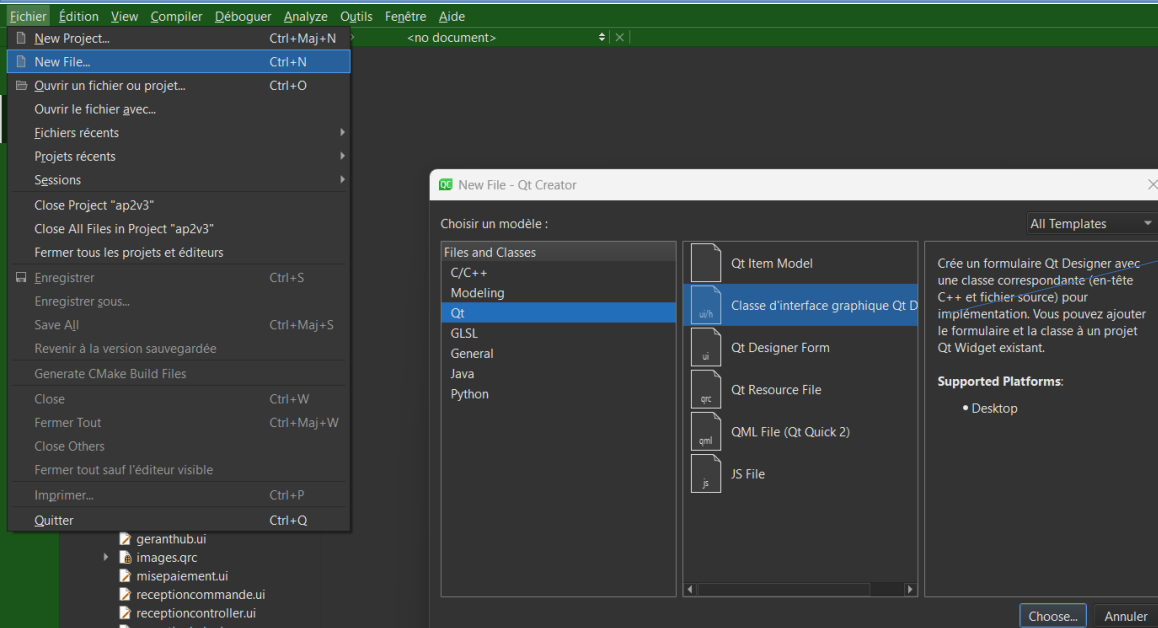
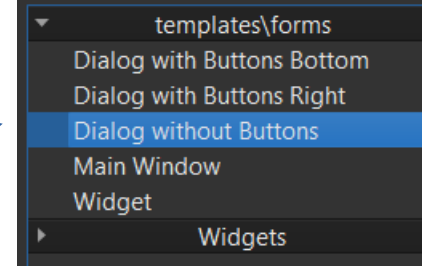


1. Créer une page avec son header et son graphique



Choisir un modèle d'interface graphique



Choisissez un nom de classe

Classe

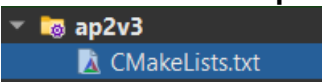
Class name:

Header file:

Source file:

Form file:

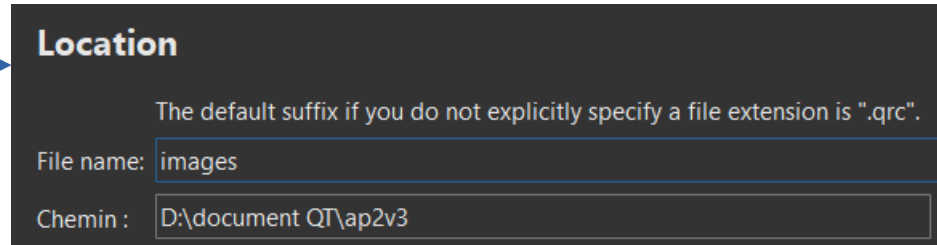
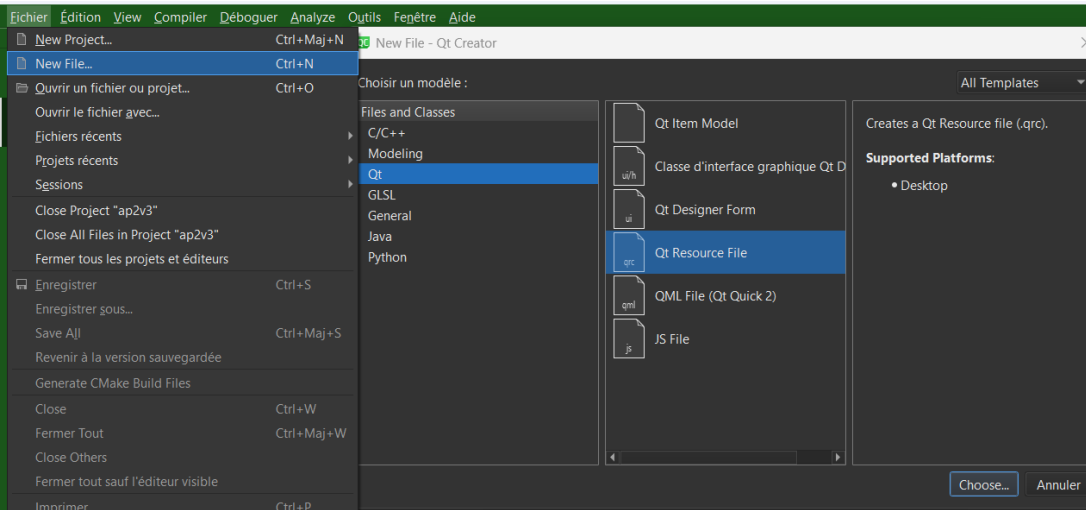
On n'oublie pas de le mettre dans le cmakeList.txt



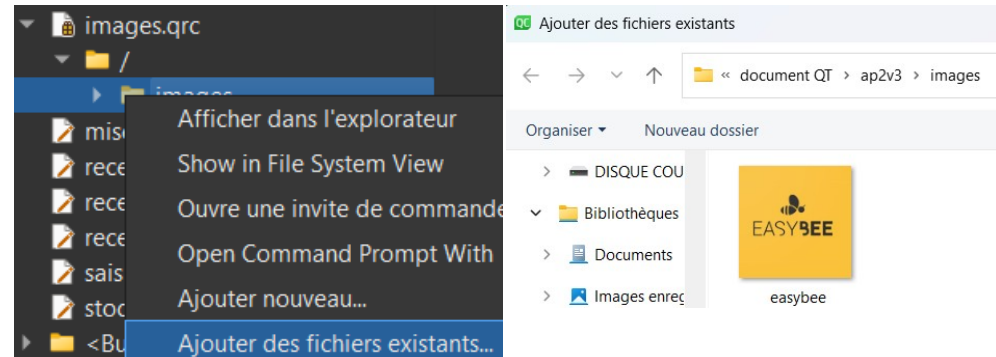
```
set(PROJECT_SOURCES
    initdb.h main.cpp images.qrc
    accueil.h accueil.cpp accueil.ui)
```

2. Créer un fichier ressources et importer une image

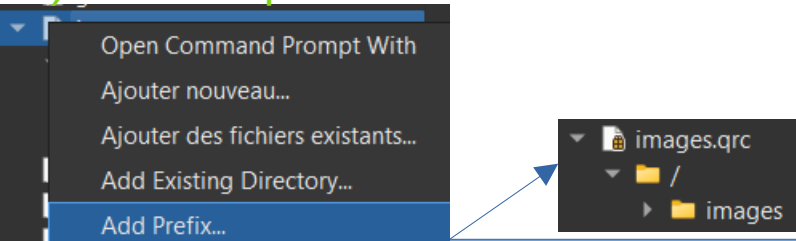
A) Créer le fichier ressources



C) Ajouter l'image



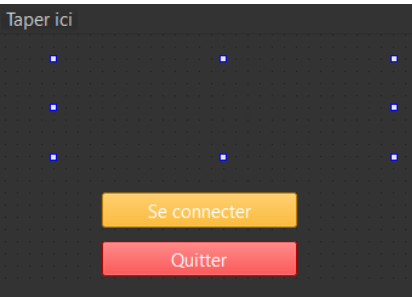
B) Créer le préfixe



3. Ajouter une image, changer le titre de la fenêtre et changer de page par un bouton customisé

A) Ajouter une image

Créer un label vide et adapter sa taille avec l'image.

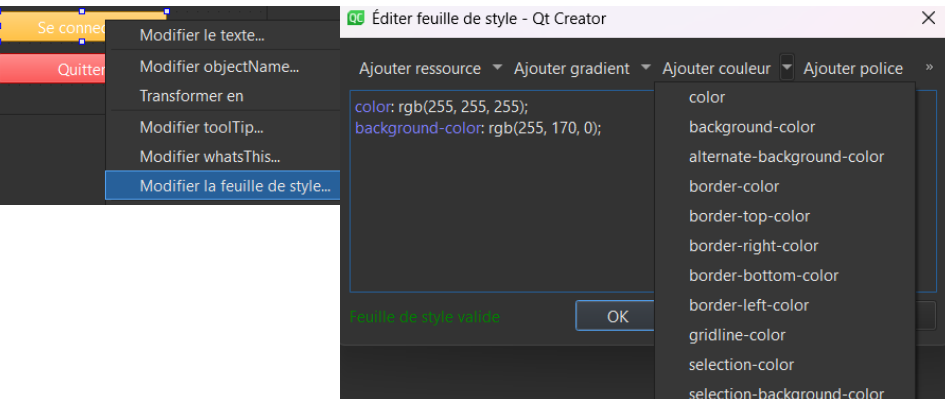


```
#include <QPixmap>
QPixmap image(":/images/easybee.png");
ui->label->setPixmap(image);
```

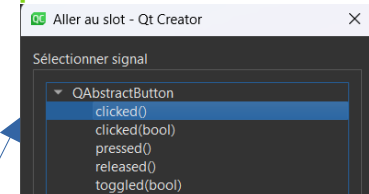
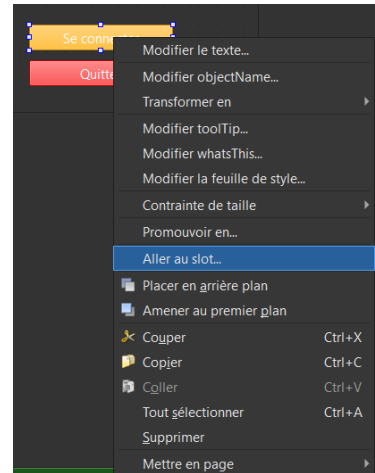
B) Changer le titre de la fenêtre

```
ui->setupUi(this);
// Le code pour l'image
this->setWindowTitle("Page d'accueil");
```

C) Le bouton customisé



D) Changer de page par un bouton



```
#include "authentification.h"
void accueil::on_pushButton_clicked()
{
    authentification mainpage;
    mainpage.setModal(true);
    this->hide();
    mainpage.exec();
}
```

4. Initier une connexion base de données

A) Fichier connectant la BDD au projet QT

```
#ifndef INITDB_H
#define INITDB_H
#include <QtSql>
#include <QMessageBox>
#include <QCoreApplication>
QSqlError initDB()
{
    QSqlDatabase database = QSqlDatabase::addDatabase("QMYSQL");
    database.setHostName("localhost");
    database.setDatabaseName("easybeebdd");
    database.setUserName("root");
    database.setPassword("");
    if (database.open()) {
        return database.lastError();
    }
    return QSqlError();
}
#endif // INITDB_H
```

On n'oublie pas le package dans le cmakeList

```
find_package(Qt6 REQUIRED COMPONENTS Sql)
target_link_libraries(ap2v3 PRIVATE Qt6::Sql)
```

B) Initier la connexion dans une page

Première utilisation (pour vérifier son fonctionne dès le départ)

Dans le cpp concerné :

```
#include "initdb.h"
#include <QtSql>
if(!QSqlDatabase::drivers().contains("QMYSQL"))
    QMessageBox::critical(this,"Impossible de se connecter à la base de
données","Il y a peut être un problème de connexion avec le driver QMYSQL");
QSqlError authenticate = initDB();
if(authenticate.type() != QSqlError::NoError){
    showError(authenticate);
    return;
}
```

Dans le header concerné :

```
#include <QSqlError>
private:
    Ui::authentification *ui;
    void showError(const QSqlError &authenticate);
Si tout fonctionne, ce code sera plus rapide pour la suite
```

Dans le cpp concerné :

```
#include "initdb.h"
QSqlQuery qry(QSqlDatabase::database("easybeeBDD"));
```

Dans le header concerné :

```
#include<QSqlDatabase>
#include <QsqlTableModel>
private:
    QSqlDatabase database;
```

5. Si tu n'as pas installé le driver MySQL

En cherchant dans ton poste, ton objectif est de trouver le .h, .dll et le .lib :

<https://dev.mysql.com/downloads/installer/>

How to Build the QMYSQL Plugin on Windows

You need to get the MySQL installation files (e.g. [mysql-installer-web-community-8.0.22.0.msi](#) or [mariadb-connector-c-3.1.11-win64.msi](#)). Run the installer, select custom installation and install the MySQL C Connector which matches your Qt installation (x86 or x64). After installation check that the needed files are there:

```
> <MySQL_dir>/lib/libmysql.lib
> <MySQL_dir>/lib/libmysql.dll
> <MySQL_dir>/include/mysql.h
```

Note: As of MySQL 8.0.19, the C Connector is no longer offered as a standalone installable component. Instead, you can get `mysql.h` and `libmysql.*` by installing the full MySQL Server (x64 only) or the [MariaDB C Connector](#).

```
C:\Users\Darkus>cd ..
```

```
C:\Users>cd ..
```

```
C:\>d:
```

```
D:\>cd QT6.3.1\mingw_64\build-sqldrivers
```

```
D:\QT6.3.1\mingw_64\build-sqldrivers>qt-cmake -G Ninja "D:\document QT\build-ap2v3-Desktop_Qt_6_3_1_MinGW_64_bit-Debug"
```

```
-DCMAKE_INSTALL_PREFIX="D:\QT6.3.1\mingw_64" -DMySQL_INCLUDE_DIR="C:\Program Files\MySQL\MySQL Server 8.0\include" -DMySQL_LIBRARY="C:\Program Files\MySQL\MySQL Server 8.0\lib\libmysql.lib"
```

```
-- Could NOT find WrapVulkanHeaders (missing: Vulkan_INCLUDE_DIR)
```

```
-- Configuring done
```

```
-- Generating done
```

```
CMake Warning:
```

```
Manually-specified variables were not used by the project:
```

```
  CMAKE_TOOLCHAIN_FILE
```

```
  MySQL_INCLUDE_DIR
```


```
  MySQL_LIBRARY
```

```
-- Build files have been written to: D:/document QT/build-ap2v3-Desktop_Qt_6_3_1_MinGW_64_bit-Debug
```

```
D:\QT6.3.1\mingw_64\build-sqldrivers>cmake --build .
```

```
D:\QT6.3.1\mingw_64\build-sqldrivers>cmake --install .
```

```
D:\QT6.3.1\mingw_64\build-sqldrivers>
```

 Modifier les variables d'environn...

D:\Qt\Tools\CMake_64\bin

D:\Qt\Tools\mingw1120_64

D:\Qt\Tools\mingw1120_64\bin

D:\Qt\6.3.1\mingw_64

D:\Qt\6.3.1\mingw_64\bin

D:\Qt\Tools\Ninja

D:\Qt\6.3.1\mingw_64\plugins\sqldrivers

6. Effectuer des requêtes préparées avec QT

#include <QSqlQuery>

Type de requête	Préparation et exécution de la requête
SELECT	<pre>QSqlQuery qry(QSqlDatabase::database("easybeeBDD")); qry.prepare("SELECT * FROM detailCommande WHERE id_commande = :id_commande"); qry.bindValue(":id_commande", id_commande); qry.exec();</pre>
INSERT	<pre>QSqlQuery qry(QSqlDatabase::database("easybeeBDD")); qry.prepare("INSERT INTO commande (etatCommande, idUtilisateur, dateCommande, prixTotal) " "VALUES (1, 2, :dateCommande, 0)"); qry.bindValue(":dateCommande", dateCommande); qry.exec();</pre>
UPDATE	<pre>QSqlQuery qry(QSqlDatabase::database("easybeeBDD")); qry.prepare("UPDATE commande SET etatCommande='2' WHERE idCommande = :id_commande"); qry.bindValue(":id_commande", id_commande); qry.exec();</pre>
DELETE	<pre>QSqlQuery qry(QSqlDatabase::database("easybeeBDD")); qry.prepare("DELETE FROM commande WHERE idCommande = :id_commande"); qry.bindValue(":id_commande", id_commande); qry.exec();</pre>

7. Configurer des tableView et récupérer des index

A) Gérer la vue de la table

Dans ton header concerné :

```
#include <QSqlTableModel>
private:
    QSqlDatabase database;
    QSqlTableModel *modeleTable;
};
```

Dans ton cpp concerné :

```
modeleTable = new QSqlTableModel();
modeleTable->setTable("produit");
modeleTable->setFilter("stockMinEntrepot > stockEntrepot");
modeleTable → select();
ui->tableView->setModel(modeleTable);
```

D) Paramétrer les colonnes d'une tableView

Au lieu d'un QSqlTableModel, nous allons devoir passer par un QSqlQueryModel :

```
#include <QSqlQueryModel>
QSqlQueryModel *modeleTable = new QSqlQueryModel(this);
QString query = "SELECT libelle FROM produit";
modeleTable → setQuery(query);
ui->tableView->setModel(modeleTable);
```

B) Paramétrer la sélection d'un index

ui → **tableView** → setSelectionBehavior(QAbstractItemView::SelectRows);
Si vous voulez forcer l'utilisateur à sélectionner un élément uniquement.
ui → **tableView** → setSelectionMode(QAbstractItemView::SingleSelection);
Si vous voulez permettre à l'utilisateur de sélectionner 1 à n éléments.
ui → **tableView** → setSelectionMode(QAbstractItemView::MultiSelection);

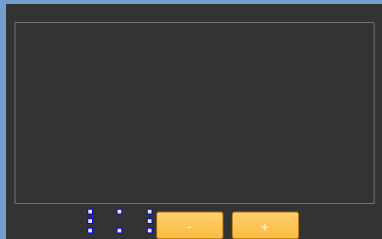
C) Récupérer les valeurs d'un index

```
QModelIndex index = ui->tableView->currentIndex();
int id = modeleTable->record(index.row()).value("id").toInt();
if(!index.isValid()){
    QMessageBox::warning(this,"Erreur de sélection","Sélectionnez une ligne");
    return;
}
//Récupération des données
QSqlRecord record = modeleTable->record(index.row());
```

```
#include <QSqlQuery>
#include <QSqlRecord>
```

8. Créer un compteur stylé

Il faudra mettre un label et 2 boutons



A) Création du code du compteur

Le cpp :

```
#include "compteurEntrepot.h"
#include "ui_compteurEntrepot.h"
#include <QApplication>
#include <QObject>
#include <QPushButton>
#include <QWidget>
compteurEntrepot::compteurEntrepot():entrepot_(0)
{
}
int compteurEntrepot::getEntrepot(){
    return entrepot_;
}
void compteurEntrepot::augmenterEntrepot(){
    entrepot_++;
    emit envoyerEntrepot(entrepot_);
}
void compteurEntrepot::decrementerEntrepot(){
    if(entrepot_==0){
        qDebug() << "La quantité d'un produit ne peut être négative.";
        entrepot_==0;
    }
    else{
        entrepot_--;
    }
    emit envoyerEntrepot(entrepot_);
}
```

Le header

```
#ifndef COMPTEURENTREPOT_H
#define COMPTEURENTREPOT_H
#include <QObject>
class compteurEntrepot: public QObject
{
    Q_OBJECT
public:
    compteurEntrepot();
    int getEntrepot();
    void augmenterEntrepot();
    void decrementerEntrepot();
signals:
    int envoyerEntrepot(int);
private:
    int entrepot_;
};
#endif // COMPTEUR_H
```

B) Setup le compteur dans la page concernée

Le header concerné :

```
public:
    explicit stockEntrepot(QWidget *parent = nullptr);
    ~stockEntrepot();
    void setup();
private slots:
    void on_pushButton_clicked();
    void on_pushButton_2_clicked();
private:
    Ui::stockEntrepot *ui;
    QSqlDatabase database;
    QSqlTableModel *modeleTable;
    compteurEntrepot entrepot_;
```

Le cpp concerné :

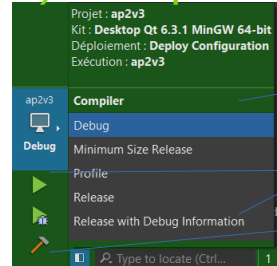
```
ui->label_2->setNum(entrepot_.getEntrepot());
}
stockEntrepot::~stockEntrepot()
{
    delete ui;
}
void stockEntrepot::setup() {
    // Mettre à jour les labels avec les valeurs initiales de entrepot_
    ui->label_2->setNum(entrepot_.getEntrepot());
    // Connecter les signaux et slots entre les boutons et les méthodes correspondantes
    connect(ui->pushButton, SIGNAL(clicked()), this, SLOT(decrementerEntrepot()));
    connect(ui->pushButton_2, SIGNAL(clicked()), this, SLOT(augmenterEntrepot()));
    // Connecter les signaux et slots entre les objets mag_ et res_ et les labels correspondants
    connect(&entrepot_, SIGNAL(envoyerEntrepot(int)), ui->label_2, SLOT(setNum(int)));
}
void stockEntrepot::on_pushButton_clicked() //decrementer entrepot
{
    entrepot_.decrementerEntrepot();
    ui->label_2->setNum(entrepot_.getEntrepot());
}
void stockEntrepot::on_pushButton_2_clicked() //augmenter entrepot
{
    entrepot_.augmenterEntrepot();
    ui->label_2->setNum(entrepot_.getEntrepot());
}
```


9. Boîtes de messages, compilation du code et entête

A) QMessageBox et Debug

```
#include <QMessageBox>
QMessageBox::critical(this, "Title", "Subtitle");
QMessageBox::information(this, "Title", "Subtitle");
qDebug() << "id: " << id;
```

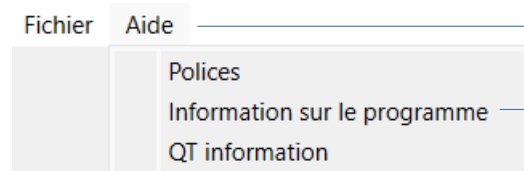
B) Compilation du code



Debug mode
Release mode
Exécuter
Compiler

C) Créer une entête et le relier à une méthode

```
void accueil::creerBarreMenu()
{
    QAction *closeMenu = new QAction(tr("&Fermer"), this);
    QAction *FontInfo = new QAction(tr("&Polices"), this);
    QAction *infoProg = new QAction(tr("&Information sur le programme"), this);
    QAction *QTInfo = new QAction(tr("&QT information"), this);
    QMenu *fileMenu = menuBar()->addMenu(tr("&Fichier"));
    QMenu *helpMenu = menuBar()->addMenu(tr("&Aide"));
    fileMenu->addAction(closeMenu);
    helpMenu->addAction(FontInfo);
    helpMenu->addAction(infoProg);
    helpMenu->addAction(QTInfo);
    connect(closeMenu, &QAction::triggered, this, &accueil::closeMenu);
    connect(FontInfo, &QAction::triggered, this, &accueil::showInfo);
    connect(infoProg, &QAction::triggered, this, &accueil::QTInfo);
    connect(QTInfo, &QAction::triggered, this, &QApplication::aboutQt);
}
```



QMenu helpMenu

Qaction FontInfo

Ici, on ajoute l'action
FontInfo au menu
helpMenu

Ici, on associe
l'action à la méthode
showInfo

Appellez creerBarreMenu(); au début de la page !