty()){
        if(tableModel)
            delete tableModel;
        tableModel= new QSqlTableModel(this,db);
        tableModel->setTable("enregistrement");
        tableModel->setEditStrategy(QSqlTableModel::OnManualSubmit);
        ui->tableView->hideColumn(0);
    }
}

```

```

        if(tableModel->select())
        {
            ui->tableView->setModel(tableModel);
        }
    }
else
{
    if(queryModel)
        delete queryModel;
    queryModel= new QSqlQueryModel(this);
    queryModel->setQuery("SELECT * FROM login WHERE user LIKE
    '"+selNom.text()+"'",db);
    ui->tableView->setModel(queryModel);
}
ui->lineEditCoef->clear();

}

void MainWindow::on_comboBoxTableARafraichir_currentTextChanged(const QString
&arg1)
{
    table_selectionnee = arg1.toLower();
    this->on_actionRafaichir_triggered();
}

void MainWindow::on_pushButtonSupprimerMateriau_clicked()
{
    QSqlQuery query(db);
    int ligne = ui->tableView->currentIndex().row();
    if(ligne < 0) return;
    QSqlRecord sqlRecord= tableModel->record(ligne);
    QString id = sqlRecord.field("id_materiau").value().toString();

    query.prepare("DELETE FROM materiau WHERE materiau.id_materiau = " + id);
    if(!query.exec())
    {
        QString error = query.lastError().text();
        QMessageBox::critical(this,"Attention","Impossible d'effacer le
matériau. ERREUR : " + error,QMessageBox::Ok);
    }

    this->on_actionRafaichir_triggered();
}

void MainWindow::on_actionafficher_1_ID_triggered()
{
    ui->tableView->showColumn(0);
}

void MainWindow::on_pushButtonAjouterMesures_clicked()
{
    double coef_abs = ui->lineEditCoef->text().toDouble();
    if((0 >= coef_abs) || (1 <= coef_abs) )
        QMessageBox::critical(this,"Attention","Le coefficient doit être compris
entre 0 et 1",QMessageBox::Ok);
    else
    {
        QSqlQuery query(db);
        query.prepare("INSERT INTO enregistrement (absorption, frequence,
id_materiau) VALUES (" + QString::number(coef_abs) + ", " + frequence + ", " +
id_materiau + ")");
        if(!query.exec())
    }

```



```

        QMessageBox::critical(this, "Attention", "ERREUR : " +
query.lastError().text(), QMessageBox::Ok);
        qDebug() << query.lastError();

        this->on_actionRafaichir_triggered();
    }

}

void MainWindow::on_pushButtonSupprimerMesures_clicked()
{
    QSqlQuery query(db);
    int ligne = ui->tableView->currentIndex().row();
    if(ligne < 0) return;
    QSqlRecord sqlRecord= tableModel->record(ligne);
    QString id = sqlRecord.field("id_enregistrement").value().toString();

    query.prepare("DELETE FROM enregistrement WHERE
enregistrement.id_enregistrement = " + id);
    query.exec();

    this->on_actionRafaichir_triggered();
}

void MainWindow::on_comboBoxFrequence_currentTextChanged(const QString &arg1)
{
    frequence = arg1;
}

```

DIALOG LOGIN

```
#include "dialoglogin.h"
#include "ui_dialoglogin.h"
#include <QMessageBox>
#include <QSqlDatabase>
#include <QDebug>
#include <QSqlQuery>

DialogLogin::DialogLogin(QWidget *parent) :
    QDialog(parent),
    ui(new Ui::DialogLogin)
{
    ui->setupUi(this);
}

DialogLogin::~DialogLogin()
{
    delete ui;
}

void DialogLogin::on_buttonBox_accepted()
{
    int accepted = QDialog::Rejected;
    {
        QSqlDatabase db= QSqlDatabase::addDatabase("QMYSQL","login");
        db.setDatabaseName("Kundt");
        //db.setPort(3306);
        db.setHostName("172.22.16.9");
        db.setUserName("Kundt");
        db.setPassword("Tube");
        if(db.open())
        { qDebug()<<"MaBase ouverte";
          QString user = ui->lineEditUser->text(); //lecture du lineEdit user
          QString password = ui->lineEditPassword->text(); //lecture du
lineEdit password
          QSqlQuery query(db);
          if(!query.exec("SELECT password,id_login FROM login WHERE user='" +
user + "'"))
              QMessageBox::critical(this,"Attention","Pb
Req",QMessageBox::Ok);
          else
          { while(query.next())
            {
                qDebug()<<query.value(0);
                if(query.value(0).toString() == password)
                {
                    accepted = QDialog::Accepted;
                    break;
                }
            }
          }
          db.close();
        }
        else
        {
            QMessageBox::critical(this,"Attention","Pb d'accès
2",QMessageBox::Ok);
        }
        QSqlDatabase::removeDatabase("Login");
        done(accepted);
    }
}
```

MAIN

```
#include "dialoglogin.h"

#include "mainwindow.h"
#include <QApplication>
#include <QMessageBox>

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    DialogLogin dlg; // Création de la boîte de dialogue
    if(dlg.exec()) // affichage
    {
        MainWindow w; // Création de la fenêtre principale
        w.show(); // Affichage
        return a.exec();// exécution de l'application
    }
    else
        QMessageBox::critical(0,"Erreur","Problème d'authentification");
    return 1;
}
```