//main.cpp

#include "Headers/mainwindow.h"

#include <QApplication>

#include <QFile>

// main файл для запуска самого приложения и главного окна приложения.

// Вызывается конструктор главного окна приложения

int main(int argc, char \*argv[])

{

QApplication a(argc, argv);

MainWindow w;

w.show();

return a.exec();

}

//addrecipewindow.h

#ifndef ADDRECIPEWINDOW\_H

#define ADDRECIPEWINDOW\_H

#include "recipemanager.h"

#include "recipewindowbase.h"

class AddRecipeWindow : public RecipeWindowBase

{

Q\_OBJECT

public:

explicit AddRecipeWindow(QWidget \*parent = nullptr);

private slots:

virtual void saveRecipe() override;

};

#endif // ADDRECIPEWINDOW\_H

//addrecipewindow.cpp

#include "Headers/addrecipewindow.h"

// Конструктор для вызова окна добавления рецепта

// Данный класс унаследован от класса RecipeWindowBase, следовательно

// в конструкторе автоматически вызывается загрузка интерфейса.

AddRecipeWindow::AddRecipeWindow(QWidget \*parent)

: RecipeWindowBase(parent)

{

setWindowTitle("Добавить рецепт");

}

// Переопределенная виртуальная функция для сохранения рецепта в базу данных

void AddRecipeWindow::saveRecipe()

{

RecipeManager recipeManager;

Recipe recipe;

QString name = nameLineEdit->text();

QString description = descriptionTextEdit->toPlainText();

QString ingredients = ingredientsTextEdit->toPlainText();

QString instruction = instructionTextEdit->toPlainText();

int prepTime = prepTimeSpinBox->value();

int servings = servingsSpinBox->value();

QString category = categoryComboBox->currentText();

QString kitchen = kitchenComboBox->currentText();

if (name.isEmpty() || prepTime == 0 || servings == 0) {

QMessageBox::warning(this, "Ошибка", "Заполните все обязательные поля!");

return;

}

recipe.setName(name);

recipe.setDescription(description);

recipe.setIngredients(ingredients);

recipe.setInstruction(instruction);

recipe.setPrepTime(prepTime);

recipe.setServings(servings);

recipe.setCategory(category);

recipe.setKitchen(kitchen);

recipeManager.addRecipeToDatabase(recipe, selectedPhotoPath);

accept();

}

//comboboxloader.h

#ifndef COMBOBOXLOADER\_H

#define COMBOBOXLOADER\_H

#include <QComboBox>

#include <QFile>

#include <QTextStream>

class ComboBoxLoader

{

public:

ComboBoxLoader() = default;

void loadComboBoxItems(QComboBox\* comboBox, const QString& filePath);

void saveComboBoxItems(QComboBox\* comboBox, const QString& filePath);

};

#endif // COMBOBOXLOADER\_H

//comboboxloader.cpp

#include "Headers/comboboxloader.h"

// Функция для выгрузки данных из .txt файла в виджет ComboBox

void ComboBoxLoader::loadComboBoxItems(QComboBox\* comboBox, const QString& filePath) {

QFile file(filePath);

if (file.open(QIODevice::ReadOnly | QIODevice::Text)) {

QTextStream in(&file);

while (!in.atEnd()) {

QString line = in.readLine();

comboBox->addItem(line);

}

file.close();

}

}

// Функция для сохранения данных в .txt файл из виджета ComboBox

void ComboBoxLoader::saveComboBoxItems(QComboBox\* comboBox, const QString& filePath) {

QFile file(filePath);

if (file.open(QIODevice::WriteOnly | QIODevice::Text)) {

QTextStream out(&file);

for (int i = 0; i < comboBox->count(); ++i) {

out << comboBox->itemText(i) << "\n";

}

file.close();

}

}

//databasemanager.h

#ifndef DATABASEMANAGER\_H

#define DATABASEMANAGER\_H

#include <QSqlDatabase>

#include <QSqlQuery>

#include <QSqlTableModel>

class DatabaseManager

{

private:

QSqlDatabase db;

QString databasePath;

public:

DatabaseManager(const QString &databasePath);

~DatabaseManager();

bool openDatabase();

void closeDatabase();

QSqlTableModel\* createRecipeModel();

QSqlTableModel\* createRecipeModelForSearch();

QSqlQuery executeQuery(const QString &queryString, const QMap<QString, QVariant> &bindValues);

};

#endif // DATABASEMANAGER\_H

//databasemanager.cpp

#include "Headers/databasemanager.h"

// Конструктор класса

DatabaseManager::DatabaseManager(const QString &databasePath) : databasePath(databasePath) {}

// Деструктор. Закрывает базу данных

DatabaseManager::~DatabaseManager() {

closeDatabase();

}

// Функция для открытия базы данных типа "SQLite"

bool DatabaseManager::openDatabase() {

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

db.setDatabaseName(databasePath);

return db.open();

}

// Функция для закрытия базы данных.

// Используется в деструкторе

void DatabaseManager::closeDatabase() {

db.close();

}

// Функция для настройки модели таблицы рецептов базы данных для отображения в приложении

QSqlTableModel\* DatabaseManager::createRecipeModel() {

QSqlTableModel \*model = new QSqlTableModel(nullptr, db);

model->setTable("RecipesInfo");

model->select();

model->setHeaderData(1, Qt::Horizontal, "Название рецепта", Qt::DisplayRole);

model->setHeaderData(2, Qt::Horizontal, "Краткое описание", Qt::DisplayRole);

model->setEditStrategy(QSqlTableModel::OnManualSubmit);

return model;

}

// Функция для настройки модели таблицы найденных рецептов базы данных для отображения в приложении

QSqlTableModel\* DatabaseManager::createRecipeModelForSearch() {

QSqlTableModel \*model = new QSqlTableModel(nullptr, db);

model->setTable("RecipesInfo");

model->setFilter("1 = 0");

model->select();

model->setHeaderData(1, Qt::Horizontal, "Название рецепта", Qt::DisplayRole);

model->setHeaderData(2, Qt::Horizontal, "Краткое описание", Qt::DisplayRole);

model->setEditStrategy(QSqlTableModel::OnManualSubmit);

return model;

}

// Функция для реализации SQL-запроса.

QSqlQuery DatabaseManager::executeQuery(const QString &queryString, const QMap<QString, QVariant> &bindValues) {

QSqlQuery query;

query.prepare(queryString);

for (auto it = bindValues.constBegin(); it != bindValues.constEnd(); ++it) {

query.bindValue(it.key(), it.value());

}

query.exec();

return query;

}

//editrecipewindow.h

#ifndef EDITRECIPEWINDOW\_H

#define EDITRECIPEWINDOW\_H

#include "recipemanager.h"

#include "recipewindowbase.h"

class EditRecipeWindow : public RecipeWindowBase

{

Q\_OBJECT

public:

explicit EditRecipeWindow(int recipeId, QWidget\* parent = nullptr);

private slots:

virtual void saveRecipe() override;

void loadRecipeData();

private:

int recipeId;

};

#endif // EDITRECIPEWINDOW\_H

//editrecipewindow.cpp

#include "Headers/editrecipewindow.h"

// Конструктор для вызова окна редактирования рецепта

// Данный класс унаследован от класса RecipeWindowBase, следовательно

// в конструкторе автоматически вызывается загрузка интерфейса.

// Также выгружаем данные рецепта, который редактируем, во все поля,

// чтобы пользователь мог отредактировать уже существующие данные

EditRecipeWindow::EditRecipeWindow(int recipeId, QWidget\* parent)

: RecipeWindowBase(parent), recipeId(recipeId)

{

setWindowTitle("Редактировать рецепт");

EditRecipeWindow::loadRecipeData();

}

// Переопределенная виртуальная функция для сохранения рецепта в базу данных

void EditRecipeWindow::saveRecipe()

{

QSqlDatabase db = QSqlDatabase::database();

QSqlQuery query(db);

RecipeManager recipeManager;

QString name = nameLineEdit->text();

QString description = descriptionTextEdit->toPlainText();

QString ingredients = ingredientsTextEdit->toPlainText();

QString instruction = instructionTextEdit->toPlainText();

int prepTime = prepTimeSpinBox->value();

int servings = servingsSpinBox->value();

QString category = categoryComboBox->currentText();

QString kitchen = kitchenComboBox->currentText();

QByteArray newPhotoData;

QByteArray oldPhotoData;

Recipe recipe;

recipe.setName(name);

recipe.setDescription(description);

recipe.setIngredients(ingredients);

recipe.setInstruction(instruction);

recipe.setPrepTime(prepTime);

recipe.setServings(servings);

recipe.setCategory(category);

recipe.setKitchen(kitchen);

if (name.isEmpty() || prepTime == 0 || servings == 0) {

QMessageBox::warning(this, "Ошибка", "Заполните все обязательные поля!");

return;

}

if (!selectedPhotoPath.isEmpty()) {

QFile photoFile(selectedPhotoPath);

if (photoFile.open(QIODevice::ReadOnly)) {

newPhotoData = photoFile.readAll();

photoFile.close();

}

} else {

query.prepare("SELECT photo FROM RecipesInfo WHERE recipeId = :recipeId");

query.bindValue(":recipeId", recipeId);

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

oldPhotoData = query.value("photo").toByteArray();

}

}

query.prepare("UPDATE RecipesInfo SET photo = :photo WHERE recipeId = :recipeId");

if (newPhotoData.isEmpty()) {

query.bindValue(":photo", oldPhotoData);

} else {

query.bindValue(":photo", newPhotoData);

}

query.bindValue(":recipeId", recipeId);

if (!query.exec()) {

qDebug() << "Ошибка запроса:" << query.lastError().text();

db.rollback();

}

if (recipeManager.updateRecipe(recipeId, recipe)) {

accept();

}

}

// Функция для выгрузки данных рецепта, который редактируем, во все поля,

// чтобы пользователь мог отредактировать уже существующие данные

// Используется в конструкторе

void EditRecipeWindow::loadRecipeData()

{

QSqlDatabase db = QSqlDatabase::database();

QSqlQuery query(db);

query.prepare("SELECT name, description, ingredients, instruction, prepTime, servings, photo, category, kitchen FROM RecipesInfo WHERE recipeId = :recipeId");

query.bindValue(":recipeId", recipeId);

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

QString name = query.value("name").toString();

QString description = query.value("description").toString();

QString ingredients = query.value("ingredients").toString();

QString instruction = query.value("instruction").toString();

int prepTime = query.value("prepTime").toInt();

int servings = query.value("servings").toInt();

QString category = query.value("category").toString();

QString kitchen = query.value("kitchen").toString();

nameLineEdit->setText(name);

descriptionTextEdit->setPlainText(description);

ingredientsTextEdit->setPlainText(ingredients);

instructionTextEdit->setPlainText(instruction);

prepTimeSpinBox->setValue(prepTime);

servingsSpinBox->setValue(servings);

categoryComboBox->setCurrentText(category);

kitchenComboBox->setCurrentText(kitchen);

}

}

//mainwindow.h

#ifndef MAINWINDOW\_H

#define MAINWINDOW\_H

#include <QMainWindow>

#include "recipemanager.h"

#include "databasemanager.h"

#include "Headers/recipedetailswindow.h"

#include "Headers/addrecipewindow.h"

#include "Headers/editrecipewindow.h"

#include <QInputDialog>

QT\_BEGIN\_NAMESPACE

namespace Ui { class MainWindow; }

QT\_END\_NAMESPACE

class MainWindow : public QMainWindow

{

Q\_OBJECT

public:

MainWindow(QWidget \*parent = nullptr);

~MainWindow();

private slots:

void on\_action\_triggered();

void on\_buttonAdd\_clicked();

void on\_buttonDelete\_clicked();

void on\_buttonFind\_clicked();

void on\_tableView\_doubleClicked(const QModelIndex &index);

void on\_tableView1\_doubleClicked(const QModelIndex &index);

void on\_buttonEdit\_clicked();

void on\_tableView\_clicked();

void on\_tableView1\_clicked();

void on\_comboBox\_currentIndexChanged();

void on\_comboBox\_2\_currentIndexChanged();

void updateRecipeView(const QString& category, const QString& kitchen);

void on\_actionAddCategory\_triggered();

void on\_actionDeleteCategory\_triggered();

void on\_actionAddKitchen\_triggered();

void on\_actionDeleteKitchen\_triggered();

private:

Ui::MainWindow \*ui;

QSqlTableModel \*model;

QSqlTableModel \*modelForSearch;

int currentRow;

DatabaseManager \*dbManager;

ComboBoxLoader \*comboBoxLoader;

// ДЛЯ СБОРКИ ПРОЕКТА НА ДРУГОМ УСТРОЙСТВЕ НЕОБХОДИМО ПОМЕНЯТЬ ПУТИ

const QString categoryPath = "/Users/willygodx/Qt/qt projets/3sem-CulinaryBook-C++/DataBase/comboBoxCategoryInfo.txt";

const QString kitchenPath = "/Users/willygodx/Qt/qt projets/3sem-CulinaryBook-C++/DataBase/comboBoxKitchenInfo.txt";

};

#endif // MAINWINDOW\_H

//mainwindow.cpp

#include "Headers/mainwindow.h"

#include "ui\_mainwindow.h"

// Конструктор для главного окна приложения. Здесь находится установка базы данных

// и настройка моделей таблиц базы данных для отображения всех рецептов и

// для отображения найденных рецептов по поиску. Также выгрузка данных в виджеты ComboBox

// для категорий блюд и кухонь

// Элементы библиотеки QT настроены с помощью QT Designer. Настройки хранятся в ui файлах в билде

// ДЛЯ СБОРКИ ПРОЕКТА НА ДРУГОМ УСТРОЙСТВЕ НЕОБХОДИМО ПОМЕНЯТЬ ПУТЬ К БАЗЕ ДАННЫХ

MainWindow::MainWindow(QWidget \*parent)

: QMainWindow(parent)

, ui(new Ui::MainWindow)

{

ui->setupUi(this);

setWindowTitle("Кулинарный справочник");

dbManager = new DatabaseManager("/Users/willygodx/Qt/qt projets/3sem-CulinaryBook-C++/DataBase/RecipeNames.db");

if (dbManager->openDatabase()) {

model = dbManager->createRecipeModel();

ui->tableView->setModel(model);

ui->tableView->horizontalHeader()->setSectionResizeMode(QHeaderView::ResizeToContents);

ui->tableView->setColumnHidden(0, true);

ui->tableView->setColumnHidden(3, true);

ui->tableView->setColumnHidden(4, true);

ui->tableView->setColumnHidden(5, true);

ui->tableView->setColumnHidden(6, true);

ui->tableView->setColumnHidden(7, true);

ui->tableView->setColumnHidden(8, true);

ui->tableView->setColumnHidden(9, true);

ui->tableView->setSortingEnabled(true);

ui->tableView->setEditTriggers(QAbstractItemView::NoEditTriggers);

modelForSearch = dbManager->createRecipeModelForSearch();

ui->tableView1->setModel(modelForSearch);

ui->tableView1->horizontalHeader()->setSectionResizeMode(QHeaderView::ResizeToContents);

ui->tableView1->setColumnHidden(0, true);

ui->tableView1->setColumnHidden(3, true);

ui->tableView1->setColumnHidden(4, true);

ui->tableView1->setColumnHidden(5, true);

ui->tableView1->setColumnHidden(6, true);

ui->tableView1->setColumnHidden(7, true);

ui->tableView1->setColumnHidden(8, true);

ui->tableView1->setColumnHidden(9, true);

ui->tableView1->setSortingEnabled(true);

ui->tableView1->setEditTriggers(QAbstractItemView::NoEditTriggers);

} else {

ui->statusbar->showMessage("Error! Database not found!");

}

comboBoxLoader->loadComboBoxItems(ui->comboBox, categoryPath);

comboBoxLoader->loadComboBoxItems(ui->comboBox\_2, kitchenPath);

}

// Деструктор. Закрывает базу данных и удаляет интерфейс

MainWindow::~MainWindow()

{

dbManager->closeDatabase();

delete ui;

}

// Реализация триггера в меню для выхода из приложения

void MainWindow::on\_action\_triggered()

{

QApplication::quit();

}

// Реализация кнопки для удаления рецепта

// Здесь мы очищаем индексы после удаления рецепта, чтобы не было путаницы в индексах

// при удалении рецепта из списка найденных рецептов.

// Для удаления необходимо предварительно выбрать рецепт одиночным нажатием на него в таблице

void MainWindow::on\_buttonDelete\_clicked()

{

QModelIndex index = ui->tableView->currentIndex();

QModelIndex index1 = ui->tableView1->currentIndex();

if (index.isValid()) {

int recipeId = model->data(model->index(index.row(), 0)).toInt();

RecipeManager recipeManager(model);

if (recipeManager.deleteRecipeById(recipeId)) {

QMessageBox::information(this, "Успех", "Рецепт удален.");

} else {

QMessageBox::critical(this, "Ошибка", "Ошибка при удалении рецепта.");

}

} else if (index1.isValid()) {

int recipeId = modelForSearch->data(model->index(index1.row(), 0)).toInt();

RecipeManager recipeManager(modelForSearch);

if (recipeManager.deleteRecipeById(recipeId)) {

QMessageBox::information(this, "Успех", "Рецепт удален.");

} else {

QMessageBox::critical(this, "Ошибка", "Ошибка при удалении рецепта.");

}

}

ui->tableView1->selectionModel()->clear();

ui->tableView->selectionModel()->clear();

}

// Реализация кнопки поиска. Принимаем текст из виджета и выполняем

// SQL-запрос для фильтрации таблицы

void MainWindow::on\_buttonFind\_clicked()

{

QString searchText = ui->searchLineEdit->text().trimmed();

QString category = ui->comboBox->currentText();

QString kitchen = ui->comboBox\_2->currentText();

QString queryStr = "SELECT \* FROM RecipesInfo WHERE ";

if (!searchText.isEmpty()) {

queryStr += "(name LIKE :searchText)";

} else {

queryStr += "1";

}

if (category != "Все категории") {

queryStr += " AND category = :category";

}

if (kitchen != "Все кухни") {

queryStr += " AND kitchen = :kitchen";

}

QSqlQuery query;

query.prepare(queryStr);

if (!searchText.isEmpty()) {

query.bindValue(":searchText", "%" + searchText + "%");

}

if (category != "Все категории") {

query.bindValue(":category", category);

}

if (kitchen != "Все кухни") {

query.bindValue(":kitchen", kitchen);

}

if (query.exec()) {

modelForSearch->setQuery(query);

ui->tableView1->setModel(modelForSearch);

} else {

qDebug() << "SQL Query Error:" << query.lastError().text();

}

}

// Реализация двойного нажатия на рецепт в таблице всех рецептов. Открывается окно с деталями рецепта

void MainWindow::on\_tableView\_doubleClicked(const QModelIndex &index)

{

int recipeId = model->data(model->index(index.row(), 0)).toInt();

RecipeManager recipeManager(model);

Recipe recipe = recipeManager.getRecipeById(recipeId);

RecipeDetailsWindow \*detailsWindow = new RecipeDetailsWindow(recipe, this);

detailsWindow->exec();

}

// Реализация двойного нажатия на рецепт в таблице найденных рецептов. Открывается окно с деталями рецепта

void MainWindow::on\_tableView1\_doubleClicked(const QModelIndex &index)

{

int recipeId = modelForSearch->data(model->index(index.row(), 0)).toInt();

RecipeManager recipeManager(modelForSearch);

Recipe recipe = recipeManager.getRecipeById(recipeId);

RecipeDetailsWindow \*detailsWindow = new RecipeDetailsWindow(recipe, this);

detailsWindow->exec();

}

// Реализация кнопки редактирования рецепта. Открывается окно для редактирования

// существующего рецепта.

// Для редактирования необходимо предварительно выбрать рецепт одиночным нажатием на него в таблице

void MainWindow::on\_buttonEdit\_clicked()

{

RecipeManager recipeManager;

int recipeId = recipeManager.getRecipeIdFromIndex(ui->tableView->selectionModel()->currentIndex(), model);

int recipeIdForSearch = recipeManager.getRecipeIdFromIndex(ui->tableView1->selectionModel()->currentIndex(), modelForSearch);

if (recipeId != -1) {

EditRecipeWindow \*editWindow = new EditRecipeWindow(recipeId, this);

editWindow->exec();

} else if (recipeIdForSearch != -1) {

EditRecipeWindow \*editWindow = new EditRecipeWindow(recipeIdForSearch, this);

editWindow->exec();

} else {

QMessageBox::warning(this, "Ошибка", "Пожалуйста, выберите рецепт для редактирования.");

}

ui->tableView1->selectionModel()->clear();

ui->tableView->selectionModel()->clear();

}

// Реализация кнопки добавления нового рецепта. Открывается окно для добавления

void MainWindow::on\_buttonAdd\_clicked()

{

AddRecipeWindow \*addingWindow = new AddRecipeWindow(this);

addingWindow->exec();

model->select();

modelForSearch->select();

}

// Очищает указатель на выбор рецепта в таблице найденных рецептов

void MainWindow::on\_tableView\_clicked()

{

ui->tableView1->selectionModel()->clear();

}

// Очищает указатель на выбор рецепта в таблице всех рецептов

void MainWindow::on\_tableView1\_clicked()

{

ui->tableView->selectionModel()->clear();

}

// Реализация обновления таблицы рецептов при смене категории

void MainWindow::on\_comboBox\_currentIndexChanged()

{

QString category = ui->comboBox->currentText();

QString kitchen = ui->comboBox\_2->currentText();

updateRecipeView(category, kitchen);

}

// Реализация обновления таблицы рецептов при смене кухни

void MainWindow::on\_comboBox\_2\_currentIndexChanged()

{

QString kitchen = ui->comboBox\_2->currentText();

QString category = ui->comboBox->currentText();

updateRecipeView(category, kitchen);

}

// Функция для обновления таблицы рецептов при смене категории либо кухни.

// Используется выше в коде

void MainWindow::updateRecipeView(const QString& category, const QString& kitchen)

{

QString queryString = "SELECT \* FROM RecipesInfo WHERE ";

if (category == "Все категории" && kitchen == "Все кухни") {

queryString = "SELECT \* FROM RecipesInfo";

} else if (category == "Все категории") {

queryString += "kitchen = :kitchen";

} else if (kitchen == "Все кухни") {

queryString += "category = :category";

} else {

queryString += "category = :category AND kitchen = :kitchen";

}

QSqlQuery query;

query.prepare(queryString);

if (category != "Все категории") {

query.bindValue(":category", category);

}

if (kitchen != "Все кухни") {

query.bindValue(":kitchen", kitchen);

}

if (query.exec()) {

model->setQuery(query);

modelForSearch->setQuery(query);

modelForSearch->setFilter("1 = 0");

modelForSearch->select();

ui->tableView->setModel(model);

ui->tableView1->setModel(modelForSearch);

} else {

qDebug() << "SQL Query Error: " << query.lastError().text();

}

}

// Реализация триггера в меню для открытия диалогового окна при добавлении новой категории

void MainWindow::on\_actionAddCategory\_triggered()

{

bool ok;

QString newItem = QInputDialog::getText(this, tr("Добавление"), tr("Введите новую категорию:"), QLineEdit::Normal, "", &ok);

if (ok && !newItem.isEmpty())

{

ui->comboBox->addItem(newItem);

comboBoxLoader->saveComboBoxItems(ui->comboBox, categoryPath);

}

}

// Реализация триггера в меню для открытия диалогового окна при удалении существующей категории

void MainWindow::on\_actionDeleteCategory\_triggered()

{

int currentIndex = ui->comboBox->currentIndex();

if (currentIndex != -1)

{

QString currentItemText = ui->comboBox->itemText(currentIndex);

if (currentItemText != "Все категории")

{

ui->comboBox->removeItem(currentIndex);

comboBoxLoader->saveComboBoxItems(ui->comboBox, categoryPath);

}

else

{

QMessageBox::warning(this, tr("Предупреждение"), tr("Нельзя удалить выбранную категорию."), QMessageBox::Ok);

}

}

else

{

QMessageBox::warning(this, tr("Предупреждение"), tr("Выберите категорию для удаления."), QMessageBox::Ok);

}

}

// Реализация триггера в меню для открытия диалогового окна при добавлении новой кухни

void MainWindow::on\_actionAddKitchen\_triggered()

{

bool ok;

QString newItem = QInputDialog::getText(this, tr("Добавление"), tr("Введите новую кухню:"), QLineEdit::Normal, "", &ok);

if (ok && !newItem.isEmpty())

{

ui->comboBox\_2->addItem(newItem);

comboBoxLoader->saveComboBoxItems(ui->comboBox\_2, kitchenPath);

}

}

// Реализация триггера в меню для открытия диалогового окна при удалении существующей кухни

void MainWindow::on\_actionDeleteKitchen\_triggered()

{

int currentIndex = ui->comboBox\_2->currentIndex();

if (currentIndex != -1)

{

QString currentItemText = ui->comboBox\_2->itemText(currentIndex);

if (currentItemText != "Все кухни" || currentItemText != "Неопределенная кухня")

{

ui->comboBox\_2->removeItem(currentIndex);

comboBoxLoader->saveComboBoxItems(ui->comboBox\_2, kitchenPath);

}

else

{

QMessageBox::warning(this, tr("Предупреждение"), tr("Нельзя удалить выбранную кухню."), QMessageBox::Ok);

}

}

else

{

QMessageBox::warning(this, tr("Предупреждение"), tr("Выберите кухню для удаления."), QMessageBox::Ok);

}

}

//recipe.h

#ifndef RECIPE\_H

#define RECIPE\_H

#include <QString>

class Recipe

{

private:

int recipeId;

QString name;

QString description;

QString ingredients;

QString instruction;

int prepTime;

int servings;

QByteArray photo;

QString category;

QString kitchen;

public:

Recipe();

Recipe(int recipeId, const QString &name, const QString &description, const QString &ingredients, const QString &instruction, int prepTime, int servings, const QByteArray &photo, const QString &category, const QString &kitchen);

~Recipe();

int getRecipeId() const;

QString getName() const;

QString getDescription() const;

QString getIngredients() const;

QString getInstruction() const;

int getPrepTime() const;

int getServings() const;

QByteArray getPhoto() const;

QString getCategory() const;

QString getKitchen() const;

void setRecipeId(int recipeId);

void setName(const QString &name);

void setDescription(const QString &description);

void setIngredients(const QString &ingredients);

void setInstruction(const QString &instruction);

void setPrepTime(int prepTime);

void setServings(int servings);

void setPhoto(const QByteArray &photo);

void setCategory(const QString &category);

void setKitchen(const QString &kitchen);

};

#endif // RECIPE\_H

//recipe.cpp

#include "Headers/recipe.h"

// Конструктор для класса рецепта. Уникальный номер рецепта по-умолчанию "-1"

Recipe::Recipe() : recipeId(-1), prepTime(0), servings(0) {}

// Конструктор для класса рецепта со всеми параметрами

Recipe::Recipe(int recipeId, const QString &name, const QString &description, const QString &ingredients, const QString &instruction, int prepTime, int servings, const QByteArray &photo, const QString &category, const QString &kitchen)

: recipeId(recipeId), name(name), description(description), ingredients(ingredients),instruction(instruction), prepTime(prepTime), servings(servings), photo(photo), category(category), kitchen(kitchen) {}

// Геттер для уникального номера рецепта

int Recipe::getRecipeId() const {

return recipeId;

}

// Геттер для названия рецепта

QString Recipe::getName() const {

return name;

}

// Геттер для краткого описания рецепта

QString Recipe::getDescription() const {

return description;

}

// Геттер для ингредиентов рецепта

QString Recipe::getIngredients() const {

return ingredients;

}

// Геттер для инструкции по приготовлению

QString Recipe::getInstruction() const {

return instruction;

}

// Геттер для времени приготовления

int Recipe::getPrepTime() const {

return prepTime;

}

// Геттер для количества порций

int Recipe::getServings() const {

return servings;

}

// Геттер для фотографии блюда

QByteArray Recipe::getPhoto() const {

return photo;

}

// Геттер для категории блюда

QString Recipe::getCategory() const {

return category;

}

// Геттер для происхождения рецепта

QString Recipe::getKitchen() const {

return kitchen;

}

// Сеттер для уникального номера рецепта

void Recipe::setRecipeId(int recipeId) {

this->recipeId = recipeId;

}

// Сеттер для названия рецепта

void Recipe::setName(const QString &name) {

this->name = name;

}

// Сеттер для краткого описания рецепта

void Recipe::setDescription(const QString &description) {

this->description = description;

}

// Сеттер для ингредиентов

void Recipe::setIngredients(const QString &ingredients) {

this->ingredients = ingredients;

}

// Сеттер для инструкции по приготовлению

void Recipe::setInstruction(const QString &instruction) {

this->instruction = instruction;

}

// Сеттер для времени приготовления

void Recipe::setPrepTime(int prepTime) {

this->prepTime = prepTime;

}

// Сеттер для количества порций

void Recipe::setServings(int servings) {

this->servings = servings;

}

// Сеттер для фотографии блюда

void Recipe::setPhoto(const QByteArray &photo) {

this->photo = photo;

}

// Сеттер для категории блюда

void Recipe::setCategory(const QString &category) {

this->category = category;

}

// Сеттер для происхождения блюда

void Recipe::setKitchen(const QString &kitchen) {

this->kitchen = kitchen;

}

// Деструктор

Recipe::~Recipe() {

}

//recipedetailswindow.h

#ifndef RECIPEDETAILSWINDOW\_H

#define RECIPEDETAILSWINDOW\_H

#include <QDialog>

#include <QLabel>

#include <QVBoxLayout>

#include <QPushButton>

#include "recipe.h"

namespace Ui {

class RecipeDetailsWindow;

}

class RecipeDetailsWindow : public QDialog

{

Q\_OBJECT

public:

explicit RecipeDetailsWindow(QWidget \*parent = nullptr);

RecipeDetailsWindow(const Recipe &recipe, QWidget \*parent);

~RecipeDetailsWindow();

private:

Ui::RecipeDetailsWindow \*ui;

QLabel \*recipeNameLabel;

QLabel \*descriptionLabel;

QLabel \*ingredientsLabel;

QLabel \*instructionLabel;

QLabel \*prepTimeLabel;

QLabel \*servingsLabel;

QLabel \*photoLabel;

QLabel \*kitchenLabel;

QPushButton \*exitButton;

};

#endif // RECIPEDETAILSWINDOW\_H

//recipedetailswindow.cpp

#include "Headers/recipedetailswindow.h"

#include "ui\_recipedetailswindow.h"

// Конструктор класса без лишних параметров.

RecipeDetailsWindow::RecipeDetailsWindow(QWidget \*parent) :

QDialog(parent),

ui(new Ui::RecipeDetailsWindow)

{

ui->setupUi(this);

}

// Деструктор класса

RecipeDetailsWindow::~RecipeDetailsWindow()

{

delete ui;

}

// Конструктор класса с доп. параметром в виде рецепта.

// В конструкторе сразу находится реализация добавления элементов библиотеки QT,

// чтобы при открытии окна, все отрисовывалось. Окно открывается при вызове конструктора.

RecipeDetailsWindow::RecipeDetailsWindow(const Recipe &recipe, QWidget \*parent)

: QDialog(parent)

{

setWindowTitle(recipe.getName());

QVBoxLayout \*layout = new QVBoxLayout(this);

kitchenLabel = new QLabel(recipe.getKitchen(), this);

layout->addWidget(kitchenLabel);

recipeNameLabel = new QLabel("Название рецепта: " + recipe.getName() + " (" + recipe.getCategory() + ")", this);

layout->addWidget(recipeNameLabel);

if (!recipe.getDescription().isEmpty()) {

descriptionLabel = new QLabel("Описание: " + recipe.getDescription(), this);

layout->addWidget(descriptionLabel);

}

if (!recipe.getIngredients().isEmpty()) {

ingredientsLabel = new QLabel("Ингредиенты:\n" + recipe.getIngredients(), this);

layout->addWidget(ingredientsLabel);

}

if (!recipe.getInstruction().isEmpty()) {

instructionLabel = new QLabel("Инструкция:\n" + recipe.getInstruction(), this);

layout->addWidget(instructionLabel);

}

prepTimeLabel = new QLabel("Время приготовления: " + QString::number(recipe.getPrepTime()) + " мин", this);

layout->addWidget(prepTimeLabel);

servingsLabel = new QLabel("Порции: " + QString::number(recipe.getServings()), this);

layout->addWidget(servingsLabel);

if (!recipe.getPhoto().isEmpty()) {

QLabel \*photoLabel = new QLabel(this);

QPixmap photo;

photo.loadFromData(recipe.getPhoto());

photoLabel->setPixmap(photo);

QPixmap resizedPhoto = photo.scaled(540, 340, Qt::KeepAspectRatio);

photoLabel->setPixmap(resizedPhoto);

layout->addWidget(photoLabel);

}

exitButton = new QPushButton("Закрыть");

connect(exitButton, &QPushButton::clicked, this, &RecipeDetailsWindow::close);

layout->addWidget(exitButton);

}

//recipemanager.h

#ifndef RECIPEMANAGER\_H

#define RECIPEMANAGER\_H

#include "recipe.h"

#include <QSqlDatabase>

#include <QSqlError>

#include <QSqlQuery>

#include <QSqlTableModel>

#include <QFileDialog>

#include <QMessageBox>

class RecipeManager

{

private:

QSqlTableModel \*model;

bool updateRecipeInfo(int recipeId, const Recipe& recipe);

public:

RecipeManager();

RecipeManager(QSqlTableModel \*model);

~RecipeManager();

bool deleteRecipeById(int recipeId);

int getRecipeRowById(int recipeId);

Recipe getRecipeById(int recipeId);

int getRecipeIdFromIndex(const QModelIndex &index, QSqlQueryModel \*model);

void addRecipeToDatabase(const Recipe& recipe, const QString& photoPath);

bool updateRecipe(int recipeId, const Recipe& recipe);

};

#endif // RECIPEMANAGER\_H

//recipemanager.cpp

#include "Headers/recipemanager.h"

// Конструктор класса

RecipeManager::RecipeManager() {

}

// Деструктор

RecipeManager::RecipeManager(QSqlTableModel \*model) : model(model) {

}

// Функция для удаления рецепта из базы данных по уникальному номеру

// Здесь используется SQL-запрос "DELETE FROM" для удаления определенного

// рецепта из базы данных

bool RecipeManager::deleteRecipeById(int recipeId) {

QSqlDatabase db = QSqlDatabase::database();

QSqlQuery query(db);

db.transaction();

query.prepare("DELETE FROM RecipesInfo WHERE recipeId = :recipeId");

query.bindValue(":recipeId", recipeId);

if (!query.exec()) {

qDebug() << "Ошибка при удалении рецепта: " << query.lastError().text();

db.rollback();

return false;

}

if (model->removeRow(getRecipeRowById(recipeId))) {

if (model->submitAll()) {

db.commit();

return true;

}

}

db.rollback();

return false;

}

// Функция для получения номера строки рецепта в таблице всех рецептов

int RecipeManager::getRecipeRowById(int recipeId) {

for (int row = 0; row < model->rowCount(); ++row) {

QModelIndex index = model->index(row, 0);

if (model->data(index).toInt() == recipeId) {

return row;

}

}

return -1;

}

// Функция для получения рецепта из базы данных по уникальному номеру. Здесь

// используется SQL-запрос "SELECT \* FROM" по всем полям для выбора рецепта

// из базы данных

Recipe RecipeManager::getRecipeById(int recipeId) {

QSqlQuery query;

query.prepare("SELECT \* FROM RecipesInfo WHERE recipeId = :recipeId");

query.bindValue(":recipeId", recipeId);

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

Recipe recipe;

recipe.setRecipeId(query.value(0).toInt());

recipe.setName(query.value(1).toString());

recipe.setDescription(query.value(2).toString());

recipe.setIngredients(query.value(3).toString());

recipe.setInstruction(query.value(4).toString());

recipe.setPrepTime(query.value(5).toInt());

recipe.setServings(query.value(6).toInt());

recipe.setPhoto(query.value(7).toByteArray());

recipe.setCategory(query.value(8).toString());

recipe.setKitchen(query.value(9).toString());

return recipe;

} else {

return Recipe();

}

}

// Функция получения уникального номера рецепта по индексу в модели таблицы базы данных

int RecipeManager::getRecipeIdFromIndex(const QModelIndex &index, QSqlQueryModel \*model) {

if (index.isValid()) {

return model->data(model->index(index.row(), 0)).toInt();

} else {

return -1;

}

}

// Функция добавления рецепта в базу данных. Здесь используется SQL-запрос "INSERT INTO"

// по всем полям для занесения рецепта в базу данных

void RecipeManager::addRecipeToDatabase(const Recipe& recipe, const QString& photoPath) {

QSqlDatabase db = QSqlDatabase::database();

QSqlQuery query(db);

query.prepare("INSERT INTO RecipesInfo (name, description, ingredients, instruction, prepTime, servings, photo, category, kitchen) "

"VALUES (:name, :description, :ingredients, :instruction, :prepTime, :servings, :photo, :category, :kitchen)");

query.bindValue(":name", recipe.getName());

query.bindValue(":description", recipe.getDescription());

query.bindValue(":ingredients", recipe.getIngredients());

query.bindValue(":instruction", recipe.getInstruction());

query.bindValue(":prepTime", recipe.getPrepTime());

query.bindValue(":servings", recipe.getServings());

query.bindValue(":category", recipe.getCategory());

query.bindValue(":kitchen", recipe.getKitchen());

QFile photoFile(photoPath);

if (photoFile.open(QIODevice::ReadOnly)) {

QByteArray photoData = photoFile.readAll();

query.bindValue(":photo", photoData);

photoFile.close();

if (!query.exec()) {

qDebug() << "Ошибка запроса:" << query.lastError().text();

db.rollback();

}

db.commit();

}

// Функция для обновления данных рецепта в базе данных. Здесь используется

// SQL-запрос "UPDATE" для реализации обновления данных

bool RecipeManager::updateRecipeInfo(int recipeId, const Recipe& recipe) {

QSqlDatabase db = QSqlDatabase::database();

QSqlQuery query(db);

query.prepare("UPDATE RecipesInfo SET name = :name, description = :description, ingredients = :ingredients, "

"instruction = :instruction, prepTime = :prepTime, servings = :servings, category = :category, kitchen = :kitchen "

"WHERE recipeId = :recipeId");

query.bindValue(":name", recipe.getName());

query.bindValue(":description", recipe.getDescription());

query.bindValue(":ingredients", recipe.getIngredients());

query.bindValue(":instruction", recipe.getInstruction());

query.bindValue(":prepTime", recipe.getPrepTime());

query.bindValue(":servings", recipe.getServings());

query.bindValue(":recipeId", recipeId);

query.bindValue(":category", recipe.getCategory());

query.bindValue(":kitchen", recipe.getKitchen());

if (!query.exec()) {

qDebug() << "Ошибка запроса:" << query.lastError().text();

db.rollback();

return false;

}

return true;

}

bool RecipeManager::updateRecipe(int recipeId, const Recipe& recipe) {

if (updateRecipeInfo(recipeId, recipe)) {

return true;

}

return false;

}

RecipeManager::~RecipeManager() {

}

//recipewindowbase.h

#ifndef RECIPEWINDOWBASE\_H

#define RECIPEWINDOWBASE\_H

#include <QWidget>

#include <QVBoxLayout>

#include <QLabel>

#include <QComboBox>

#include <QLineEdit>

#include <QTextEdit>

#include <QSpinBox>

#include <QPushButton>

#include <QFileDialog>

#include "comboboxloader.h"

class RecipeWindowBase : public QDialog

{

Q\_OBJECT

public:

explicit RecipeWindowBase(QWidget \*parent = nullptr);

~RecipeWindowBase();

private:

// ДЛЯ СБОРКИ ПРОЕКТА НА ДРУГОМ УСТРОЙСТВЕ НЕОБХОДИМО ПОМЕНЯТЬ ПУТИ

const QString categoryPath = "/Users/willygodx/Qt/qt projets/3sem-CulinaryBook-C++/DataBase/comboBoxCategoryInfo.txt";

const QString kitchenPath = "/Users/willygodx/Qt/qt projets/3sem-CulinaryBook-C++/DataBase/comboBoxKitchenInfo.txt";

private slots:

void browsePhoto();

virtual void saveRecipe() = 0;

protected:

void setupUI();

void loadComboBoxItems(QComboBox \*comboBox, const QString &filePath);

QLabel \*titleLabel;

QLabel \*categoryLabel;

QComboBox \*categoryComboBox;

QLabel \*kitchenLabel;

QComboBox \*kitchenComboBox;

QLabel \*nameLabel;

QLineEdit \*nameLineEdit;

QLabel \*descriptionLabel;

QTextEdit \*descriptionTextEdit;

QLabel \*ingredientsLabel;

QTextEdit \*ingredientsTextEdit;

QLabel \*instructionLabel;

QTextEdit \*instructionTextEdit;

QLabel \*prepTimeLabel;

QSpinBox \*prepTimeSpinBox;

QLabel \*servingsLabel;

QSpinBox \*servingsSpinBox;

QLabel \*photoLabel;

QLabel \*selectedPhotoLabel;

QPushButton \*browsePhotoButton;

QPushButton \*saveButton;

QPushButton \*exitButton;

QString selectedPhotoPath;

};

#endif // RECIPEWINDOWBASE\_H

//recipewindowbase.cpp

#include "Headers/recipewindowbase.h"

// Конструктор класса

RecipeWindowBase::RecipeWindowBase(QWidget \*parent)

: QDialog(parent)

{

setupUI();

}

// Деструктор

RecipeWindowBase::~RecipeWindowBase()

{

}

// В данной функции мы выполняем установку элементов библиотеки

// QT для создания графического интерфейса окна создания/редакт

// ирования рецепта

void RecipeWindowBase::setupUI()

{

QVBoxLayout \*mainLayout = new QVBoxLayout(this);

QVBoxLayout \*formLayout = new QVBoxLayout();

titleLabel = new QLabel("Введите данные о рецепте.");

formLayout->addWidget(titleLabel);

categoryLabel = new QLabel("Категория:");

categoryComboBox = new QComboBox();

formLayout->addWidget(categoryLabel);

formLayout->addWidget(categoryComboBox);

loadComboBoxItems(categoryComboBox, categoryPath);

kitchenLabel = new QLabel("Кухня: ");

kitchenComboBox = new QComboBox();

formLayout->addWidget(kitchenLabel);

formLayout->addWidget(kitchenComboBox);

loadComboBoxItems(kitchenComboBox, kitchenPath);

nameLabel = new QLabel("Название:");

nameLineEdit = new QLineEdit();

formLayout->addWidget(nameLabel);

formLayout->addWidget(nameLineEdit);

descriptionLabel = new QLabel("Краткое описание:");

descriptionTextEdit = new QTextEdit();

formLayout->addWidget(descriptionLabel);

formLayout->addWidget(descriptionTextEdit);

ingredientsLabel = new QLabel("Ингредиенты:");

ingredientsTextEdit = new QTextEdit();

formLayout->addWidget(ingredientsLabel);

formLayout->addWidget(ingredientsTextEdit);

instructionLabel = new QLabel("Инструкция:");

instructionTextEdit = new QTextEdit();

formLayout->addWidget(instructionLabel);

formLayout->addWidget(instructionTextEdit);

prepTimeLabel = new QLabel("Время приготовления (минуты):");

prepTimeLabel->setMinimumWidth(200);

prepTimeLabel->setMaximumWidth(200);

prepTimeSpinBox = new QSpinBox();

prepTimeSpinBox->setMinimumWidth(50);

prepTimeSpinBox->setMaximumWidth(50);

prepTimeSpinBox->setMaximum(999);

formLayout->addWidget(prepTimeLabel);

formLayout->addWidget(prepTimeSpinBox);

servingsLabel = new QLabel("Количество порций:");

servingsLabel->setMinimumWidth(200);

servingsLabel->setMaximumWidth(200);

servingsSpinBox = new QSpinBox();

servingsSpinBox->setMinimumWidth(50);

servingsSpinBox->setMaximumWidth(50);

servingsSpinBox->setMaximum(999);

formLayout->addWidget(servingsLabel);

formLayout->addWidget(servingsSpinBox);

photoLabel = new QLabel("Фото:");

selectedPhotoLabel = new QLabel("Нет выбранного фото");

browsePhotoButton = new QPushButton("Выбрать фото");

connect(browsePhotoButton, &QPushButton::clicked, this, &RecipeWindowBase::browsePhoto);

QHBoxLayout \*photoLayout = new QHBoxLayout();

photoLayout->addWidget(photoLabel);

photoLayout->addWidget(selectedPhotoLabel);

photoLayout->addWidget(browsePhotoButton);

formLayout->addLayout(photoLayout);

mainLayout->addLayout(formLayout);

saveButton = new QPushButton("Сохранить");

connect(saveButton, &QPushButton::clicked, this, &RecipeWindowBase::saveRecipe);

mainLayout->addWidget(saveButton);

exitButton = new QPushButton("Закрыть");

connect(exitButton, &QPushButton::clicked, this, &RecipeWindowBase::close);

mainLayout->addWidget(exitButton);

}

// Функция для реализации слота выбора фотографии из системного каталога

void RecipeWindowBase::browsePhoto()

{

selectedPhotoPath = QFileDialog::getOpenFileName(this, "Выберите фото", "", "Images (\*.png \*.jpg)");

if (!selectedPhotoPath.isEmpty()) {

selectedPhotoLabel->setText("Фото выбрано: " + selectedPhotoPath);

}

}

// Функция для выгрузки данных из .txt файлов для виджета ComboBox

void RecipeWindowBase::loadComboBoxItems(QComboBox \*comboBox, const QString &filePath)

{

ComboBoxLoader comboBoxLoader;

comboBoxLoader.loadComboBoxItems(comboBox, filePath);

}