Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное учреждение

высшего образования

**«Пермский национальный** **исследовательский политехнический университет»**

Электротехнический факультет

Кафедра «Информационные технологии и автоматизированные системы»

направление подготовки: 09.03.01– «Информатика и вычислительная техника»

**Лабораторная работа**

**по дисциплине**

**«Информатика»**

**на тему**

**«Автоматизированное рабочее место управляющего сотрудниками»**

Выполнил студент гр. ИВТ-23-1б

Бакин Владислав Артемович

Проверил:

доц. каф. ИТАС

Полякова Ольга Андреевна

Яруллин Денис Владимирович

(оценка) (подпись)

(дата)

г. Пермь, 2024

# Функционал автоматизированного рабочего места

Название автоматизированного рабочего места ChiefDesk (от англ. Главный стол) - автоматизированное рабочее место управляющего сотрудниками.

**Функционал**:

1. Создание профиля сотрудника компании и начальника компании.
2. Два разных окна управления (ChiefDesk и EmployeeDesk), в зависимости от роли сотрудника (chief или employee).
3. Просмотр профиля:
   1. имя профиля,
   2. права доступа,
   3. название компании,
   4. количество сотрудников в компании.
      1. Для профиля начальника реализована возможность получить код приглашения, который будет выдаваться сотрудникам, чтобы они могли создать профиль в системе ChiefDesk.
4. Просмотр сотрудников в компании и их должностей.
5. Контроль ресурсов компании.
6. Просмотр задач компании:
   1. Для начальника:
      1. создание новой задачи и её присвоение к сотруднику,
      2. получение отчета по задаче.
   2. Для сотрудника:
      1. возможность отметить задачу как выполненную,
      2. написание отчета по задаче.

# Реализация АРМ

Для хранения данных добавлена база данных SQLite.

**Структура базы данных**:

Таблица Companies:

CREATE TABLE "Companies" (  
 "company\_id" INTEGER,  
 "company\_name" TEXT NOT NULL,  
 "company\_code" TEXT NOT NULL,  
 PRIMARY KEY("company\_id" AUTOINCREMENT)  
);

Таблица Employees:

CREATE TABLE "Employees" (  
 "employee\_id" INTEGER,  
 "login" TEXT NOT NULL,  
 "password" TEXT NOT NULL,  
 "role" TEXT NOT NULL,  
 "company\_id" INTEGER NOT NULL,  
 "task\_id" INTEGER,  
 PRIMARY KEY("employee\_id")  
);

Таблица Resources:

CREATE TABLE "Resources" (  
 "res\_id" INTEGER NOT NULL,  
 "res\_name" INTEGER NOT NULL,  
 "res\_amount" INTEGER NOT NULL,  
 "company\_id" INTEGER NOT NULL,  
 PRIMARY KEY("res\_id" AUTOINCREMENT)  
);

Таблица Tasks:

CREATE TABLE "Tasks" (  
 "task\_id" INTEGER NOT NULL,  
 "task\_name" TEXT NOT NULL,  
 "status" INTEGER NOT NULL,  
 "employee\_id" INTEGER,  
 "company\_id" INTEGER,  
 "report" TEXT,  
 PRIMARY KEY("task\_id" AUTOINCREMENT)  
);

Для создания графического интерфейса использовался фреймворк Qt. Для взаимодействия с базой данных подключены QSqlDatabase, для обработки ошибок QSqlError, для обработки запросов QSqlQuery, для визуализации таблицы в интерфейсе использовался QTableView, куда передавались модели QSqlTableModel.

# Код приложения

### Заголовочные файлы

#ifndef AUTHORIZATION\_H  
#define AUTHORIZATION\_H  
  
#include "loginwindow.h"  
#include "registrationwindow.h"  
  
#include <QMainWindow>  
#include <QSqlDatabase>  
#include <QSqlError>  
   
namespace Ui {  
class Authorization;  
}  
  
   
  
class Authorization : public QMainWindow  
{  
 Q\_OBJECT  
public:  
 explicit Authorization(QWidget \*parent = nullptr);  
 ~Authorization();  
  
public slots:  
 void login();  
 void signUp();  
  
private:  
 QSqlDatabase db;  
 Ui::Authorization \*ui;  
};  
  
#endif // AUTHORIZATION\_H

#ifndef REGISTRATIONWINDOW\_H  
#define REGISTRATIONWINDOW\_H  
#include <QDialog>  
#include <QDebug>  
#include <QSqlDatabase>  
#include <QSqlQuery>  
#include <QSqlError>   
  
namespace Ui {  
class RegistrationWindow;  
}  
  
class RegistrationWindow : public QDialog  
{  
 Q\_OBJECT  
  
public:  
 explicit RegistrationWindow(QSqlDatabase& \_db, QWidget \*parent = nullptr);  
 ~RegistrationWindow();   
  
 QString generateRandomCode(int length);  
  
   
  
public slots:  
 void changeMode(int state);  
 void reg();   
  
private:  
 QSqlDatabase db;  
 Ui::RegistrationWindow \*ui;  
  
};  
  
#endif // REGISTRATIONWINDOW\_H

#ifndef LOGINWINDOW\_H  
#define LOGINWINDOW\_H  
#include <QDialog>  
#include <QDebug>  
#include <QSqlDatabase>  
#include <QSqlError>  
#include <QSqlQuery>   
  
#include "chiefdesk.h"  
#include "employeedesk.h"  
  
namespace Ui {  
class LoginWindow;  
}  
  
class LoginWindow : public QDialog  
{  
 Q\_OBJECT   
  
public:  
 explicit LoginWindow(QSqlDatabase& \_db, QWidget \*parent = nullptr);  
 ~LoginWindow();  
   
public slots:  
 void login();  
   
  
private:  
 QSqlDatabase db;  
 Ui::LoginWindow \*ui;  
};  
   
#endif // LOGINWINDOW\_H

#ifndef CHIEFDESK\_H  
#define CHIEFDESK\_H  
  
#include <QMainWindow>  
#include <QDebug>  
#include <QSqlDatabase>  
#include <QSqlQuery>  
#include <QSqlError>  
  
#include "profile.h"  
#include "employeesview.h"  
#include "resourcesview.h"  
#include "tasksview.h"  
  
namespace Ui {  
class ChiefDesk;  
}  
  
   
  
class ChiefDesk : public QMainWindow  
{  
 Q\_OBJECT  
  
public:  
 explicit ChiefDesk(QSqlDatabase& \_db, int employeeId, QWidget \*parent = nullptr);  
 ~ChiefDesk();   
  
public slots:  
 void viewProfile();  
 void viewEmployees();  
 void viewResources();  
 void viewTasks();  
  
   
  
private:  
 QSqlDatabase db;  
 int id;  
 Ui::ChiefDesk \*ui;  
};  
  
#endif // CHIEFDESK\_H

#ifndef EMPLOYEEDESK\_H  
#define EMPLOYEEDESK\_H   
  
#include <QMainWindow>  
#include <QDebug>  
#include <QSqlDatabase>  
#include <QSqlQuery>  
#include <QSqlError>   
  
#include "profile.h"  
#include "employeesview.h"  
#include "resourcesview.h"  
#include "tasksviewforemployee.h"  
   
namespace Ui {  
 class EmployeeDesk;  
}  
  
   
class EmployeeDesk : public QMainWindow  
{  
 Q\_OBJECT   
  
public:  
 explicit EmployeeDesk(QSqlDatabase& \_db, int employeeId, QWidget \*parent = nullptr);  
 ~EmployeeDesk();  
  
   
  
public slots:  
 void viewProfile();  
 void viewEmployees();  
 void viewResources();  
 void viewTasks();  
  
private:  
 QSqlDatabase db;  
 int id;  
 Ui::EmployeeDesk \*ui;  
};  
   
#endif // EMPLOYEEDESK\_H

#ifndef PROFILE\_H  
#define PROFILE\_H  
  
#include <QDialog>  
#include <QClipboard>  
#include <QTimer>  
  
namespace Ui {  
 class Profile;  
}  
  
   
  
class Profile : public QDialog  
{  
 Q\_OBJECT  
  
public:  
 explicit Profile(QList<QString> \_profileData, QWidget \*parent = nullptr);  
 ~Profile();  
  
public slots:  
 void getInvCode();  
 void back();  
  
private:  
 QList<QString> profileData;  
 Ui::Profile \*ui;  
};  
   
  
#endif // PROFILE\_H

#ifndef EMPLOYEESVIEW\_H  
#define EMPLOYEESVIEW\_H  
  
#include <QDialog>  
#include <QSqlDatabase>  
#include <QSqlQueryModel>  
#include <QSqlError>  
#include <QDebug>  
#include <QSqlTableModel>  
#include <QTableView>  
  
namespace Ui {  
 class EmployeesView;  
}   
  
class EmployeesView : public QDialog  
{  
 Q\_OBJECT  
  
public:  
 explicit EmployeesView(QSqlDatabase& \_db, QString \_companyId, QWidget \*parent = nullptr);  
 ~EmployeesView();  
 void printTable();  
public slots:  
 void back();  
   
private:  
 QSqlDatabase db;  
 QString companyId;  
 QSqlTableModel \*modelEmpl;  
 QSqlTableModel \*modelTasks;  
 Ui::EmployeesView \*ui;  
};  
  
#endif // EMPLOYEESVIEW\_H

#ifndef RESOURCESVIEW\_H  
#define RESOURCESVIEW\_H  
  
#include <QDialog>  
#include <QDebug>  
#include <QSqlDatabase>  
#include <QSqlTableModel>  
#include <QSqlError>  
#include <QSqlQuery>  
  
namespace Ui {  
class ResourcesView;  
}  
  
class ResourcesView : public QDialog  
{  
 Q\_OBJECT  
  
public:  
 explicit ResourcesView(QSqlDatabase& \_db, QString \_companyId, QWidget \*parent = nullptr);  
 ~ResourcesView();  
  
public slots:  
 void back();  
 void addRes();  
 void removeRes();  
  
private:  
 QSqlDatabase db;  
 QString companyId;  
 QSqlTableModel \*modelRes;  
 Ui::ResourcesView \*ui;  
};  
  
#endif // RESOURCESVIEW\_H

#ifndef TASKSVIEW\_H  
#define TASKSVIEW\_H  
  
#include <QDialog>  
#include <QDebug>  
#include <QSqlDatabase>  
#include <QSqlError>  
#include <QSqlTableModel>  
#include <QSqlQuery>  
  
#include "getreport.h"  
  
namespace Ui {  
class TasksView;  
}  
  
class TasksView : public QDialog  
{  
 Q\_OBJECT  
  
public:  
 explicit TasksView(QSqlDatabase& \_db, QString \_companyId, QWidget \*parent = nullptr);  
 ~TasksView();  
  
public slots:  
 void back();  
 void addTask();  
 void getReport();  
  
private:  
 QSqlDatabase db;  
 QString companyId;  
 QSqlTableModel \*modelTasks;  
 Ui::TasksView \*ui;  
};  
  
#endif // TASKSVIEW\_H

#ifndef TASKSVIEWFOREMPLOYEE\_H  
#define TASKSVIEWFOREMPLOYEE\_H  
  
#include <QDialog>  
#include <QDebug>  
#include <QSqlDatabase>  
#include <QSqlTableModel>  
#include <QSqlQuery>  
#include <QSqlError>  
  
#include "createreport.h"  
  
namespace Ui {  
class TasksViewForEmployee;  
}  
  
class TasksViewForEmployee : public QDialog  
{  
 Q\_OBJECT  
  
public:  
 explicit TasksViewForEmployee(QSqlDatabase& \_db, QString \_companyId, QString \_employeeId, QWidget \*parent = nullptr);  
 ~TasksViewForEmployee();  
  
public slots:  
 void back();  
 void complete();  
 void createReport();  
  
private:  
 QSqlDatabase db;  
 QString companyId;  
 QString employeeId;  
 QSqlTableModel \*modelTasks;  
 Ui::TasksViewForEmployee \*ui;  
};  
  
#endif // TASKSVIEWFOREMPLOYEE\_H

#ifndef GETREPORT\_H  
#define GETREPORT\_H  
  
#include <QDialog>  
#include <QDebug>  
#include <QSqlDatabase>  
#include <QSqlQuery>  
#include <QSqlError>  
  
namespace Ui {  
class GetReport;  
}  
  
class GetReport : public QDialog  
{  
 Q\_OBJECT  
  
public:  
 explicit GetReport(QSqlDatabase& \_db, QString \_taskId, QWidget \*parent = nullptr);  
 ~GetReport();  
  
public slots:  
 void back();  
 void getReport();  
  
private:  
 QSqlDatabase db;  
 QString taskId;  
 Ui::GetReport \*ui;  
};  
  
#endif // GETREPORT\_H

#ifndef CREATEREPORT\_H  
#define CREATEREPORT\_H  
  
#include <QDialog>  
#include <QDebug>  
#include <QSqlDatabase>  
#include <QSqlQuery>  
#include <QSqlError>  
  
namespace Ui {  
class CreateReport;  
}  
  
class CreateReport : public QDialog  
{  
 Q\_OBJECT  
  
public:  
 explicit CreateReport(QSqlDatabase& \_db, QString \_companyId, QString \_employeeId, QString \_taskId, QWidget \*parent = nullptr);  
 ~CreateReport();  
  
public slots:  
 void addReport();  
  
private:  
 QSqlDatabase db;  
 QString companyId;  
 QString employeeId;  
 QString taskId;  
 Ui::CreateReport \*ui;  
};  
  
#endif // CREATEREPORT\_H

### Исходные файлы

#include "authorization.h"  
#include "ui\_authorization.h"  
  
Authorization::Authorization(QWidget \*parent) :  
 QMainWindow(parent),  
 ui(new Ui::Authorization)  
{  
 ui->setupUi(this);  
  
 db = QSqlDatabase::addDatabase("QSQLITE");  
 db.setDatabaseName("/home/meidori/Рабочий стол/main/Sem\_2/Labs/automated\_workstation/Database/chiefdesk.db");  
 if (db.open())  
 {  
 ui->statusbar->showMessage("Successful connection to the database: " + db.databaseName());  
 }  
 else  
 {  
 ui->statusbar->showMessage("An error occurred while connecting to the database: " + db.lastError().databaseText());  
 }  
  
 connect(ui->logInBtn, &QPushButton::clicked, this, &Authorization::login);  
 connect(ui->signUpBtn, &QPushButton::clicked, this, &Authorization::signUp);  
}  
  
Authorization::~Authorization()  
{  
 delete ui;  
}  
  
void Authorization::login()  
{  
 hide();  
  
 LoginWindow loginWindow(db);  
 loginWindow.setModal(true);  
 loginWindow.exec();  
}  
  
void Authorization::signUp()  
{  
 hide();  
  
 RegistrationWindow regWindow(db);  
 regWindow.setModal(true);  
 regWindow.exec();  
  
 show();  
}

#include "registrationwindow.h"  
#include "ui\_registrationwindow.h"  
  
RegistrationWindow::RegistrationWindow(QSqlDatabase& \_db, QWidget \*parent) :  
 QDialog(parent), db(\_db),  
 ui(new Ui::RegistrationWindow)  
{  
 ui->setupUi(this);  
  
 connect(ui->modeCheckBox, &QCheckBox::stateChanged, this, &RegistrationWindow::changeMode);  
 connect(ui->signupBtn, &QPushButton::clicked, this, &RegistrationWindow::reg);  
}  
  
RegistrationWindow::~RegistrationWindow()  
{  
 delete ui;  
}  
  
void RegistrationWindow::changeMode(int state)  
{  
 if (state == Qt::Checked)  
 {  
 ui->companyLabel->setText("Company");  
 }  
 else  
 {  
 ui->companyLabel->setText("Invitation");  
 }  
}  
  
void RegistrationWindow::reg()  
{  
 QString log = ui->loginLineEdit->text();  
 QString pass = ui->passLineEdit->text();  
 QString comp = ui->companyLineEdit->text();  
 bool modeChecked = ui->modeCheckBox->isChecked();  
 if (!log.isEmpty() && !pass.isEmpty() && !comp.isEmpty() && modeChecked)  
 {  
 // Генерация company\_code  
 QString companyCode = comp + "-" + generateRandomCode(7);  
  
 // Создание компании  
 QSqlQuery query;  
 query.prepare("INSERT INTO Companies (company\_name, company\_code) VALUES (:name, :code)");  
 query.bindValue(":name", comp);  
 query.bindValue(":code", companyCode);  
 if (!query.exec())  
 {  
 qDebug() << "Error inserting company:" << query.lastError().text();  
 return; // Выйти из функции при ошибке  
 }  
  
 // Получение company\_id  
 int companyId = query.lastInsertId().toInt();  
  
 // Создание главного сотрудника  
 query.prepare("INSERT INTO Employees (login, password, role, company\_id) VALUES (:login, :password, :role, :companyId)");  
 query.bindValue(":login", log);  
 query.bindValue(":password", pass);  
 query.bindValue(":role", "chief");  
 query.bindValue(":companyId", companyId);  
 if (!query.exec())  
 {  
 qDebug() << "Error inserting chief:" << query.lastError().text();  
 return; // Выйти из функции при ошибке  
 }  
  
 // Успешно завершено  
 qDebug() << "Registration successful!";  
 close();  
 }  
 else if (!log.isEmpty() && !pass.isEmpty() && !comp.isEmpty() && !modeChecked)  
 {  
 // Поиск company\_id по company\_code  
 QSqlQuery query;  
 query.prepare("SELECT company\_id FROM Companies WHERE company\_code = :code");  
 query.bindValue(":code", comp);  
 if (!query.exec())  
 {  
 qDebug() << "Error searching for company:" << query.lastError().text();  
 return; // Выйти из функции при ошибке  
 }  
  
 int companyId = -1;  
 if (query.next())  
 {  
 companyId = query.value(0).toInt();  
 }  
 else  
 {  
 qDebug() << "Company with code" << comp << "not found!";  
 return; // Выйти из функции, если компания не найдена  
 }  
  
 // Создание сотрудника с ролью "employee"  
 query.prepare("INSERT INTO Employees (login, password, role, company\_id) VALUES (:login, :password, :role, :companyId)");  
 query.bindValue(":login", log);  
 query.bindValue(":password", pass);  
 query.bindValue(":role", "employee");  
 query.bindValue(":companyId", companyId);  
 if (!query.exec())  
 {  
 qDebug() << "Error inserting employee:" << query.lastError().text();  
 return; // Выйти из функции при ошибке  
 }  
  
 // Успешно завершено  
 qDebug() << "Employee registration successful!";  
 close();  
 }  
  
}  
  
// Функция для генерации случайного кода указанной длины  
QString RegistrationWindow::generateRandomCode(int length)  
{  
 QString possibleCharacters = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789";  
 QString randomString;  
 for(int i = 0; i < length; i++)  
 {  
 int index = qrand() % possibleCharacters.length();  
 QChar nextChar = possibleCharacters.at(index);  
 randomString.append(nextChar);  
 }  
 return randomString;  
}

#include "loginwindow.h"  
#include "ui\_loginwindow.h"  
  
LoginWindow::LoginWindow(QSqlDatabase& \_db, QWidget \*parent) :  
 QDialog(parent), db(\_db),  
 ui(new Ui::LoginWindow)  
{  
 ui->setupUi(this);  
  
 connect(ui->loginBtn, &QPushButton::clicked, this, &LoginWindow::login);  
}  
  
LoginWindow::~LoginWindow()  
{  
 delete ui;  
}  
  
void LoginWindow::login()  
{  
 QString log = ui->loginLineEdit->text();  
 QString pass = ui->passLineEdit->text();  
 qDebug() << log << ":" << pass;  
 if (!log.isEmpty() && !pass.isEmpty())  
 {  
 // Проверка наличия пользователя в базе данных  
 QSqlQuery query;  
 query.prepare("SELECT \* FROM Employees WHERE login = :login AND password = :password");  
 query.bindValue(":login", log);  
 query.bindValue(":password", pass);  
 if (!query.exec())  
 {  
 qDebug() << "Error searching for user:" << query.lastError().text();  
 return; // Выйти из функции при ошибке  
 }  
  
 // Проверка роли пользователя  
 if (query.next())  
 {  
 int employeeId = query.value("employee\_id").toInt(); // Получаем employee\_id из результата запроса  
 QString role = query.value("role").toString();  
 if (role == "chief")  
 {  
 // Открыть основное окно класса ChiefDesk  
 hide();  
 ChiefDesk \*chiefDesk = new ChiefDesk(db, employeeId);  
 chiefDesk->show();  
  
 }  
 else if (role == "employee")  
 {  
 // Открыть основное окно класса EmployeeDesk  
 hide();  
 EmployeeDesk \*employeeDesk = new EmployeeDesk(db, employeeId);  
 employeeDesk->show();  
 }  
 }  
 else  
 {  
 qDebug() << "User not found.";  
 }  
 }  
}

#include "chiefdesk.h"  
#include "ui\_chiefdesk.h"  
  
ChiefDesk::ChiefDesk(QSqlDatabase& \_db, int employeeId, QWidget \*parent) :  
 QMainWindow(parent), db(\_db), id(employeeId),  
 ui(new Ui::ChiefDesk)  
{  
 ui->setupUi(this);  
  
 connect(ui->viewProfileBtn, &QPushButton::clicked, this, &ChiefDesk::viewProfile);  
 connect(ui->viewEmployeesBtn, &QPushButton::clicked, this, &ChiefDesk::viewEmployees);  
 connect(ui->resControlBtn, &QPushButton::clicked, this, &ChiefDesk::viewResources);  
 connect(ui->viewTasksBtn, &QPushButton::clicked, this, &ChiefDesk::viewTasks);  
}  
  
ChiefDesk::~ChiefDesk()  
{  
 delete ui;  
}  
  
void ChiefDesk::viewProfile()  
{  
 // Запрос для получения данных профиля сотрудника  
 QSqlQuery profileQuery;  
 profileQuery.prepare("SELECT Employees.login, Employees.role, Companies.company\_name, Companies.company\_code "  
 "FROM Employees "  
 "JOIN Companies ON Employees.company\_id = Companies.company\_id "  
 "WHERE employee\_id = :id");  
 profileQuery.bindValue(":id", id);  
 if (!profileQuery.exec())  
 {  
 qDebug() << "Error retrieving profile data:" << profileQuery.lastError().text();  
 return;  
 }  
  
 // Извлечение данных профиля сотрудника  
 QString login, role, companyName, invitationCode;  
 if (profileQuery.next())  
 {  
 login = profileQuery.value("login").toString();  
 role = profileQuery.value("role").toString();  
 companyName = profileQuery.value("company\_name").toString();  
 invitationCode = profileQuery.value("company\_code").toString();  
 }  
 else  
 {  
 qDebug() << "Profile data not found for employee with ID:" << id;  
 return;  
 }  
  
 // Запрос для подсчета количества сотрудников, принадлежащих той же компании  
 QSqlQuery countQuery;  
 countQuery.prepare("SELECT COUNT(\*) AS amountOfEmployees "  
 "FROM Employees "  
 "WHERE company\_id = (SELECT company\_id FROM Companies WHERE company\_name = :companyName)");  
 countQuery.bindValue(":companyName", companyName);  
 if (!countQuery.exec())  
 {  
 qDebug() << "Error counting employees:" << countQuery.lastError().text();  
 return;  
 }  
  
 // Извлечение количества сотрудников  
 QString amountOfEmployees;  
 if (countQuery.next())  
 {  
 amountOfEmployees = countQuery.value("amountOfEmployees").toString();  
 }  
 else  
 {  
 qDebug() << "Error: Count query returned no results";  
 return;  
 }  
  
 // Вывод в консоль  
 qDebug() << "Profile data retrieved successfully:";  
 qDebug() << "Login:" << login;  
 qDebug() << "Role:" << role;  
 qDebug() << "Company Name:" << companyName;  
 qDebug() << "Invitation Code:" << invitationCode;  
 qDebug() << "Amount of Employees in the Same Company:" << amountOfEmployees;  
  
 QList<QString> profileData;  
 profileData.append(login);  
 profileData.append(role);  
 profileData.append(companyName);  
 profileData.append(invitationCode);  
 profileData.append(amountOfEmployees);  
  
 Profile prof (profileData);  
 prof.setModal(true);  
 prof.exec();  
}  
  
void ChiefDesk::viewEmployees()  
{  
 // Запрос для получения company\_id сотрудника по его id  
 QSqlQuery companyIdQuery;  
 companyIdQuery.prepare("SELECT company\_id FROM Employees WHERE employee\_id = :id");  
 companyIdQuery.bindValue(":id", id);  
 if (!companyIdQuery.exec())  
 {  
 qDebug() << "Error retrieving company\_id:" << companyIdQuery.lastError().text();  
 return;  
 }  
  
 QString companyId; // Переменная для хранения company\_id сотрудника  
 if (companyIdQuery.next())  
 {  
 companyId = companyIdQuery.value("company\_id").toString();  
 }  
 else  
 {  
 qDebug() << "Error: company\_id not found for employee with ID:" << id;  
 return;  
 }  
  
 // Открывает окно сотрудников с полученным company\_id  
 EmployeesView empl(db, companyId);  
 empl.setModal(true);  
 empl.exec();  
}  
  
void ChiefDesk::viewResources()  
{  
 // Запрос для получения company\_id сотрудника по его id  
 QSqlQuery companyIdQuery;  
 companyIdQuery.prepare("SELECT company\_id FROM Employees WHERE employee\_id = :id");  
 companyIdQuery.bindValue(":id", id);  
 if (!companyIdQuery.exec())  
 {  
 qDebug() << "Error retrieving company\_id:" << companyIdQuery.lastError().text();  
 return;  
 }  
  
 QString companyId; // Переменная для хранения company\_id сотрудника  
 if (companyIdQuery.next())  
 {  
 companyId = companyIdQuery.value("company\_id").toString();  
 }  
 else  
 {  
 qDebug() << "Error: company\_id not found for employee with ID:" << id;  
 return;  
 }  
  
 // Открывает окно сотрудников с полученным company\_id  
 ResourcesView res(db, companyId);  
 res.setModal(true);  
 res.exec();  
}  
  
void ChiefDesk::viewTasks()  
{  
 // Запрос для получения company\_id сотрудника по его id  
 QSqlQuery companyIdQuery;  
 companyIdQuery.prepare("SELECT company\_id FROM Employees WHERE employee\_id = :id");  
 companyIdQuery.bindValue(":id", id);  
 if (!companyIdQuery.exec())  
 {  
 qDebug() << "Error retrieving company\_id:" << companyIdQuery.lastError().text();  
 return;  
 }  
  
 QString companyId; // Переменная для хранения company\_id сотрудника  
 if (companyIdQuery.next())  
 {  
 companyId = companyIdQuery.value("company\_id").toString();  
 }  
 else  
 {  
 qDebug() << "Error: company\_id not found for employee with ID:" << id;  
 return;  
 }  
  
 // Открывает окно сотрудников с полученным company\_id  
 TasksView tasks(db, companyId);  
 tasks.setModal(true);  
 tasks.exec();  
}

#include "employeedesk.h"  
#include "ui\_employeedesk.h"  
  
EmployeeDesk::EmployeeDesk(QSqlDatabase& \_db, int employeeId, QWidget \*parent) :  
 QMainWindow(parent), db(\_db), id(employeeId),  
 ui(new Ui::EmployeeDesk)  
{  
 ui->setupUi(this);  
  
 connect(ui->viewProfileBtn, &QPushButton::clicked, this, &EmployeeDesk::viewProfile);  
 connect(ui->viewEmployeesBtn, &QPushButton::clicked, this, &EmployeeDesk::viewEmployees);  
 connect(ui->resControlBtn, &QPushButton::clicked, this, &EmployeeDesk::viewResources);  
 connect(ui->viewTasksBtn, &QPushButton::clicked, this, &EmployeeDesk::viewTasks);  
}  
  
EmployeeDesk::~EmployeeDesk()  
{  
 delete ui;  
}  
  
void EmployeeDesk::viewProfile()  
{  
 // Запрос для получения данных профиля сотрудника  
 QSqlQuery profileQuery;  
 profileQuery.prepare("SELECT Employees.login, Employees.role, Companies.company\_name, Companies.company\_code "  
 "FROM Employees "  
 "JOIN Companies ON Employees.company\_id = Companies.company\_id "  
 "WHERE employee\_id = :id");  
 profileQuery.bindValue(":id", id);  
 if (!profileQuery.exec())  
 {  
 qDebug() << "Error retrieving profile data:" << profileQuery.lastError().text();  
 return;  
 }  
  
 // Извлечение данных профиля сотрудника  
 QString login, role, companyName, invitationCode;  
 if (profileQuery.next())  
 {  
 login = profileQuery.value("login").toString();  
 role = profileQuery.value("role").toString();  
 companyName = profileQuery.value("company\_name").toString();  
 invitationCode = profileQuery.value("company\_code").toString();  
 }  
 else  
 {  
 qDebug() << "Profile data not found for employee with ID:" << id;  
 return;  
 }  
  
 // Запрос для подсчета количества сотрудников, принадлежащих той же компании  
 QSqlQuery countQuery;  
 countQuery.prepare("SELECT COUNT(\*) AS amountOfEmployees "  
 "FROM Employees "  
 "WHERE company\_id = (SELECT company\_id FROM Companies WHERE company\_name = :companyName)");  
 countQuery.bindValue(":companyName", companyName);  
 if (!countQuery.exec())  
 {  
 qDebug() << "Error counting employees:" << countQuery.lastError().text();  
 return;  
 }  
  
 // Извлечение количества сотрудников  
 QString amountOfEmployees;  
 if (countQuery.next())  
 {  
 amountOfEmployees = countQuery.value("amountOfEmployees").toString();  
 }  
 else  
 {  
 qDebug() << "Error: Count query returned no results";  
 return;  
 }  
  
 // Вывод в консоль  
 qDebug() << "Profile data retrieved successfully:";  
 qDebug() << "Login:" << login;  
 qDebug() << "Role:" << role;  
 qDebug() << "Company Name:" << companyName;  
 qDebug() << "Invitation Code:" << invitationCode;  
 qDebug() << "Amount of Employees in the Same Company:" << amountOfEmployees;  
  
 QList<QString> profileData;  
 profileData.append(login);  
 profileData.append(role);  
 profileData.append(companyName);  
 profileData.append(invitationCode);  
 profileData.append(amountOfEmployees);  
  
 Profile prof (profileData);  
 prof.setModal(true);  
 prof.exec();  
}  
  
void EmployeeDesk::viewEmployees()  
{  
 // Запрос для получения company\_id сотрудника по его id  
 QSqlQuery companyIdQuery;  
 companyIdQuery.prepare("SELECT company\_id FROM Employees WHERE employee\_id = :id");  
 companyIdQuery.bindValue(":id", id);  
 if (!companyIdQuery.exec())  
 {  
 qDebug() << "Error retrieving company\_id:" << companyIdQuery.lastError().text();  
 return;  
 }  
  
 QString companyId; // Переменная для хранения company\_id сотрудника  
 if (companyIdQuery.next())  
 {  
 companyId = companyIdQuery.value("company\_id").toString();  
 }  
 else  
 {  
 qDebug() << "Error: company\_id not found for employee with ID:" << id;  
 return;  
 }  
  
 // Открывает окно сотрудников с полученным company\_id  
 EmployeesView empl(db, companyId);  
 empl.setModal(true);  
 empl.exec();  
}  
  
void EmployeeDesk::viewResources()  
{  
 // Запрос для получения company\_id сотрудника по его id  
 QSqlQuery companyIdQuery;  
 companyIdQuery.prepare("SELECT company\_id FROM Employees WHERE employee\_id = :id");  
 companyIdQuery.bindValue(":id", id);  
 if (!companyIdQuery.exec())  
 {  
 qDebug() << "Error retrieving company\_id:" << companyIdQuery.lastError().text();  
 return;  
 }  
  
 QString companyId; // Переменная для хранения company\_id сотрудника  
 if (companyIdQuery.next())  
 {  
 companyId = companyIdQuery.value("company\_id").toString();  
 }  
 else  
 {  
 qDebug() << "Error: company\_id not found for employee with ID:" << id;  
 return;  
 }  
  
 // Открывает окно сотрудников с полученным company\_id  
 ResourcesView res(db, companyId);  
 res.setModal(true);  
 res.exec();  
}  
  
void EmployeeDesk::viewTasks()  
{  
 // Запрос для получения company\_id сотрудника по его id  
 QSqlQuery companyIdQuery;  
 companyIdQuery.prepare("SELECT company\_id FROM Employees WHERE employee\_id = :id");  
 companyIdQuery.bindValue(":id", id);  
 if (!companyIdQuery.exec())  
 {  
 qDebug() << "Error retrieving company\_id:" << companyIdQuery.lastError().text();  
 return;  
 }  
  
 QString companyId; // Переменная для хранения company\_id сотрудника  
 if (companyIdQuery.next())  
 {  
 companyId = companyIdQuery.value("company\_id").toString();  
 }  
 else  
 {  
 qDebug() << "Error: company\_id not found for employee with ID:" << id;  
 return;  
 }  
  
 QString employeeId = QString::number(id);  
  
 // Открывает окно сотрудников с полученным company\_id  
 TasksViewForEmployee tasks(db, companyId, employeeId);  
 tasks.setModal(true);  
 tasks.exec();  
}

#include "profile.h"  
#include "ui\_profile.h"  
  
Profile::Profile(QList<QString> \_profileData, QWidget \*parent) :  
 QDialog(parent), profileData(\_profileData),  
 ui(new Ui::Profile)  
{  
 ui->setupUi(this);  
  
 // Обновляем информацию профиля:  
 QString outputInfo = "Profile name: " + profileData[0] + "\nAccess rights: " + profileData[1] +  
 "\nCompany name: " + profileData[2] + "\nAmount of employees: " + profileData[4];  
 ui->infoLabel->setText(outputInfo);  
  
 connect(ui->getInvCodeBtn, &QPushButton::clicked, this, &Profile::getInvCode);  
 connect(ui->backBtn, &QPushButton::clicked, this, &Profile::back);  
}  
  
Profile::~Profile()  
{  
 QList<QTimer\*> timers = this->findChildren<QTimer\*>();  
 for (QTimer\* timer : timers) {  
 timer->stop();  
 }  
 delete ui;  
}  
  
void Profile::getInvCode()  
{  
 QString role = profileData[1];  
 QString code = profileData[3];  
  
 if (role == "chief")  
 {  
 QClipboard \*clipboard = QApplication::clipboard();  
 clipboard->setText(code);  
 ui->getInvCodeBtn->setText("Copied!");  
 QTimer::singleShot(1000, [&]() {  
 ui->getInvCodeBtn->setText("Get invitation\ncode");  
 });  
 }  
 else if (role == "employee")  
 {  
 ui->getInvCodeBtn->setText("No access!");  
 QTimer::singleShot(1000, [&]() {  
 ui->getInvCodeBtn->setText("Get invitation\ncode");  
 });  
 }  
}  
  
void Profile::back()  
{  
 close();  
}

#include "employeesview.h"  
#include "ui\_employeesview.h"  
  
EmployeesView::EmployeesView(QSqlDatabase& \_db, QString \_companyId, QWidget \*parent) :  
 QDialog(parent), db(\_db), companyId(\_companyId),  
 ui(new Ui::EmployeesView)  
{  
 ui->setupUi(this);  
  
 // Вывод таблицы сотрудников  
 modelEmpl = new QSqlTableModel(this, db);  
 modelEmpl->setTable("Employees");  
  
 // Фильтрация по companyId  
 modelEmpl->setFilter("company\_id = " + companyId);  
  
 // Скрытие столбца password  
 modelEmpl->removeColumn(modelEmpl->fieldIndex("password"));  
  
 modelEmpl->select();  
  
 ui->employeesTableView->setModel(modelEmpl);  
  
 // Вывод таблицы задач  
 modelTasks = new QSqlTableModel(this, db);  
 modelTasks->setTable("Tasks");  
  
 // Фильтрация по companyId  
 modelTasks->setFilter("company\_id = " + companyId);  
  
 modelTasks->select();  
  
 ui->tasksTableView->setModel(modelTasks);  
  
 connect(ui->backBtn, &QPushButton::clicked, this, &EmployeesView::back);  
}  
  
EmployeesView::~EmployeesView()  
{  
 delete ui;  
 delete modelEmpl;  
 delete modelTasks;  
}  
  
void EmployeesView::back()  
{  
 close();  
}

#include "resourcesview.h"  
#include "ui\_resourcesview.h"  
  
ResourcesView::ResourcesView(QSqlDatabase& \_db, QString \_companyId, QWidget \*parent) :  
 QDialog(parent), db(\_db), companyId(\_companyId),  
 ui(new Ui::ResourcesView)  
{  
 ui->setupUi(this);  
  
 // Вывод таблицы задач  
 modelRes = new QSqlTableModel(this, db);  
 modelRes->setTable("Resources");  
  
 // Фильтрация по companyId  
 modelRes->setFilter("company\_id = " + companyId);  
  
 modelRes->select();  
  
 ui->resTableView->setModel(modelRes);  
  
 connect(ui->backBtn, &QPushButton::clicked, this, &ResourcesView::back);  
 connect(ui->addBtn, &QPushButton::clicked, this, &ResourcesView::addRes);  
 connect(ui->removeBtn, &QPushButton::clicked, this, &ResourcesView::removeRes);  
}  
  
ResourcesView::~ResourcesView()  
{  
 delete ui;  
}  
  
void ResourcesView::back()  
{  
 close();  
}  
  
void ResourcesView::addRes()  
{  
 // Получаем данные из интерфейса  
 QString resName = ui->resNameLineEdit->text();  
 QString amountStr = ui->amountLineEdit->text();  
  
 // Проверяем, чтобы оба поля были заполнены  
 if (resName.isEmpty() || amountStr.isEmpty())  
 {  
 qDebug() << "Resource name and amount are required.";  
 return;  
 }  
  
 bool success = false;  
  
 // Проверяем, существует ли ресурс с таким именем в базе данных  
 QSqlQuery query;  
 query.prepare("SELECT \* FROM Resources WHERE res\_name = :resName");  
 query.bindValue(":resName", resName);  
 if (query.exec() && query.next())  
 {  
 // Ресурс существует, обновляем его количество  
 int currentAmount = query.value("res\_amount").toInt();  
 int amountToAdd = amountStr.toInt();  
 int newAmount = currentAmount + amountToAdd;  
  
 QSqlQuery updateQuery;  
 updateQuery.prepare("UPDATE Resources SET res\_amount = :newAmount WHERE res\_name = :resName");  
 updateQuery.bindValue(":newAmount", newAmount);  
 updateQuery.bindValue(":resName", resName);  
 success = updateQuery.exec();  
 }  
 else  
 {  
 // Ресурс не существует, добавляем его в базу данных  
 QSqlQuery insertQuery;  
 insertQuery.prepare("INSERT INTO Resources (res\_name, res\_amount, company\_id) VALUES (:resName, :amount, :companyId)");  
 insertQuery.bindValue(":resName", resName);  
 insertQuery.bindValue(":amount", amountStr.toInt());  
 insertQuery.bindValue(":companyId", companyId);  
 success = insertQuery.exec();  
 }  
  
 if (success)  
 {  
 qDebug() << "Resource added/updated successfully.";  
 }  
 else  
 {  
 qDebug() << "Error adding/updating resource:" << query.lastError().text();  
 }  
}  
  
void ResourcesView::removeRes()  
{  
 // Получаем данные из интерфейса  
 QString resName = ui->resNameLineEdit->text();  
 QString amountStr = ui->amountLineEdit->text();  
  
 // Проверяем, чтобы оба поля были заполнены  
 if (resName.isEmpty() || amountStr.isEmpty())  
 {  
 qDebug() << "Resource name and amount are required.";  
 return;  
 }  
  
 // Получаем текущее количество ресурсов из базы данных  
 QSqlQuery query;  
 query.prepare("SELECT res\_amount FROM Resources WHERE res\_name = :resName");  
 query.bindValue(":resName", resName);  
  
 if (!query.exec())  
 {  
 qDebug() << "Error retrieving resource information:" << query.lastError().text();  
 return;  
 }  
  
 if (!query.next())  
 {  
 qDebug() << "Resource not found.";  
 return;  
 }  
  
 int currentAmount = query.value("res\_amount").toInt();  
 int amountToRemove = amountStr.toInt();  
  
 // Проверяем, чтобы количество ресурсов после удаления не стало меньше нуля  
 int newAmount = currentAmount - amountToRemove;  
 if (newAmount < 0)  
 {  
 newAmount = 0;  
 }  
  
 // Обновляем количество ресурсов в базе данных  
 QSqlQuery updateQuery;  
 updateQuery.prepare("UPDATE Resources SET res\_amount = :newAmount WHERE res\_name = :resName");  
 updateQuery.bindValue(":newAmount", newAmount);  
 updateQuery.bindValue(":resName", resName);  
  
 if (!updateQuery.exec())  
 {  
 qDebug() << "Error updating resource amount:" << updateQuery.lastError().text();  
 return;  
 }  
  
 qDebug() << "Resource amount updated successfully.";  
}

#include "tasksview.h"  
#include "ui\_tasksview.h"  
  
TasksView::TasksView(QSqlDatabase& \_db, QString \_companyId, QWidget \*parent) :  
 QDialog(parent), db(\_db), companyId(\_companyId),  
 ui(new Ui::TasksView)  
{  
 ui->setupUi(this);  
  
 // Вывод таблицы задач  
 modelTasks = new QSqlTableModel(this, db);  
 modelTasks->setTable("Tasks");  
  
 // Фильтрация по companyId  
 modelTasks->setFilter("company\_id = " + companyId);  
  
 modelTasks->select();  
  
 ui->tasksTableView->setModel(modelTasks);  
  
 connect(ui->backBtn, &QPushButton::clicked, this, &TasksView::back);  
 connect(ui->addBtn, &QPushButton::clicked, this, &TasksView::addTask);  
 connect(ui->getReportBtn, &QPushButton::clicked, this, &TasksView::getReport);  
}  
  
TasksView::~TasksView()  
{  
 delete ui;  
}  
  
void TasksView::back()  
{  
 close();  
}  
  
void TasksView::addTask()  
{  
 // Получаем данные из интерфейса  
 QString taskName = ui->taskNameLineEdit->text();  
 QString empId = ui->idLineEdit->text();  
  
 // Проверяем, чтобы оба поля были заполнены  
 if (taskName.isEmpty() || empId.isEmpty())  
 {  
 qDebug() << "Task name and employee ID are required.";  
 return;  
 }  
  
 // Создаем задачу со статусом 0 (не выполнена)  
 QSqlQuery query;  
 query.prepare("INSERT INTO Tasks (task\_name, status, employee\_id, company\_id) VALUES (:taskName, 0, :employeeId, :companyId)");  
 query.bindValue(":taskName", taskName);  
 query.bindValue(":employeeId", empId.toInt());  
 query.bindValue(":companyId", companyId);  
  
 if (!query.exec()) {  
 qDebug() << "Error adding task:" << query.lastError().text();  
 return;  
 }  
  
 qDebug() << "Task added successfully.";  
  
 // Обновляем task\_id в таблице Employees  
 QSqlQuery updateQuery;  
 updateQuery.prepare("UPDATE Employees SET task\_id = :taskId WHERE employee\_id = :employeeId");  
 updateQuery.bindValue(":taskId", query.lastInsertId().toInt());  
 updateQuery.bindValue(":employeeId", empId.toInt());  
  
 if (!updateQuery.exec()) {  
 qDebug() << "Error updating employee task ID:" << updateQuery.lastError().text();  
 return;  
 }  
  
 qDebug() << "Employee task ID updated successfully.";  
}  
  
void TasksView::getReport()  
{  
 QString taskId = ui->taskIdLineEdit->text();  
 GetReport report(db, taskId);  
 report.setModal(true);  
 report.exec();  
}

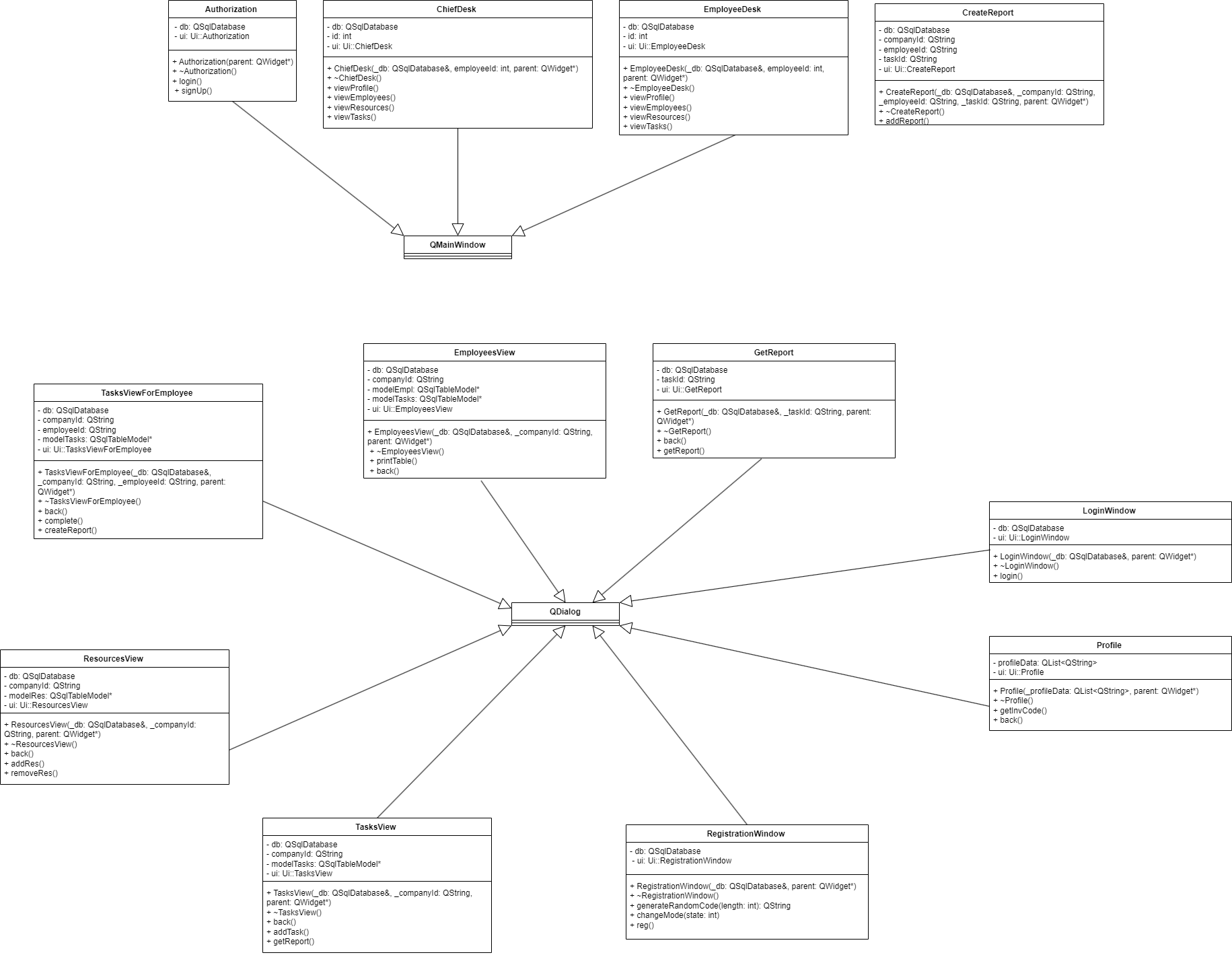
#include "tasksviewforemployee.h"  
#include "ui\_tasksviewforemployee.h"  
  
TasksViewForEmployee::TasksViewForEmployee(QSqlDatabase& \_db, QString \_companyId, QString \_employeeId, QWidget \*parent) :  
 QDialog(parent), db(\_db), companyId(\_companyId), employeeId(\_employeeId),  
 ui(new Ui::TasksViewForEmployee)  
{  
 ui->setupUi(this);  
  
 // Вывод таблицы задач  
 modelTasks = new QSqlTableModel(this, db);  
 modelTasks->setTable("Tasks");  
  
 // Фильтрация по companyId  
 modelTasks->setFilter("company\_id = " + companyId);  
  
 modelTasks->select();  
  
 ui->tasksTableView->setModel(modelTasks);  
  
 connect(ui->backBtn, &QPushButton::clicked, this, &TasksViewForEmployee::back);  
 connect(ui->completeBtn, &QPushButton::clicked, this, &TasksViewForEmployee::complete);  
 connect(ui->reportBtn, &QPushButton::clicked, this, &TasksViewForEmployee::createReport);  
}  
  
TasksViewForEmployee::~TasksViewForEmployee()  
{  
 delete ui;  
}  
  
void TasksViewForEmployee::back()  
{  
 close();  
}  
  
void TasksViewForEmployee::complete()  
{  
 QString taskId = ui->idLineEdit->text();  
  
 // Проверяем, что taskId не пустой  
 if (taskId.isEmpty())  
 {  
 qDebug() << "Task ID is required.";  
 return;  
 }  
  
 // Проверяем, что в таблице Tasks есть задача с указанным taskId и employeeId  
 QSqlQuery query;  
 query.prepare("SELECT \* FROM Tasks WHERE task\_id = :taskId AND employee\_id = :employeeId");  
 query.bindValue(":taskId", taskId);  
 query.bindValue(":employeeId", employeeId);  
  
 if (!query.exec())  
 {  
 qDebug() << "Error checking task information:" << query.lastError().text();  
 return;  
 }  
  
 if (!query.next())  
 {  
 qDebug() << "Task not found for the current employee.";  
 return;  
 }  
  
 // Обновляем значение столбца status на 1  
 QSqlQuery updateQuery;  
 updateQuery.prepare("UPDATE Tasks SET status = 1 WHERE task\_id = :taskId");  
 updateQuery.bindValue(":taskId", taskId);  
  
 if (!updateQuery.exec())  
 {  
 qDebug() << "Error updating task status:" << updateQuery.lastError().text();  
 return;  
 }  
  
 qDebug() << "Task completed successfully.";  
}  
  
void TasksViewForEmployee::createReport()  
{  
 QString taskId = ui->idLineEdit->text();  
 CreateReport report(db, companyId, employeeId, taskId);  
 report.setModal(true);  
 report.exec();  
}

#include "getreport.h"  
#include "ui\_getreport.h"  
  
GetReport::GetReport(QSqlDatabase& \_db, QString \_taskId, QWidget \*parent) :  
 QDialog(parent), db(\_db), taskId(\_taskId),  
 ui(new Ui::GetReport)  
{  
 ui->setupUi(this);  
  
 connect(ui->printBtn, &QPushButton::clicked, this, &GetReport::getReport);  
 connect(ui->backBtn, &QPushButton::clicked, this, &GetReport::back);  
}  
  
GetReport::~GetReport()  
{  
 delete ui;  
}  
  
void GetReport::back()  
{  
 close();  
}  
  
void GetReport::getReport()  
{  
 // Выполнить запрос к базе данных для получения текста из поля report для указанного taskId  
 QSqlQuery query;  
 query.prepare("SELECT report FROM Tasks WHERE task\_id = :taskId");  
 query.bindValue(":taskId", taskId);  
  
 if (!query.exec())  
 {  
 qDebug() << "Error fetching report from database:" << query.lastError().text();  
 return;  
 }  
  
 // Проверить, есть ли данные  
 if (query.next())  
 {  
 // Получить текст из поля report  
 QString reportText = query.value(0).toString();  
  
 // Установить текст в QTextEdit  
 ui->reportTextEdit->setPlainText(reportText);  
 }  
 else  
 {  
 qDebug() << "No report found for the specified task ID.";  
 }  
}

#include "createreport.h"  
#include "ui\_createreport.h"  
  
CreateReport::CreateReport(QSqlDatabase& \_db, QString \_companyId, QString \_employeeId, QString \_taskId, QWidget \*parent) :  
 QDialog(parent), db(\_db), companyId(\_companyId), employeeId(\_employeeId), taskId(\_taskId),  
 ui(new Ui::CreateReport)  
{  
 ui->setupUi(this);  
  
 connect(ui->submitBtn, &QPushButton::clicked, this, &CreateReport::addReport);  
}  
  
CreateReport::~CreateReport()  
{  
 delete ui;  
}  
  
void CreateReport::addReport()  
{  
 QString reportText = ui->reportTextEdit->toPlainText();  
  
 // Проверяем, что в таблице Tasks есть задача с указанным taskId и employeeId  
 QSqlQuery query;  
 query.prepare("SELECT \* FROM Tasks WHERE task\_id = :taskId AND employee\_id = :employeeId");  
 query.bindValue(":taskId", taskId);  
 query.bindValue(":employeeId", employeeId);  
  
 if (!query.exec())  
 {  
 qDebug() << "Error checking task information:" << query.lastError().text();  
 return;  
 }  
  
 if (!query.next())  
 {  
 qDebug() << "Task not found for the current employee.";  
 return;  
 }  
  
 // Обновляем значение столбца report на reportText  
 QSqlQuery updateQuery;  
 updateQuery.prepare("UPDATE Tasks SET report = :reportText WHERE task\_id = :taskId");  
 updateQuery.bindValue(":reportText", reportText);  
 updateQuery.bindValue(":taskId", taskId);  
  
 if (!updateQuery.exec())  
 {  
 qDebug() << "Error updating task report:" << updateQuery.lastError().text();  
 return;  
 }  
  
 qDebug() << "Report added/updated successfully.";  
 close();  
}

#include "authorization.h"  
  
#include <QApplication>  
  
int main(int argc, char \*argv[])  
{  
 QApplication a(argc, argv);  
 Authorization w;  
 w.show();  
 return a.exec();  
}

# UML-диаграмма классов



# Демонстрация работы

https://www.youtube.com/watch?v=lKbh\_Bkce7Q