**3.3 Conception avant-projet :**

Solution...

En parlant d’outil, avant la conception de ce projet, on avait besoin de Machine (c’est logique puique on parle d’une conception d’application), le problem c’est q’on avait a notre disposition que 2 Machines disponibles du coup on l’a fait avec ce qu’on avait.  
Une application pour la Gestion des Matériels Informatiques, on nous a donner a l’avance l’outil pour concevoir cette application qui est le Langage de programmation C++ avec le Framework Qt Creator dont on va vous parler juste après ; certe il existe d’autre Langage de programmation Orienté Objet comme Python, JavaScript, Java, Ruby, etc... , mais C++ a aussi sa propre avantage dont a va vous donner sous forme de Tableau de comparaison ci-dessous.

|  |  |  |  |
| --- | --- | --- | --- |
| Langage de Programmmation Orienté Objets | Caractéristiques | Avantages | Faiblesses |
| **C++** | - Orienté Objet, Procédural, Générique - Gestion mémoire fléxible - Performance ***Très élevée***  - Compatibilité avec C  - Populaire (Jeux, système, …). | - Large choix de bibliothèques et frameworks  - Très polyvalent : peut être utilisé pour des applications bas niveau et haut niveau  - Performances élevées, idéal pour les applications de haut niveau. | - Complexité accrue en raison de la gestion manuelle de la mémoire  - Erreurs difficiles à déboguer (fuites de mémoire)  - Longtemps d’apprentissage pour bien maîtriser tous les aspects. |
| Java | - Orienté Objet  - Gestion du mémoire garbage Collection  - Performance élevée, mais moins rapide que C++  - Populaire en Entreprise. | - Portabilité élevée grâce à la JVM (Java Virtual Machine)  - Support complet pour la programmation orientée objet  - Bonnes performances, même si moins rapide que C++ | - Moins performant que C++ dans les applications nécessitant des ressources intensives  - Syntaxe plus verbale, rendant le code parfois lourd  - Pas optimal pour les scripts rapides ou les petits projets. |
| Python | - Multiparadigme (Objet, Fonctionnel, etc.)  - Gestion du mémoire garbage Collection  - Performance Modérée  - Très populaire, IA et science des données. | - Syntaxe simple et facile à apprendre - Large communauté et écosystème pour la science des données, le web et l'IA  - Grand nombre de bibliothèques pour des tâches variées (web, IA, science des données). | - Moins performant, surtout pour les applications exigeant beaucoup de calcul - Gestion dynamique des types peut entraîner des erreurs au moment de l'exécution - Pas idéal pour le développement de systèmes bas niveau. |

D’après le Tableau de comparaison ci-dessus, le choix du langage était fait puiqu’on s’est déjà familiariser avec C et on a besoin de polyvalence et de performance alors va s’y pour C++.

Parlons un peut de ce fameux Langage.

Le **C++** est un langage de programmation de bas niveau, puissant et polyvalent, créé par **Bjarne Stroustrup** dans les années 1980. Il est considéré comme une extension du C, intégrant des fonctionnalités orientées objet comme les classes et l’héritage, tout en permettant une gestion fine des ressources et de la mémoire. C++ est particulièrement apprécié pour les **applications nécessitant des performances élevées**, comme les jeux vidéo, les systèmes embarqués, et les logiciels de calcul scientifique. Sa compatibilité avec le langage C permet de réutiliser les bibliothèques C tout en exploitant les concepts modernes.

En résumé, C++ est un choix incontournable pour les projets où la **vitesse** et le **contrôle bas niveau** sont primordiaux, bien qu’il nécessite une gestion rigoureuse de la mémoire et des erreurs pour éviter les fuites et les bogues, ce que je trouve être une faiblesse pour C++.

CHAPITRE 4: CONCEPTION

**4.1. Déscription de la base de données (MCD, MLD)**

Le Modèle Conceptuel de Donnée (MLD) et le Modèle Logique de donnée (MLD) pour la Gestion de Matériel Informatiques présente les données dont on va se servire durant pour que l’**application** fait ce qu’on attend de lui.

La base de donnée a été conçue pour répondre aux besoins suivants :

- Gestion des matériels informatiques par les Utilisateurs

- Gestion des intervenants par l’Utilisateurs

- Gestion des entretients de matériels informatiques par l’Utilisateurs

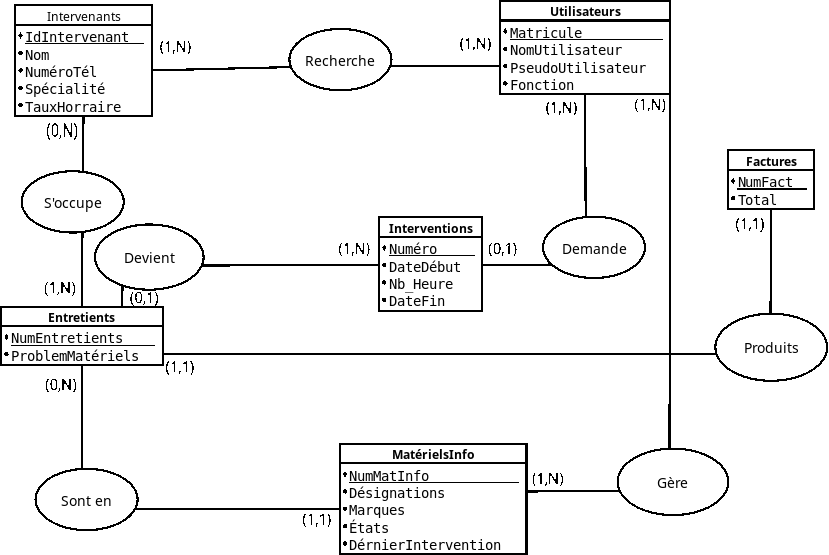
- Visualisation de chaque intervention sur chaque matériel Informatique

- Et Édition de l’état des préstations.

Dans la première section, le **MCD** fournit une vue d’ensemble des entités et relations principales sans se préoccuper des contraintes techniques. La section suivante présente le **MLD** qui va traduire le MCD en une structure de tables et de relations, en tenant compte des spécificités techniques requises pour la base de données.

Pour bien structurer une application, que ce soit de type DESKTOP, WEB ou autres, il est primordiale d’avoir une base de donnée bien structurer et bien définit. On a suivi ces étapes pour la création du MCD de notre application :

Premièrement, rassembler en vrac tous ce qu’on peut avoir comme donnée et ce qu’on peut utiliser ; Ensuite, rassembler celles qui peuvent se rassembler, qui ont une même **Entité** comme par exemple pour l’entité « Utilisateurs » elle peut regrouper le Nom, Matricule, et la Fonction de l’Utilisateur dans l’entreprise ; après ça, crée les tables pour chaque entité et y mettre leur **rubriques**, mettre les **cardinalités** qui varie en fonction de la relation qui existe entre les deux entités (il y a la relation père – fils qui se traduit par (1,N) – (1,1), père – père ou (1,N) – (1,N) et la relation (1,1) – (1,1)) ; Et enfin, mettre le verbe qui lie les deux ou plusieurs Entités.

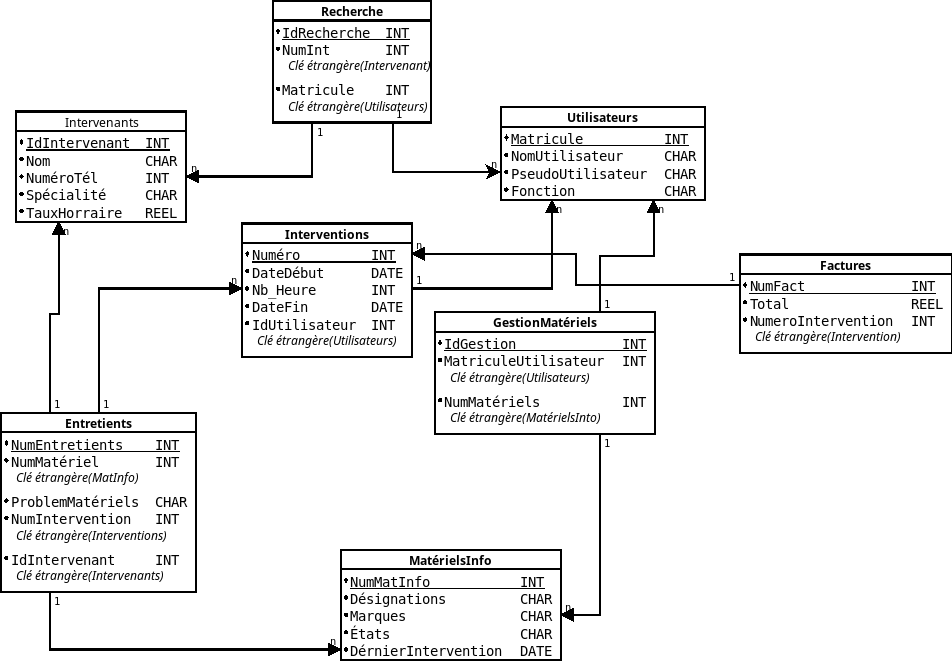
L’image suivant résume tous ce qu’on vient de dire.

Après avoir fini le MCD, passons au MLD ou Modèle Logique de donnée. Elle nous permet d’avoir une vue plus simple de notre base de donnée pour pouvoir ensuite les entrées dans **SQLite** qui est la base de donnée qu’on va utiliser pour la conception de cette application en raison de sa simplicité.

Pour pouvoir passer du MCD au MLD il va falloir suivre certaint règle :

Premièrement, chaque Entités devienent des **Tables**, les Rubriques devienent des **Attributs,** mettre les **contraintes** (unicité, non-nul, etc.), chaque Tables a une **Clé primaire** qui identifie de manière unique chaque enregistrement de la table ; Après, mettre les **Clé étrangères,** c’est la relation qui lie les Tables entre elles ; A noté que chaque relation ont leurs propre caractéristique, pour la relation (1,N) – (1,N) : le verbe qui lie les deux tables deviennent une **table** qui prend les clé primaires des deux tables comme **clé étrangère,** la relation (1,N) – (1,1) : celle de gauche est le père et la clé primaire du père devient une **clé étrangère** pour le fils et la relation (1,1) – (1,1) : la clé primaire de l’une des deux tables devient une clé étrangère pour l’une des deux tables. Enfin, on peut mettre pour chaque attributs une type pour dire s’il est de type entier ou charactère, générallement c’est le passage MPD ou Modèle Physique de Donnée mais je le met ici quand même.

Resumons tous ça avec l’image suivante



**4.2. Déscription des traitements (MCT, MOT)**

// Pas encore fait…

PARTIE III. RÉALISATION

CHAPITRE 5 : MISE EN PLACE DE L’ENVIRONNEMENT DE DÉVELOPPEMENT

**5.1. Installation des Outils**

Comme dans tous métiers quand on va travailler il nous faut des Outils de travaille.

Ici on va concevoir une Application pour le gestion de Matériels Informatiques, quand on parle de concevoir une application on parle de langage de programmation, et quand on parle de langage de programmation, on parle d’ordinateur. Tous d’abord, on a parlé du langage C++ précedemment et on a dis qu’il est très puissant et robuste, logiquement pour qu’il fonctionne normallement on aura besoin d’une machine capable de Compiler les codes qu’on va mettre dedans sans qu’il ram trop.

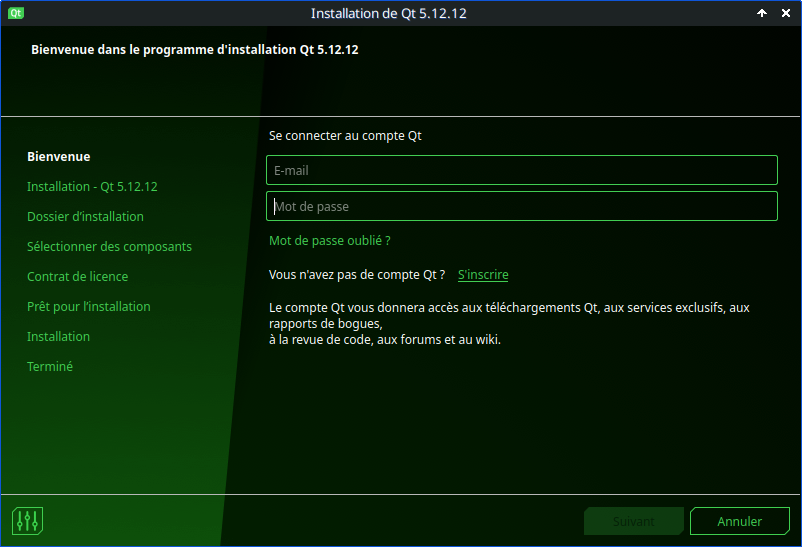
Pour avoir une expérience fluide, il est recommander d’avoir au minimum une machine qui a une processeur corei5 avec 4go de ram(petit projet), une disque dur SSD serais meilleurs pour une vitesse de compilation agréable.

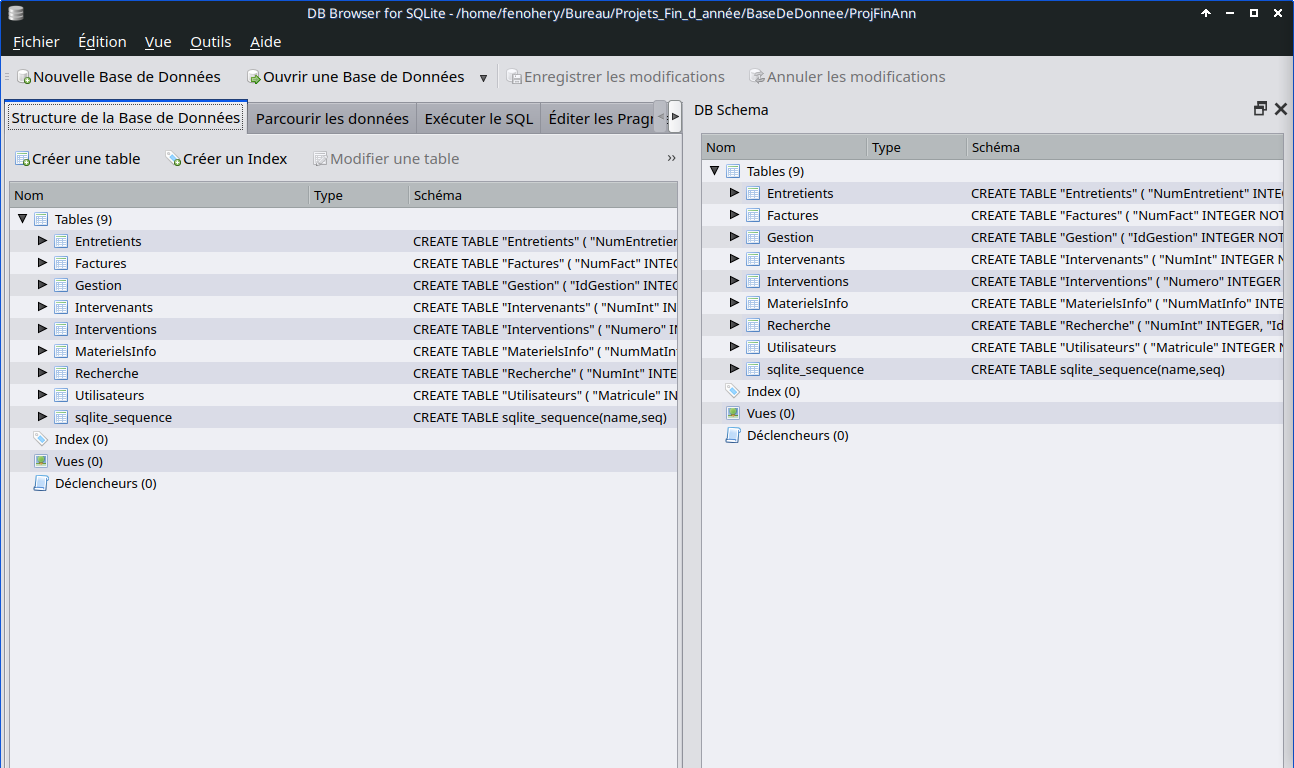
Après, pour la conception d’une application avec Qt Creator il faudrais avoir le logiciel, il est obligatoire d’avoir un compte si on veux installer Qt Creator en ligne (fig 5.1.1) mais il existe aussi en version offline. On a parler de base de donnée tout a l’heure alors on aura aussi besoin d’une outils pour crée notre base de donnée, ici j’ai **DB Browser for SQLite (fig 5.1.2)** comme base de donnée mais vous avez le choix entre SQL ou autre ; vous pouvez aussi crée votre base de donnée via des ligne de commande dans une **Terminale** comme par exemple pour SQL :

CREATE DATABASE utilisateurs;

USE utilisateurs;

CREATE TABLE utilisateur ( matricule INT PRIMARY KEY, nom VARCHAR(50) NOT NULL, pseudo VARCHAR(50), fonction VARCHAR(50) );

fig 5.1.1

fig 5.1.2

Une fois ces outils installer, on poura commencer notre conception.

**5.2. Configuration des Outils**

Une fois les Outils installer comme il faut, configurons tous ça pour qu’il dégub notre code C++.

Une fois Qt installée en ligne ou offline, ouvrer le et nous allons crée notre Projet.

Dans l’interface de Qt il y a deux boutton nouveau et ouvrir, on va crée notre projet alors on va dans « nouveau » une fois dedans Qt nous demande de choisir un **modèle** entre « Qt widget Application » et « Qt Console Application » qui est une application concole, notre but et de Concevoir une application de gestion de matériel Informatique alors on va y aller dans Qt widget Application. Celle fait, il nous demande d’entrer un nom pour le Projet, pour le notre « projet » , et sa localisation dans notre dossier ; Ensuite dans la prochaine fenêtre choisir notre « **Build system »** qui sert a automatiser la compilation du code, après ça, il demande de donner l’information sur notre **class,** le nom, la class de base dont d’autre class pourons hérité après ; Après, choisir si on veux traduire la langue qu’on utilise ou non, et après ça choisir le **kit** qu’on va utiliser qui est très important, elle varie selons la version de Qt, car c’est elle qui va **déboguer** notre code ; Et enfin, il nous donne une résumer de tous ce qu’on a fait et faire une retour s’il y a quelque chose a modifier.

Tous celle fait, on devrais être près a commencer a coder.

Notre projet est contenu dans un dossier qui est nommer avec le nom qu’on a donner au départ, il est configurer comme suit :

Il y a le **fichier.pro** qui va contenir tous notre projet et qui ne peut être ouvert que par Qt, pour ce projet de conception d’application on aura besoin d’une base de donnée, ici SQLite, alors il faut ajouter la ligne **« Qt += sql »** dans le fichier.pro si on veux que notre Application intéragis bien avec notre base de donnée. Après, il les **fichiers.h** qui sont les fichiers d’en-tête où serons déclarer les variables qu’on va utiliser par la suite, et aussi c’est là dedans qu’on va déclarer notre code pour intéragir avec notre base de donnée :

Dans connectbdd.h :

#ifndef CONNECTBDD\_H

#define CONNECTBDD\_H

#include <QSqlDatabase>

#include <QDebug>

#include <QSqlDriver>

#include <QSqlQuery>

#include <QtSql>

#include <QSql>

#include <QSqlError>

class connectbdd

{

public:

QSqlDatabase bdd;

void mihidybdd() {

bdd.close();

qDebug() << "Mihidy donnee zao...";

bdd.removeDatabase(QSqlDatabase::defaultConnection);

};

bool mivohabdd() {

// bdd = QSqlDatabase:: addDatabase("QSQLITE", "maConnection");

bdd = QSqlDatabase:: addDatabase("QSQLITE");

bdd.setDatabaseName("/home/fenohery/Bureau/Projets\_Fin\_d\_année/BaseDeDonnee/ProjFinAnn");

QStringList drivers = QSqlDatabase::drivers();

if (!bdd.open()) {

qDebug() << "TSY MIVOHA BDD";

return false;

} else {

qDebug() << "drivers dispo" << drivers;

return true;

}

}

};

#endif // CONNECTBDD\_H

On va juste utiliser ce fichier pour ouvrir et fermer la base de donnée. Après il y a les **fichier.cpp** où serons coder les algorithmes et fonction dont on aura besoin. Ensuite il y a le **fichier.ui** dans forms où on pourra faire des designs comme on le veux ; et Enfin il y a les **ressources** où serons stoker les images, logo et tous les images dont on aura besoin.

La configuration de notre base sera fait dans DB Browser for SQLite, il y a pas vraiment grand-chose à configurer là dedans, juste suivre ce qu’on a fait dans le MLD.

**CHAPITRE 6 : DÉVELOPPEMENT DE L’APPLICATION**

6.1. CRÉATION DE LA BASE DE DONNÉE

Pour bien structurer le projet on aura besoin de 6 tables qui deviendra 8 dans le MLD dis précedemment, tous les variables de type entier serons typé par INT, les caractères TEXTE et les réels REEL.

La Table Utilisateurs qui va contenir le matricule, le pseudo et le Nom de l’utilisateur qui vont nous servir après dans l’application.

CREATE TABLE "Utilisateurs" (

"Matricule" INTEGER NOT NULL,

"NomUtilisateur" TEXT NOT NULL,

"PseudoUtilisateur" TEXT NOT NULL,

"Fonction" TEXT NOT NULL,

PRIMARY KEY("Matricule" AUTOINCREMENT)

);

La Tables MatérielsInformatique qui va contenir les matériels que l’utilisateur ajoutera.

CREATE TABLE "MaterielsInfo" (

"NumMatInfo" INTEGER NOT NULL,

"Designations" TEXT,

"Marques" TEXT,

"Etats" TEXT,

"DernierIntervention" TEXT,

PRIMARY KEY("NumMatInfo" AUTOINCREMENT)

);

La Table Intervenant qui va contenir la liste d’intervenant dont l’utilisateur ou l’entreprise disposera.

CREATE TABLE "Intervenants" (

"NumInt" INTEGER NOT NULL,

"Nom" TEXT,

"NumeroTel" TEXT UNIQUE,

"Specialité" TEXT,

"TauxHorraire" REAL,

PRIMARY KEY("NumInt" AUTOINCREMENT)

);

La Table Entretients qui réunira la liste de matériels en entretient par un Intervenants.

CREATE TABLE "Entretients" (

"NumEntretient" INTEGER NOT NULL,

"NumMateriel" INTEGER,

"ProblemMateriel" TEXT,

"NumeroIntervention" INTEGER,

"IdIntervenant" INTEGER,

FOREIGN KEY("NumMateriel") REFERENCES "MaterielsInfo"("NumMatInfo"),

FOREIGN KEY("NumeroIntervention") REFERENCES "Interventions"("Numero"),

FOREIGN KEY("IdIntervenant") REFERENCES "Intervenants"("NumInt"),

PRIMARY KEY("NumEntretient" AUTOINCREMENT)

);

La Table Facture qui va contenir le « total à payer » en fonction de l’heure et du taux Horraire de l’Intervenants.

CREATE TABLE "Factures" (

"NumFact" INTEGER NOT NULL,

"Total\_a\_payer" REAL,

"NumEntervention" INTEGER UNIQUE,

FOREIGN KEY("NumEntervention") REFERENCES "Interventions"("Numero"),

PRIMARY KEY("NumFact" AUTOINCREMENT)

);

La Table Gestion qui contenira la liste de matériels avec le Matricule de l’Utilisateur dont on poura voir les détails après.

CREATE TABLE "Gestion" (

"IdGestion" INTEGER NOT NULL,

"MatriculeUtilisateur" INTEGER,

"IdMatInfo" INTEGER,

FOREIGN KEY("MatriculeUtilisateur") REFERENCES "Utilisateurs"("Matricule"),

FOREIGN KEY("IdMatInfo") REFERENCES "MaterielsInfo"("NumMatInfo"),

PRIMARY KEY("IdGestion" AUTOINCREMENT)

);

La Table Interventions qui contient des données de l’entretient de chaque Matériels.

CREATE TABLE "Interventions" (

"Numero" INTEGER NOT NULL,

"DateDebut" INTEGER,

"Nb\_heure" INTEGER,

"DateFin" INTEGER,

"IdUtilisateur" INTEGER,

PRIMARY KEY("Numero" AUTOINCREMENT),

FOREIGN KEY("IdUtilisateur") REFERENCES "Utilisateurs"("Matricule")

);

La Table Recherche qui va contenir les données que l’utilisateur tapera quand il va chercher des Intervenants.

CREATE TABLE "Recherche" (

"NumInt" INTEGER,

"IdUtilisateur" INTEGER,

FOREIGN KEY("NumInt") REFERENCES "Intervenants"("NumInt"),

FOREIGN KEY("IdUtilisateur") REFERENCES "Utilisateurs"("Matricule")

);

**6.2. CODAGE DE L’APPLICATION**

Au départ, ça nous a pris des jours pour comprendre le fonctionnement de certaint code dans l’Application (ce qui a aussi retarder certaint chose car on a commencer avec aucune connaissance en Qt et C++).

Tous d’abord, on a commencer par faire bosser sur l’interface Utilisateurs dont il y aura des LineEdit pour le Nom, Le Pseudo que l’Utilisateur choisira et le Fonction qui désignera le fonction de l’utilisateur dans l’entreprise ou il travaille.

Dans le **mainwindow.h** va y avoir les déclarations des variables, des fonctions et les différents **slots** que notre Application va utiliser.

#ifndef MAINWINDOW\_H

#define MAINWINDOW\_H

#include "stylesheet.h"

#include <QMainWindow>

#include <QtSql>

#include <QSql>

QT\_BEGIN\_NAMESPACE

namespace Ui { class MainWindow; }

QT\_END\_NAMESPACE

class MainWindow : public QMainWindow

{

Q\_OBJECT

public:

MainWindow(QWidget \*parent = nullptr);

// QSqlDatabase bdd = QSqlDatabase:: database("MaConnexion");

// connectbdd conn;

stylesheet styleUtil;

QString NomUtil;

QString PseudoUtil;

int MatriculeUtil;

QString FonctionUtil;

QString nomselectionner;

// QString pseudoselectionner;

~*MainWindow*();

private slots:

void on\_ANNULER\_clicked();

void on\_VALIDER\_clicked();

void on\_EditNom\_textChanged(const QString &arg1);

private:

Ui::MainWindow \*ui;

};

#endif // MAINWINDOW\_H

Ces Fonctions et slots vont être utiliser après dans **mainwindow.cpp** je vais juste donnée les codes pour les bouttons VALIDER car ça prend trop de place.

void MainWindow:: on\_VALIDER\_clicked()

{

// declaration variable champ de text

NomUtil = ui->EditNom->text();

PseudoUtil = ui->EditPseudo->text();

FonctionUtil = ui->EditFonction->text();

if (NomUtil.isEmpty()|| FonctionUtil.isEmpty() || PseudoUtil.isEmpty()) {

QMessageBox::warning(this, "Avertissement", "VEUILLEZ REMPLIR SVP...");

qDebug() << "fenoy oh...";

// connbdd.mihidybdd();

} else {

connectbdd connbdd;

connbdd.mivohabdd();

QSqlQuery query;

query.prepare("SELECT Matricule,PseudoUtilisateur FROM Utilisateurs WHERE PseudoUtilisateur=:pseudo");

query.bindValue(":pseudo", PseudoUtil);

// qDebug() << query.value(0);

// qDebug() << query.value(1);

if (query.exec() && query.first()) {

int matriculeUtilisateur = query.value(0).toInt(); // matricule SÉLÉCTIONNER

qDebug() << matriculeUtilisateur;

if (query.value(1).toString() > 0) {

// QMessageBox::critical(this, "", "PSEUDO EXISTANT VEUILLEZ SAISIR UN AUTRE NOM UTILISATEUR");

QMessageBox mssbx;

mssbx.setText("PSEUDO EXISTANT \n ÊTES VOUS : "+PseudoUtil+" ?");

QPushButton \*Y = mssbx.addButton(QMessageBox::Yes);

QPushButton \*N = mssbx.addButton(QMessageBox::No);

mssbx.*exec*();

if (mssbx.clickedButton()==Y) {

ListMateriels \*listMat = new ListMateriels(matriculeUtilisateur);

this->close();

connbdd.mihidybdd();

listMat->*exec*();

} else if (mssbx.clickedButton()==N) {

cout << "ENTRER UN NOUVEAU PSEUDO !" << endl;

}

} else {

cout << "Y A RIEN ICI" << endl;

}

} else {

query.prepare("INSERT INTO Utilisateurs(NomUtilisateur,PseudoUtilisateur,Fonction) VALUES (:Nom,:Pseudo,:Fonction);");

query.bindValue(":Nom",NomUtil);

query.bindValue(":Pseudo", PseudoUtil);

query.bindValue(":Fonction",FonctionUtil);

qDebug() << query.exec();

MatriculeUtil = query.lastInsertId().toInt(); // définition du matricule pour le prendre après

if (!query.exec()) {

qDebug() << "Erreur lors de l'insertion :" << query.lastError();

} else {

qDebug() << "Utilisateurs tafiditra...";

connbdd.mihidybdd();

ListMateriels \*listMat = new ListMateriels(MatriculeUtil); // APPEL DU FENÊTRE "ListeMatériel" PARAMÉTRER

this->close();

listMat->setModal(true);

listMat->*exec*();

}

}

}

}

Après ça, on a entammé la partie page d’acceuil, ce qui va trouver dans la page Entretient et ce qu’on va avoir dans la page Intervenant. Dans la partie acceuil, l’Utilisateur verra la liste des matériels a sa disposition, il pourra ajouter, modifier et supprimer une Matériel comme il le veux, a chaque **ajout** le matériels sera accompagner par le Matricule de ce dernier dont tous les Utilisateurs pourons voir dans la partie **détails.**

Dans **listmateriels.cpp**:

**void ListMateriels::on\_ajoutMat\_clicked() // AJOUT matériels**

{

cout << "mampidira materiel.." << endl;

mat = new materiel(matriculeUtilisateurGlobal); // paramétrer "matriculeGlobal dans la fenêtre materiel

mat->setModal(true);

mat->*exec*();

afficheMateriels();

}

Dans **detailsmateriels.cpp :**

**DetailsMateriels::DetailsMateriels(int matriculeU, QWidget \*parent) :**

QDialog(parent),

ui(new Ui::DetailsMateriels)

{

ui->setupUi(this);

this->setWindowTitle("DETAILS DES MATÉRIELS");

resize(700,400);

this->matriculeUt = matriculeU; // récuperation du matricule de l'utilisateurs

affichedetailsMat();

}

On a parler d’ajout de matériel tout à l’heure ; A chaque Matériels ajouter, le Matricule de l’Utilisateur sera envoyer automatiquement dans la base de donnée, spécifiquement dans **Gestion.**

Dans **materiel.cpp**:

materiel::materiel(int matricule, QWidget \*parent) :

QDialog(parent),

ui(new Ui::materiel)

{

ui->setupUi(this);

this->setWindowTitle("MATERIEL");

this->matriculeMatGlobal = matricule; // Prise du valeur de "matricule" dans la fenêtre de début

}

void materiel::on\_VALIDER\_clicked()

{

connectbdd conxbdd;

conxbdd.mivohabdd();

QString marque = ui->marqueEdit->text();

QString designation = ui->designationEdit->currentText();

QString etat = ui->etatEdit->currentText();

QString dernierInt = ui->dernierEdit->text();

if (marque.isEmpty() || designation.isEmpty() || etat.isEmpty()) {

QMessageBox::warning(this, "attention", "Il semble qu'il y a des champ vide");

} else {

QSqlQuery matquery;

lastIdU = matquery.lastInsertId();

matquery.prepare("INSERT INTO MaterielsInfo(Designations, Marques, Etats, DernierIntervention) VALUES(:type, :marque, :etat, :dernierint);");

matquery.bindValue(":type", designation);

matquery.bindValue(":marque", marque);

matquery.bindValue(":etat", etat);

matquery.bindValue(":dernierint", dernierInt);

if (!matquery.exec()) {

qDebug() << "Erreur lors de l'insertion :" << matquery.lastError();

} else {

lastIdMateriel = matquery.lastInsertId(); // prend le dernier IdMatériel

qDebug() << lastIdMateriel;

qDebug() << "Insertion réussie dans MATÉRIELINFO OK";

QSqlQuery query;

query.prepare("INSERT INTO Gestion(MatriculeUtilisateur,IdMatInfo) VALUES(:IdU, :IdMat);");

qDebug() << matriculeMatGlobal;

query.bindValue(":IdU", matriculeMatGlobal);

query.bindValue(":IdMat", lastIdMateriel);

if (!query.exec()) {

qDebug() << "Erreur :" << "tsy mandeh le insert materiel";

qDebug() << "Erreur :" << query.lastError();

} else {

qDebug() << "Insertion dans GESTION parfait ";

cout << "Insertion dans Gestion parfait";

conxbdd.mihidybdd();

this->close();

}

}

}

}

L’Utilisateur aura le pouvoir de géré chaque matériels.

Dans **listmateriels.cpp :**

void ListMateriels::on\_modifMat\_clicked() // Modification Matériels

{

modifMat modifMat;

modifMat.getData(marque, type, etat, dernierint);

modifMat.Nbr = modifMat.getNb(Nb);

modifMat.*exec*();

afficheMateriels();

}

void ListMateriels::on\_supprimMat\_clicked() // SUPPRIMER UNE MATÉRIELE INFO

{

qDebug() << Nb;

QMessageBox mess;

mess.setText("SUPPRIMER !?");

QPushButton \*o = mess.addButton(QMessageBox::Yes);

QPushButton \*n = mess.addButton(QMessageBox::Cancel);

mess.*exec*();

if (mess.clickedButton()==o) {

connectbdd connbdd;

connbdd.mivohabdd();

QSqlQuery suppr;

suppr.prepare("DELETE FROM MaterielsInfo WHERE NumMatInfo=:nb;");

suppr.bindValue(":nb", Nb);

if (suppr.exec()) {

qDebug() << "ligne supprimer: " << Nb;

connbdd.mihidybdd();

afficheMateriels();

} else {

qDebug() << "Erreur de suppression.";

qDebug() << suppr.lastError();

}

} else if (mess.clickedButton()==n) {

cout << "Donnée non supprimer.." << endl;

}

}

Chaque modification sera envoyer et afficher dans la liste des Matériels.

Dans **modifmat.cpp :**

void modifMat::on\_modif\_clicked()

{

connectbdd bdd;

bdd.mivohabdd();

Marque = ui->marqueModif->text();

Type = ui->typeModif->currentText();

Etat = ui->etatModif->currentText();

DernInt = ui->intModif->text();

if (Marque.isEmpty() || Type.isEmpty() || Etat.isEmpty() || DernInt.isEmpty()) {

QMessageBox::warning(this, "attention", "Il semble qu'il y a des champ vide");

} else {

QSqlQuery matquery;

matquery.prepare("UPDATE MaterielsInfo SET Designations=:type,Marques=:marque,Etats=:etat,DernierIntervention=:dernierint WHERE NumMatInfo=:NumMat;");

matquery.bindValue(":marque", Marque);

matquery.bindValue(":type", Type);

matquery.bindValue(":etat", Etat);

matquery.bindValue(":dernierint", DernInt);

matquery.bindValue(":NumMat", Nbr); // "Nbr" est une variable prise dans listmateriel

if (!matquery.exec()) {

qDebug() << "Erreur lors du MAJ :" << matquery.lastError();

} else {

qDebug() << "Mise à jour parfait sur la ligne: " << Nbr;

}

bdd.mihidybdd();

this->close();

QMessageBox::warning(this, "Info", "Modification fait sur le materiel Numéro: "+Nbr);

}

}

De même dans la liste d’Intervention, l’Utilisateur aura le pouvoir de gérer la liste d’Intervention en cours **en fonction** de l’états du matériels.

Dans **listmateriels.cpp :**

void ListMateriels::afficheListeIntervention() // AFFICHE LA LISTE DE MATÉRIELS EN ENTRETIENTS

{

connectbdd connbdd;

connbdd.mivohabdd();

model = new QSqlQueryModel(ui->page\_3);

model->setQuery("SELECT Entretients.NumMateriel,ProblemMateriel,MaterielsInfo.Designations,Marques,Etats,Interventions.DateDebut,Nb\_heure,DateFin,Intervenants.Nom,NumeroTel,Entretients.NumEntretient "

"FROM Entretients INNER JOIN MaterielsInfo ON MaterielsInfo.NumMatInfo=Entretients.NumMateriel "

"INNER JOIN Interventions ON Interventions.Numero=Entretients.NumeroIntervention "

"INNER JOIN Intervenants ON Intervenants.NumInt=Entretients.IdIntervenant;"

);

model->*setHeaderData*(0,Qt::Horizontal,tr("N° Matériels"));

model->*setHeaderData*(1,Qt::Horizontal,tr("Problem"));

model->*setHeaderData*(2,Qt::Horizontal,tr("Désignation"));

model->*setHeaderData*(3,Qt::Horizontal,tr("Marques"));

model->*setHeaderData*(4,Qt::Horizontal,tr("États"));

model->*setHeaderData*(5,Qt::Horizontal,tr("Debut"));

model->*setHeaderData*(6,Qt::Horizontal,tr("Heure"));

model->*setHeaderData*(7,Qt::Horizontal,tr("Fin"));

model->*setHeaderData*(8,Qt::Horizontal,tr("NomRéparateurs"));

model->*setHeaderData*(9,Qt::Horizontal,tr("Téléphone"));

model->*setHeaderData*(10,Qt::Horizontal,tr("Entretient N°"));

ui->viewEntretient->*setModel*(model);

connbdd.mihidybdd();

}

void ListMateriels::on\_IntervenirMat\_clicked() // CLICK SUR le Boutton INTERVENIR

{

EntretientMateriels \*entretientMat = new EntretientMateriels(matriculeUtilisateurGlobal);

// Récuperation des données pour les mettres dans la fenêtre

entretientMat->getdonner(type,marque,etat,Nb);

connectbdd conbdd;

conbdd.mivohabdd();

QSqlQuery queryajout;

queryajout.prepare("SELECT Nom,NumeroTel,TauxHorraire,NumInt FROM Intervenants WHERE NumInt=?");

if (type=="CLAVIER" || type=="SOURIS") {

queryajout.bindValue(0,0);

if (queryajout.exec()) {

while (queryajout.next()) {

numIntervenant = queryajout.value(3).toInt();

QString Nom = queryajout.value(0).toString();

QString Num = queryajout.value(1).toString();

QString Taux = queryajout.value(2).toString();

entretientMat->getinfoEntretient(Nom, Num, Taux);

entretientMat->getIdIntervenant(numIntervenant);

}

} else { qDebug() << "Erreur execution requête: "<<queryajout.lastError().text(); }

} else if (type=="RESEAUX" || type=="PORTABLE") {

queryajout.bindValue(0,7);

if (queryajout.exec()) {

while (queryajout.next()) {

numIntervenant = queryajout.value(3).toInt();

QString Nom = queryajout.value(0).toString();

QString Num = queryajout.value(1).toString();

QString Taux = queryajout.value(2).toString();

entretientMat->getinfoEntretient(Nom, Num, Taux);

entretientMat->getIdIntervenant(numIntervenant);

}

} else { qDebug() << "Erreur execution requête: "<<queryajout.lastError().text(); }

} else if (type=="IMPRIMANTE" || type=="ALIMENTATION") {

queryajout.bindValue(0,8);

if (queryajout.exec()) {

while (queryajout.next()) {

numIntervenant = queryajout.value(3).toInt();

QString Nom = queryajout.value(0).toString();

QString Num = queryajout.value(1).toString();

QString Taux = queryajout.value(2).toString();

entretientMat->getinfoEntretient(Nom, Num, Taux);

entretientMat->getIdIntervenant(numIntervenant);

}

} else { qDebug() << "Erreur execution requête: "<<queryajout.lastError().text(); }

} else if (type=="ECRAN" || type=="UNITE CENTRALE") {

queryajout.bindValue(0,9);

if (queryajout.exec()) {

while (queryajout.next()) {

numIntervenant = queryajout.value(3).toInt();

QString Nom = queryajout.value(0).toString();

QString Num = queryajout.value(1).toString();

QString Taux = queryajout.value(2).toString();

entretientMat->getinfoEntretient(Nom, Num, Taux);

entretientMat->getIdIntervenant(numIntervenant);

}

} else { qDebug() << "Erreur execution requête: "<<queryajout.lastError().text(); }

} else {

queryajout.bindValue(0,1);

if (queryajout.exec()) {

while (queryajout.next()) {

numIntervenant = queryajout.value(3).toInt();

QString Nom = queryajout.value(0).toString();

QString Num = queryajout.value(1).toString();

QString Taux = queryajout.value(2).toString();

entretientMat->getinfoEntretient(Nom, Num, Taux);

entretientMat->getIdIntervenant(numIntervenant);

}

} else { qDebug() << "Erreur execution requête: "<<queryajout.lastError().text(); }

}

conbdd.mihidybdd();

entretientMat->*exec*();

afficheListeIntervention();

}

void ListMateriels::on\_supprimerMatEntretenu\_clicked() // SUPPRIMER MATÉRIELS ENTRETENUE

{

qDebug() << numEntretient;

QMessageBox mess;

mess.setText("SUPPRIMER !?");

QPushButton \*o = mess.addButton(QMessageBox::Yes);

QPushButton \*n = mess.addButton(QMessageBox::Cancel);

mess.*exec*();

if (mess.clickedButton()==o) {

connectbdd connbdd;

connbdd.mivohabdd();

QSqlQuery suppr;

suppr.prepare("DELETE FROM Entretients WHERE NumEntretient=:num");

suppr.bindValue(":num", numEntretient);

if (suppr.exec()) {

qDebug() << "ligne supprimer: " << numEntretient;

connbdd.mihidybdd();

afficheListeIntervention();

QMessageBox::information(this, "Information", "ligne "+numEntretient+" supprimer");

} else {

qDebug() << "Erreur de suppression.";

qDebug() << suppr.lastError();

}

} else if (mess.clickedButton()==n) {

cout << "Donnée non supprimer.." << endl;

}

}

Dans la page intervenant, l’Utilisateur pourra chercher un Intervenants à partir de son **nom** ou de sa **numéro** et affiche VIDE s’il n’y a pas d’occurence.

Dans **listmateriels.cpp :**

void ListMateriels::on\_zonederecherche\_textChanged(const QString &text)

{

connectbdd bdd;

bdd.mivohabdd();

QString queryString = QString(

"SELECT Nom, NumeroTel,Specialité,TauxHorraire "

"FROM Intervenants "

"WHERE Nom LIKE '%%1%' OR NumeroTel LIKE '%%1%'"

).arg(text);

model->setQuery(queryString);

if (model->*rowCount*() == 0) {

ui->viewIntervenantVide->setHidden(false);

ui->viewIntervenantVide->setText("VIDE");

ui->viewIntervenant->setHidden(true);

} else {

ui->viewIntervenantVide->setHidden(true);

ui->viewIntervenant->setHidden(false);

}

ui->viewIntervenant->*setModel*(model);

bdd.mihidybdd();

}

Dans la partie liste matériel Entretenu, on pourra voir le facture et éditer l’état des préstation en PDF.

Dans **facture.cpp :**

**void facture::on\_editionPDF\_clicked()**

{

connectbdd bddfact;

bddfact.mivohabdd();

generationPDF();

}

void facture::generationPDF()

{

QPdfWriter writer("facture\_2.pdf");

writer.setPageSize(QPageSize(QPageSize::A4));

writer.setResolution(300);

QPainter painter(&writer);

painter.setFont(QFont("Arial", 20)); // Définition du police

// Écrire le titre

painter.drawText(100, 100, "Facture matériels");

painter.drawText(100, 130, "Détails des interventions :");

QSqlQuery query;

query.prepare("SELECT Entretients.NumeroIntervention,IdIntervenant,Interventions.Nb\_heure,Intervenants.TauxHorraire "

"FROM Entretients INNER JOIN Interventions ON Entretients.NumeroIntervention=Interventions.Numero "

"INNER JOIN Intervenants ON Entretients.IdIntervenant=Intervenants.NumInt;");

if (query.exec()) {

int yPosition = 200; // Position verticale de départ

// En-tête de tableau

painter.drawText(100, yPosition, "Numéro d'Intervention");

painter.drawText(500, yPosition, "ID Intervenant");

painter.drawText(900, yPosition, "Heures");

painter.drawText(1300, yPosition, "Taux");

yPosition += 30; // Espacement après l'en-tête

// Récupérer et afficher les données de chaque ligne

while (query.next()) {

int numIntervention = query.value(0).toInt();

int idIntervenant = query.value(1).toInt();

int nbHeure = query.value(2).toInt();

double taux = query.value(3).toDouble();

painter.drawText(100, yPosition, QString::number(numIntervention));

painter.drawText(250, yPosition, QString::number(idIntervenant));

painter.drawText(400, yPosition, QString::number(nbHeure));

painter.drawText(500, yPosition, QString::number(taux));

yPosition += 20; // Avancer vers le bas pour la prochaine ligne

}

painter.end();

qDebug() << "PDF généré avec succès.";

this->close();

} else {

qDebug() << "Erreur lors de la récupération des données :" << query.lastError();

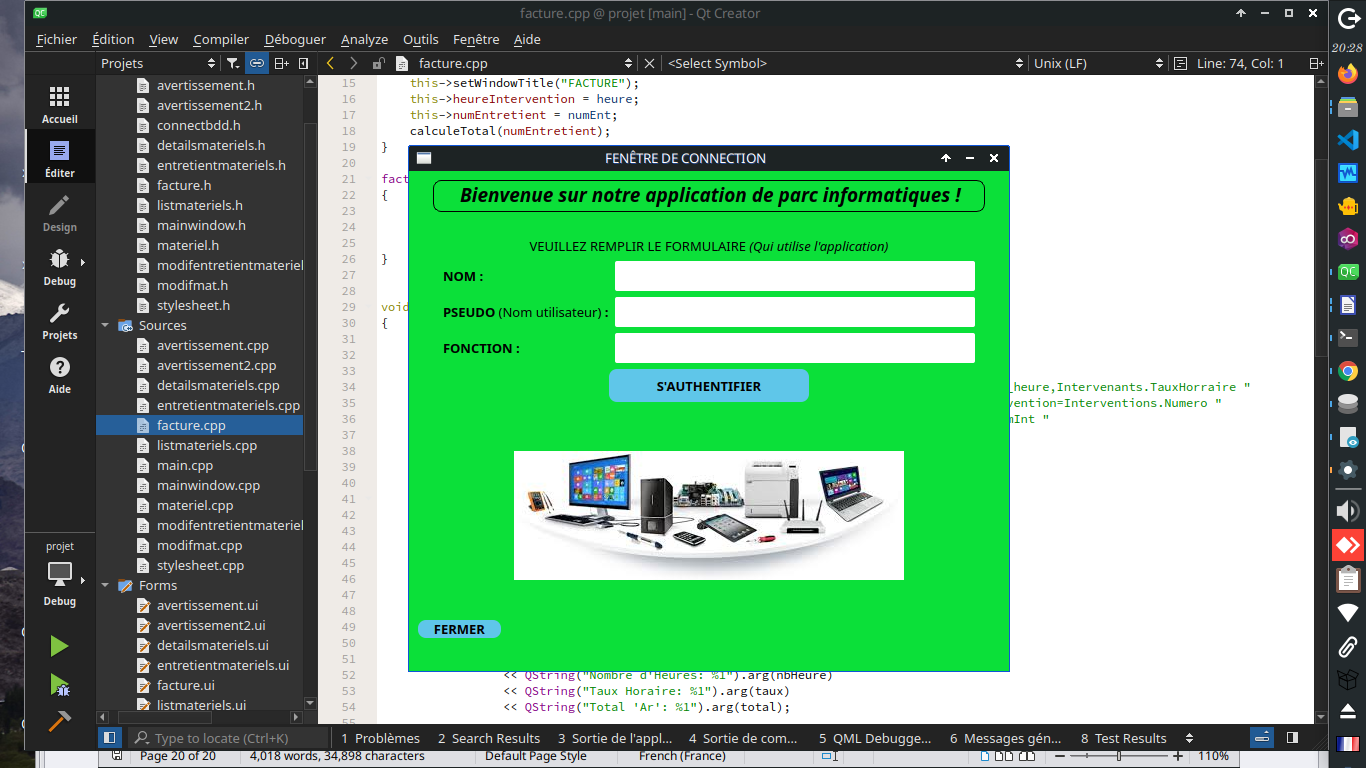
}

}

6.3. PRÉSENTATION DE L’APPLICATION

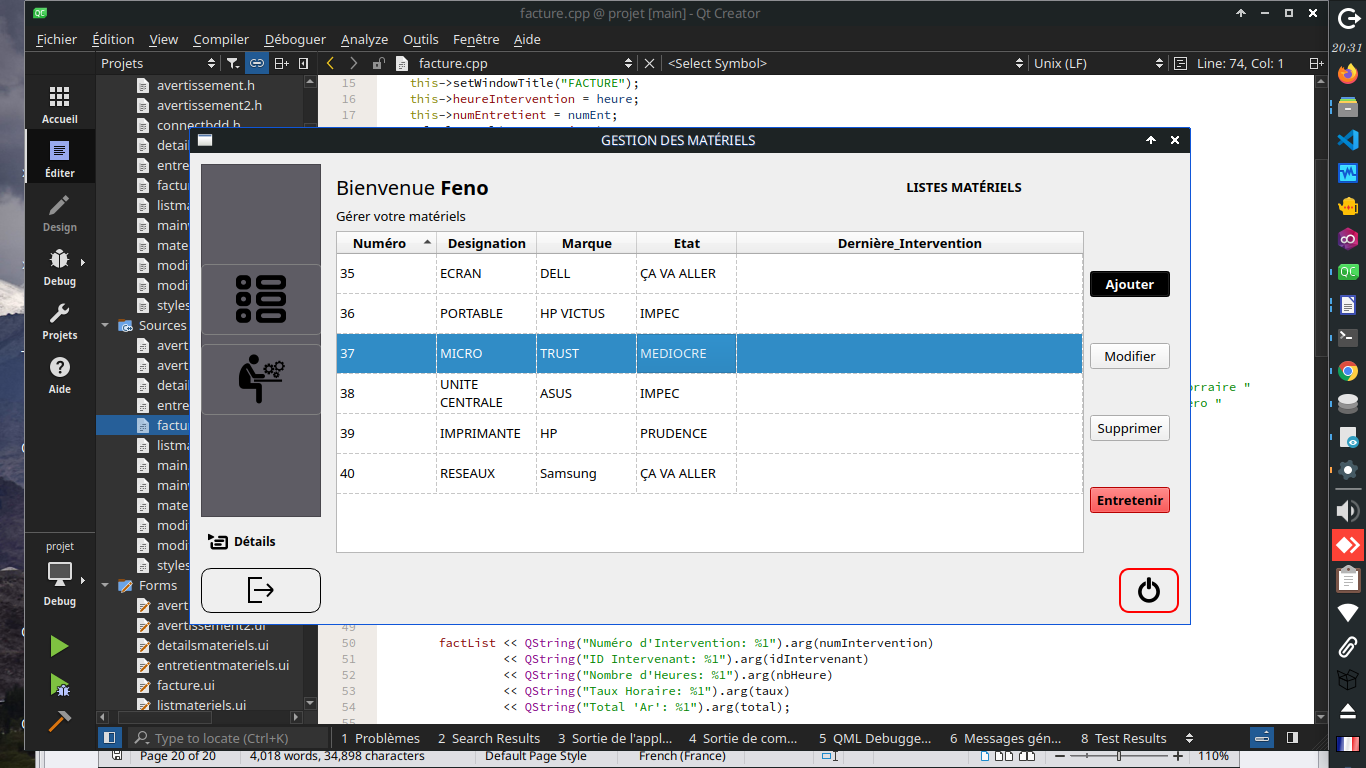
Avant de commencer, je tient à préciser qu’en terme d’interface, on a pas vraiment eu ce qu’on voulais avoir à cause du manque d’experience en terme d’interface.

* *Fenêtre Authentification*

  
En tapant sur Nom, si l’Utilisateur est déjà dans la base de donnée, il donnera automatiquement les suggestion et si dans **pseudo** il ont le même, l’Application renvera une message disant si c’est la même personne et utilisera le pseudo de ce dernier dans la suite.

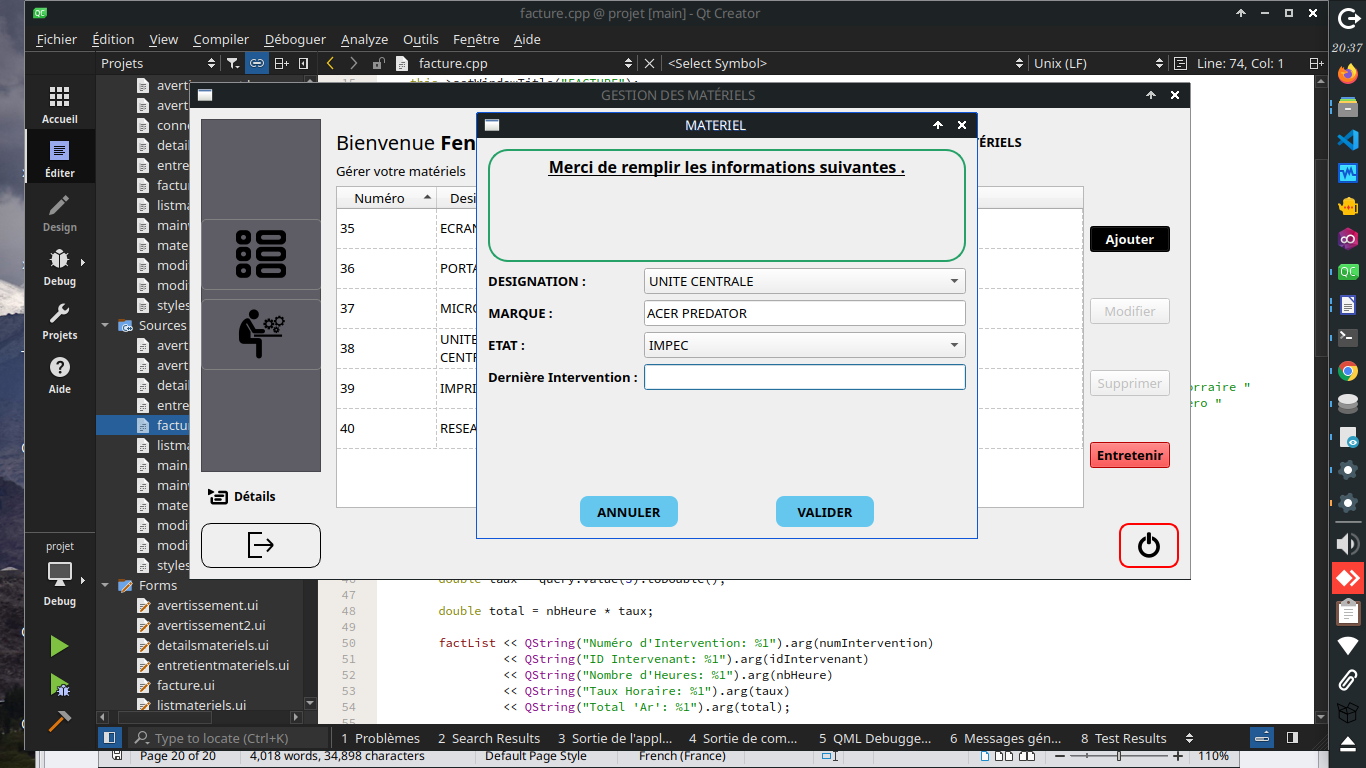
* *Fenêtre acceuil et Liste Matériels*

En entrant dans l’application, l’Utilisateurs sera acceuilli par une message qui dit en haut a gauche « Bienvenue » + pseudo de l’Utilisateurs, la liste des matériels disponible avec l’Etats, la Marque et la date de son dernier entretients



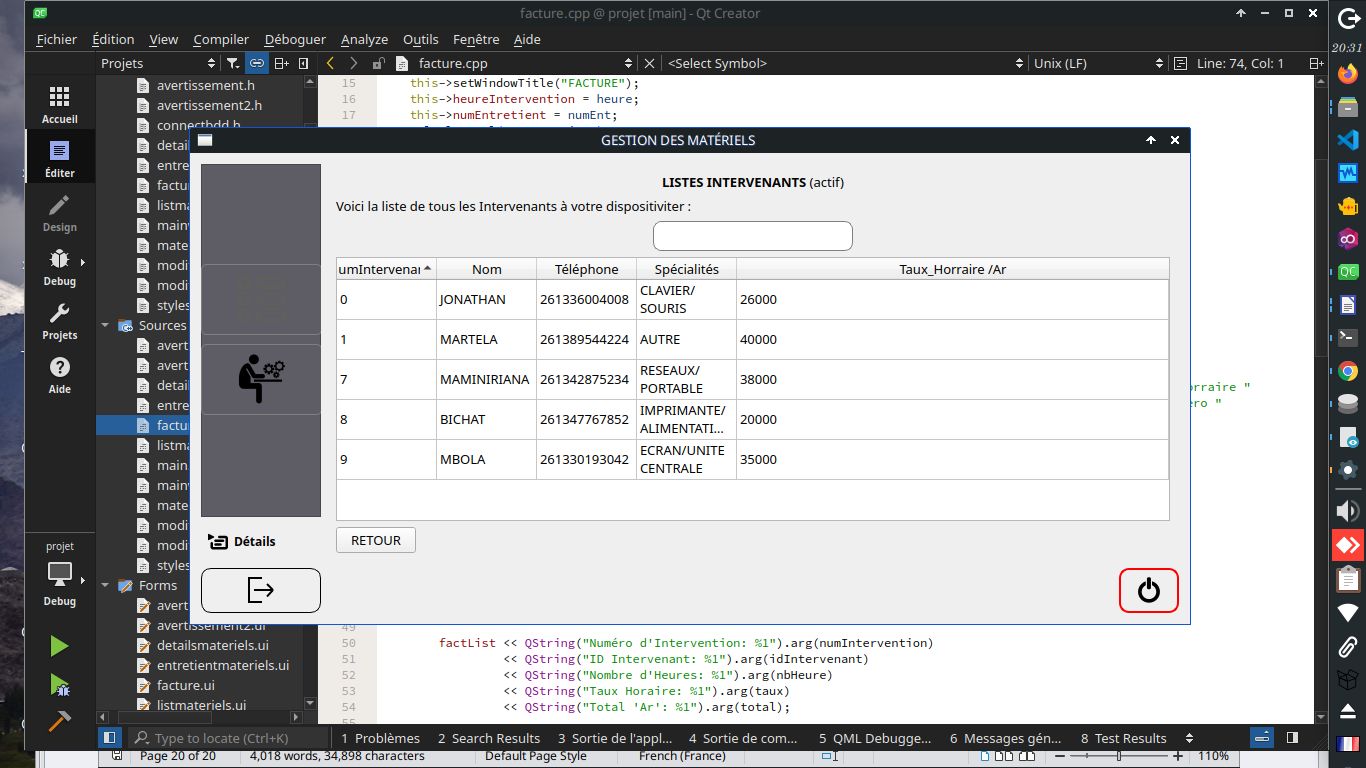
* *Fenêtre Ajout Matériels*

On peut ajouter une Matériels avec la marque qu’on veut et entrer la date du dernière entretient qui est facultatif, le matériel sera afficher immédiatements dans la liste



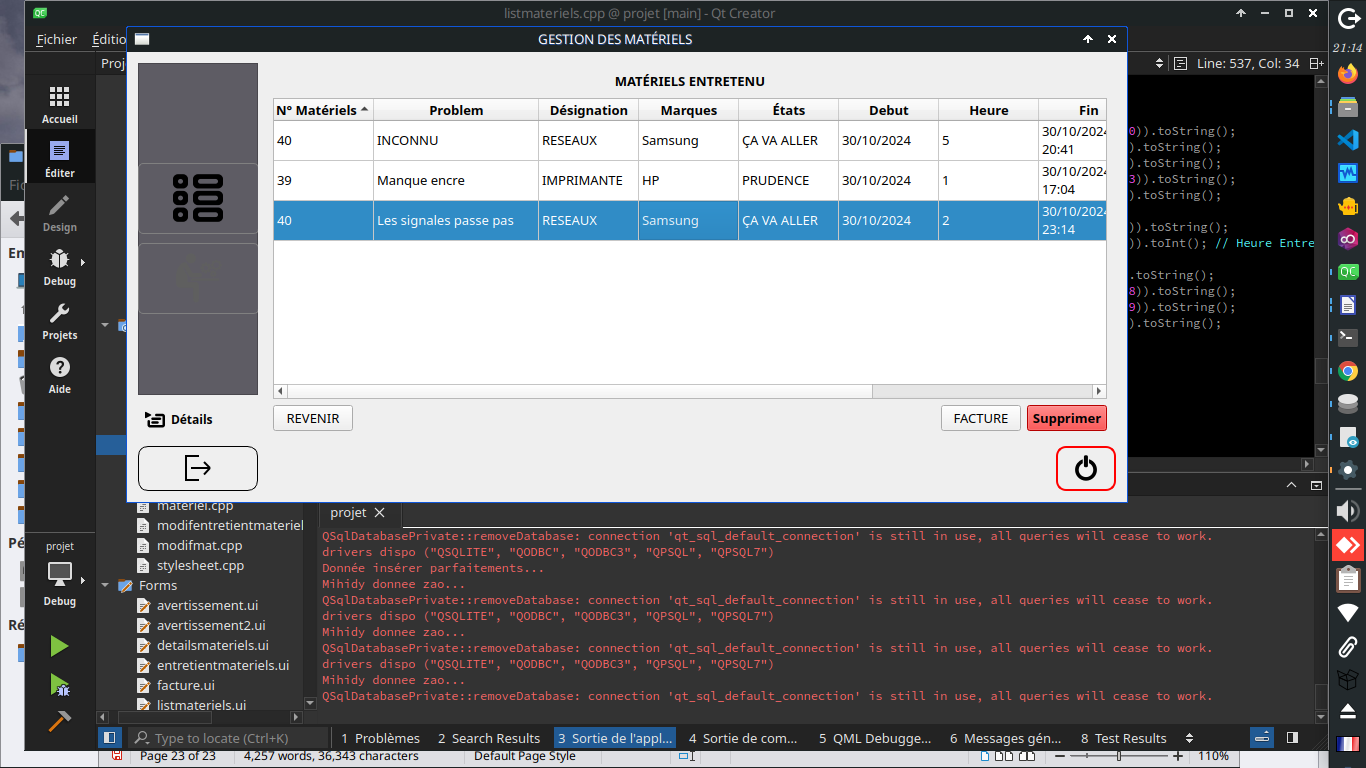
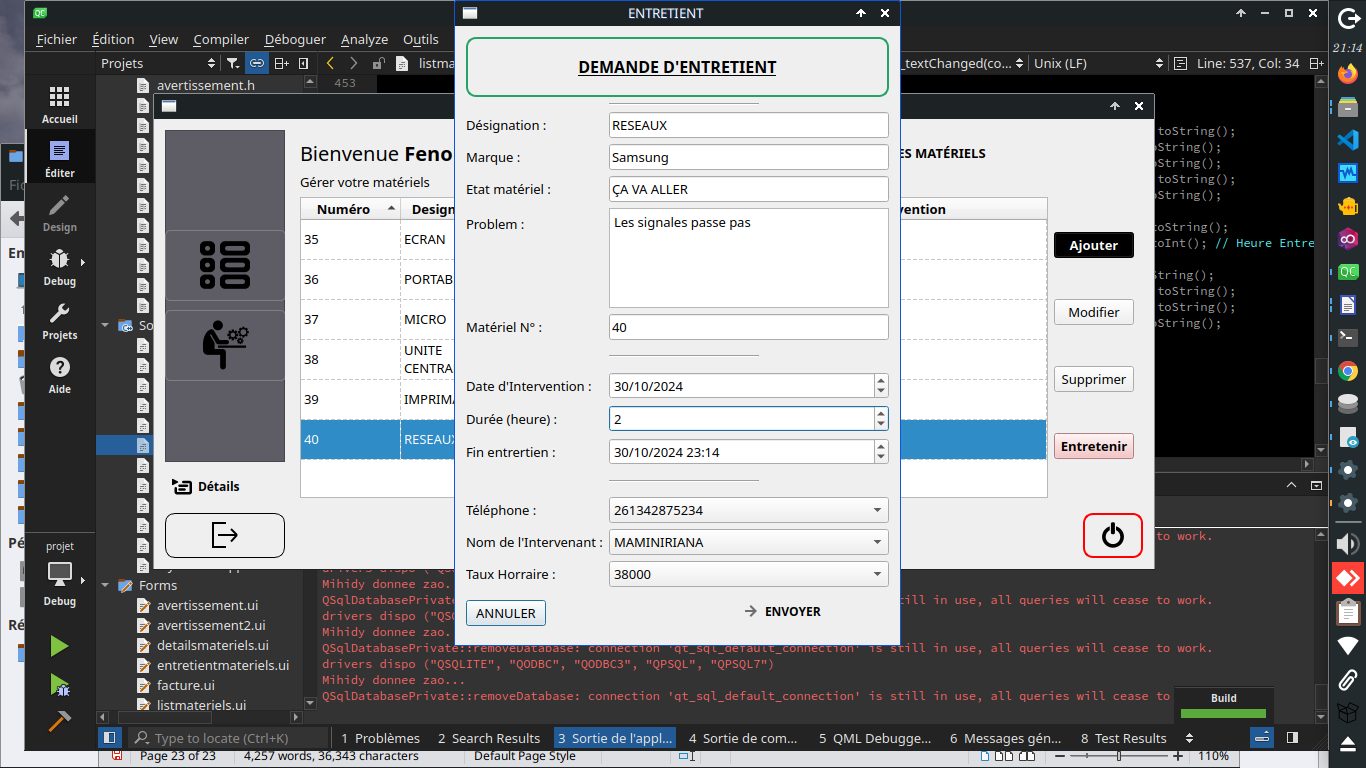
* *Fenêtre Intervenants*

En effet, on peut voir la liste d’intervenant disponible dans une fenêtre et aussi faire une recherche en fonction de son Nom ou son Numéro de Téléphone



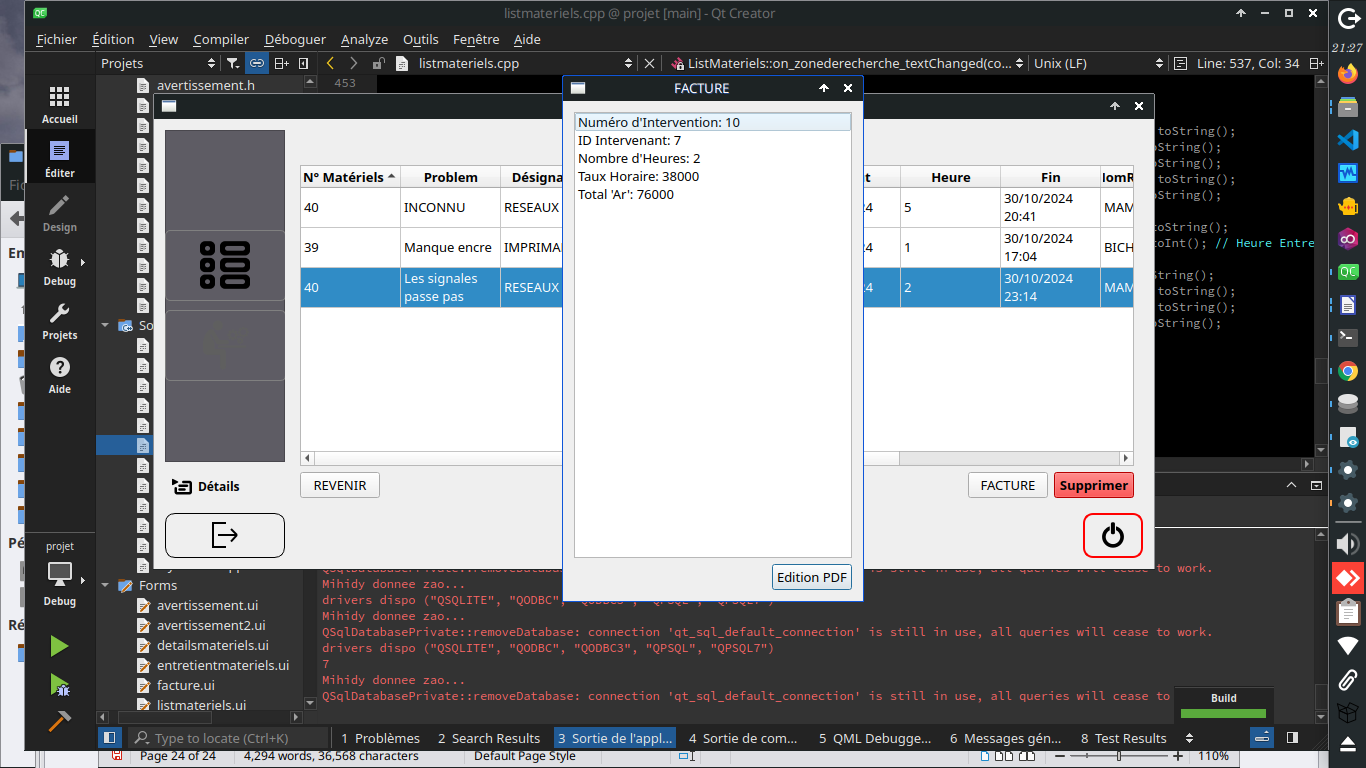
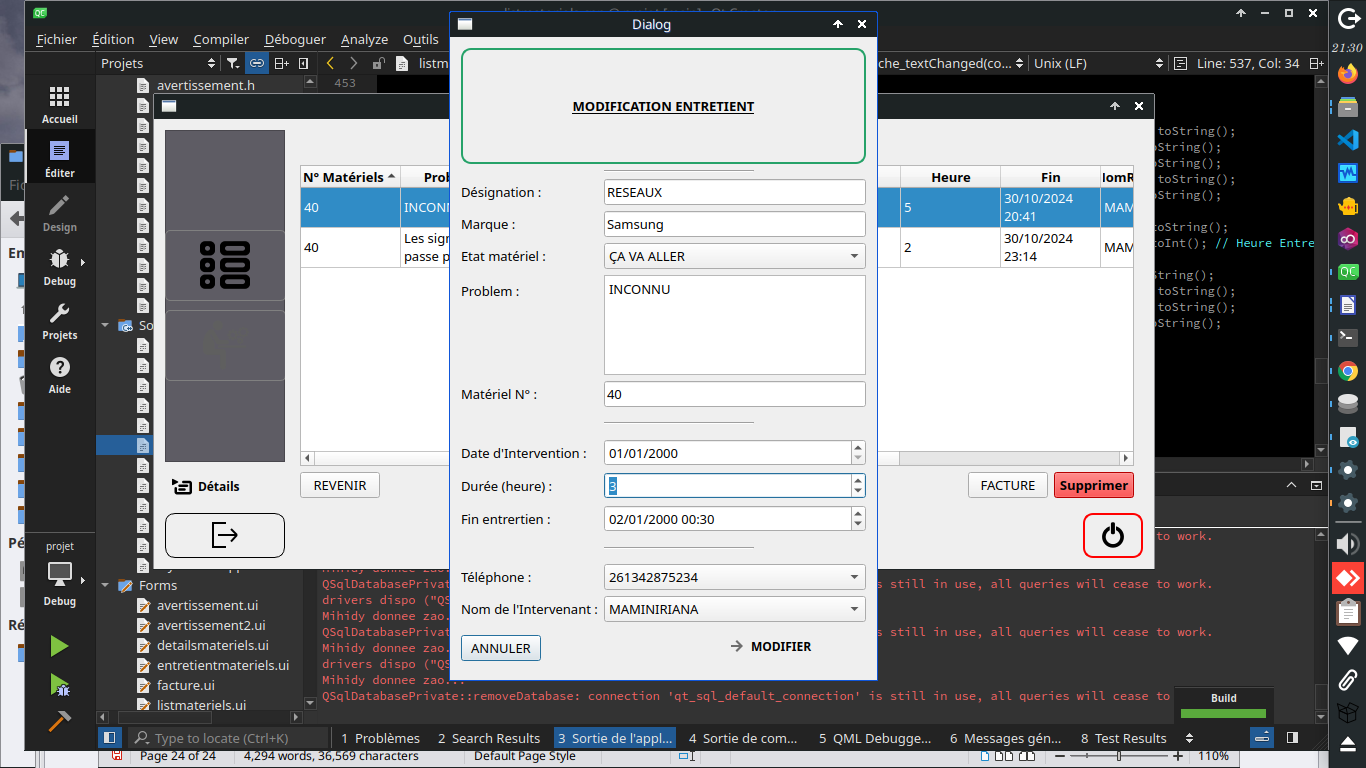
* *Fenêtre Entretients et Intervention*

Dans cette fenêtre sont lister les interventions en cours, c’est a dire que l’Utilisateur click sur une Matériels et peut faire ou demander une intervention en fonction de l’Etats du Matériel. Une fois les Informations entrer il peut envoyer cette demande et il va s’afficher dans la liste de Matériels Entretenue

  
Après

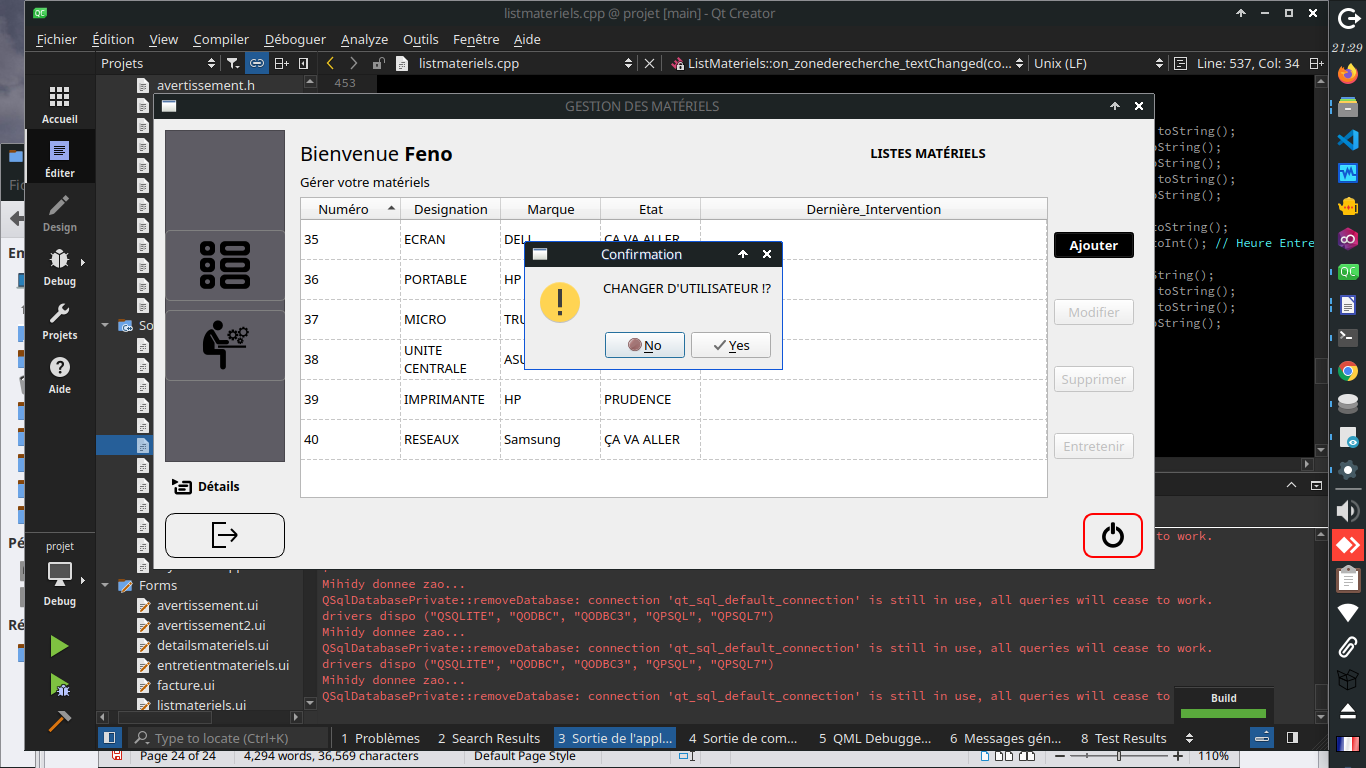
* *Fenêtre facture*

Dans la fenête Entretient, l’Utilisateur peut gérer l’Entretient en cliquant sur une des données, en doucle cliquant pour modifier, apres ça elle peut voir et éditer en PDF la facture de cette entretient

* *Changement Utilisateur*

Enfin, après avoir fini de modifier queque matériels, l’Utilisateur peut se déconnecter en laissant libre place aux autres Utilisateurs

  
***CHRONOGRAMME DU PROJET (diagramme de Gantt)***

|  |  |  |
| --- | --- | --- |
| **Activités** | **Dates** | **Durée** |
| Réception du projet | 23 Septembre 2024 | 1 jour |
| Conception du MCD | 24 au 1 Octobre 2024 | 1 semaine |
| Conception du MLD | 2 au 6 Octobre 2024 | 5 jour |
| Conception de l’Application | 7 au 30 Octobre 2024 | 3 semaine |
| Création du livre | 25 au 30 Octobre 2024 | 1 semaine |