ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«РОССИЙСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ НЕФТИ И ГАЗА
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)
ИМЕНИ И.М. ГУБКИНА»

ФАКУЛЬТЕТ КОМПЛЕКСНОЙ БЕЗОПАСНОСТИ ТЭК
КАФЕДРА БЕЗОПАСНОСТИ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ

Лабораторная работа №3

по дисциплине «Специализированные языки и технологии программирования»

на тему «РАЗРАБОТКА ИНТЕРФЕЙСА С ИСПОЛЬЗОВАНИЕМ СТАНДАРТНЫХ ВИДЖЕТОВ»

Выполнил студент:
группы КА-22-06
Воронин Алексей Дмитриевич

Преподаватель:
Греков Владимир Сергеевич

Москва, 2025

Оглавление

Цель работы:

На основе знаний, полученных в предыдущих лабораторных работах, разработать приложение "Книжный каталог" с графическим интерфейсом пользователя, используя стандартные виджеты Qt. Приложение должно позволять добавлять, просматривать, редактировать и удалять информацию о книгах.

ЧАСТЬ 1: Разработка интерфейса приложения

Создадим новый проект "Приложение Qt Widgets", который назовем "BookCatalog".

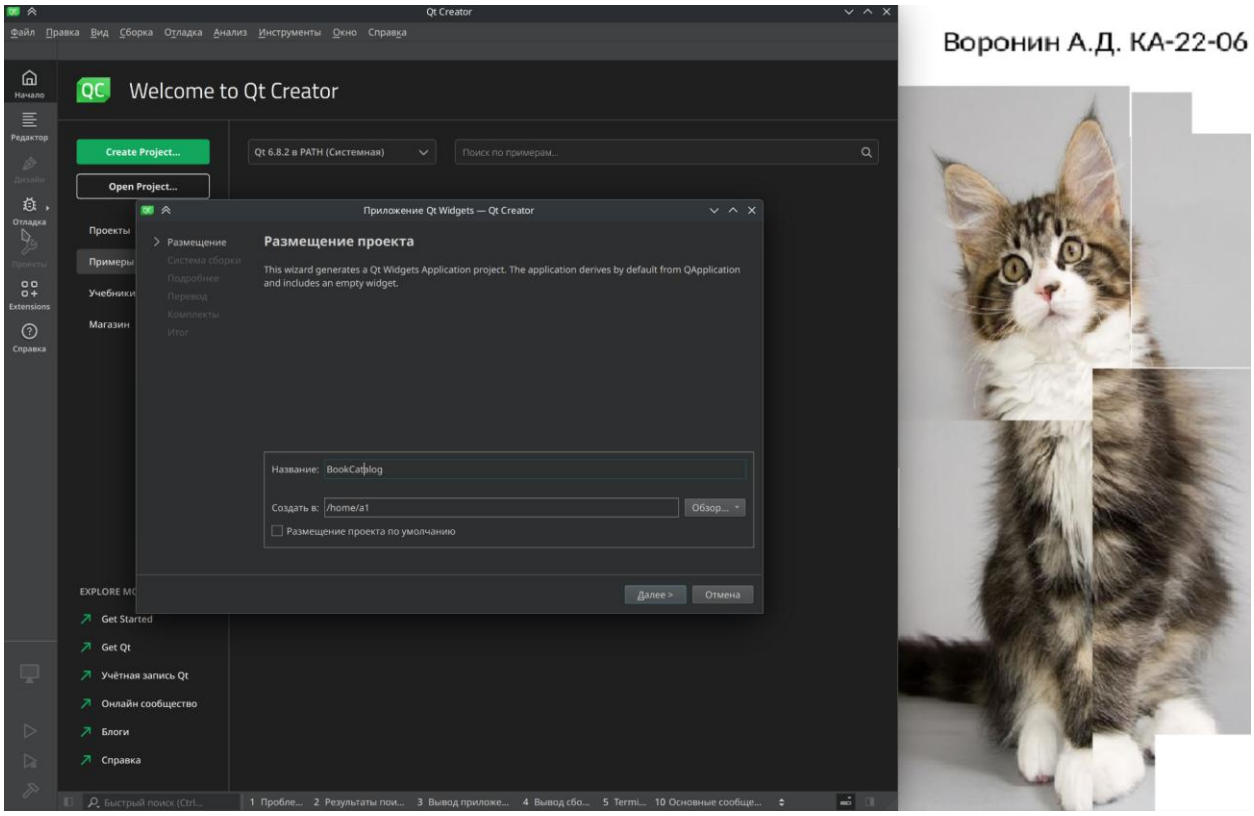

Рисунок 1 – Создание проекта

Откройте mainwindow.ui в редакторе форм Qt Creator. Добавьте виджеты:

- QTableView – для отображения каталога книг.
- QPushButton – кнопки "Добавить", "Удалить", "Редактировать".
- QMenuBar – с пунктами меню "Импорт", "Экспорт", "Выход".

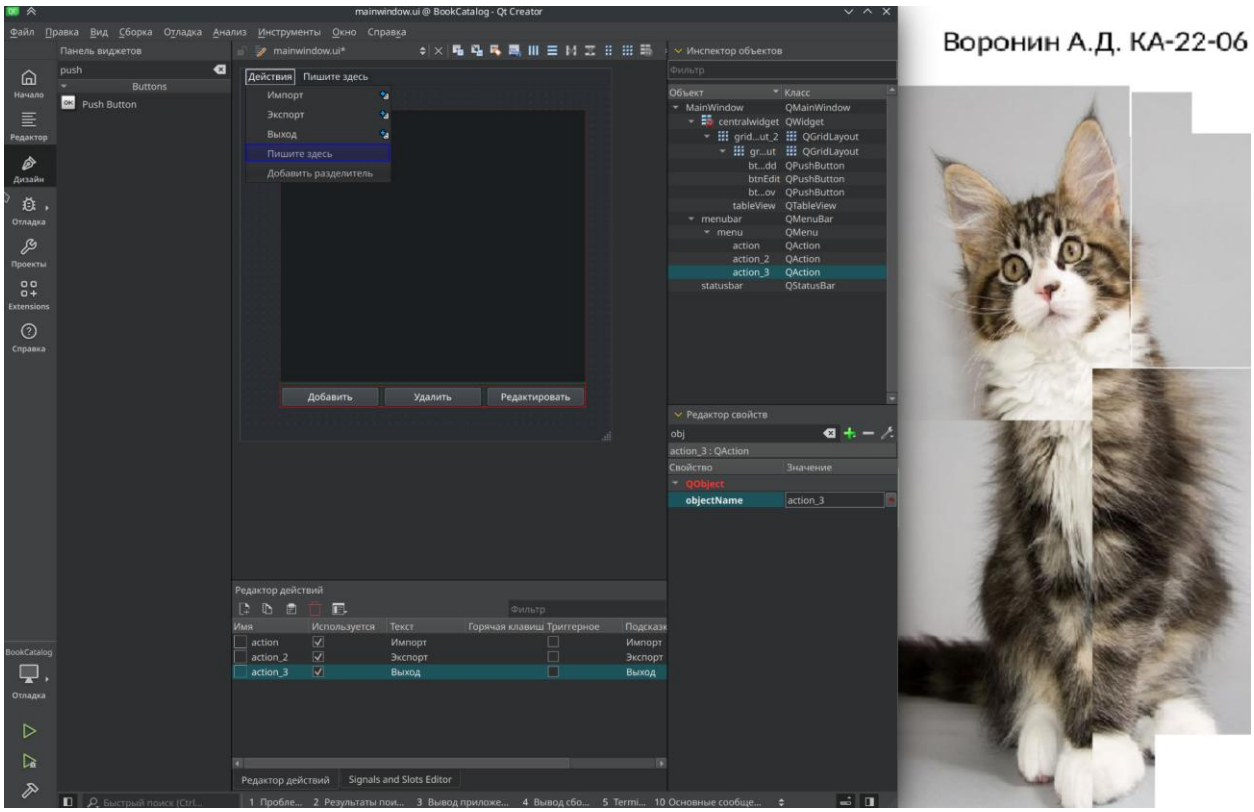

Рисунок 2 – Добавление виджетов

Создайте в папке проекта папку img, куда поместите 3 png картинки для иконок. Добавьте эти иконки в файл ресурсов Qt.

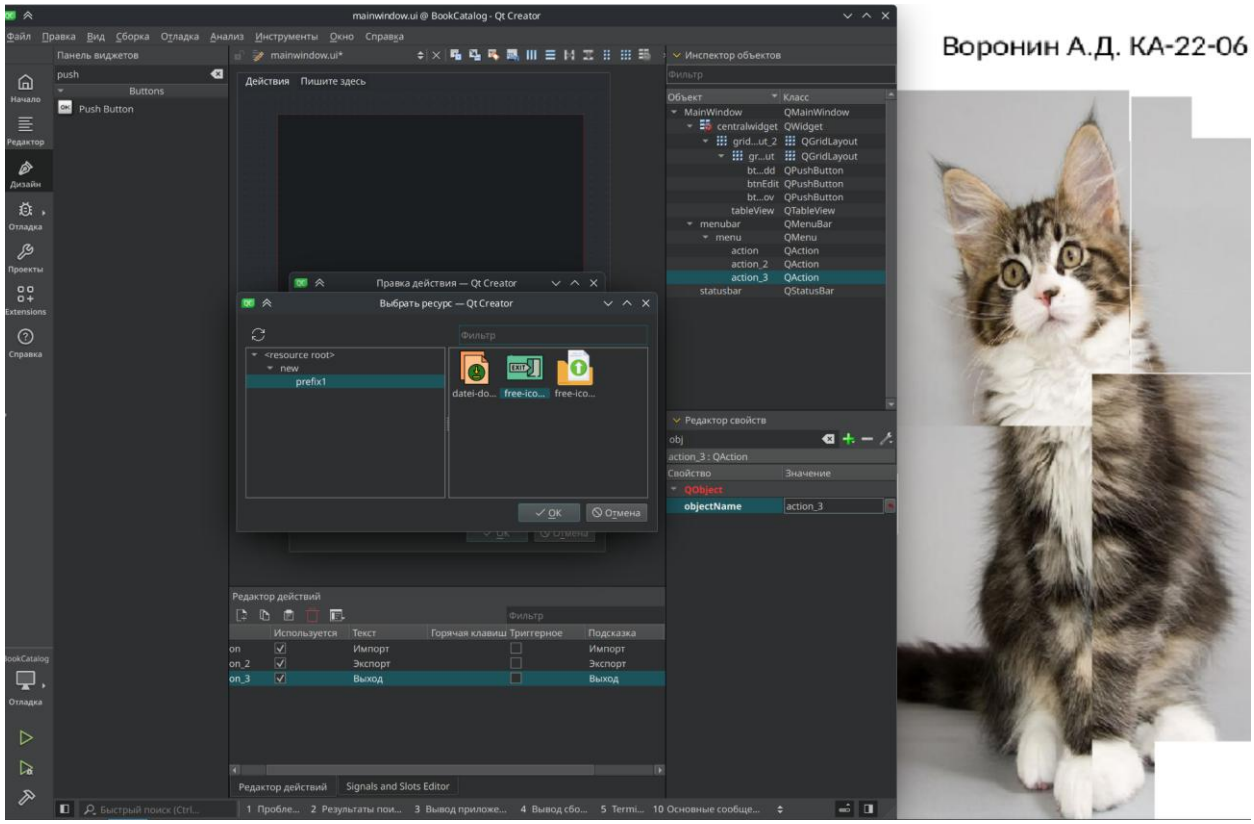

Рисунок 3 – Иконки

Перейдите в mainwindow.ui и для каждого пункта прикрепите соответствующие иконки. Для этого в редакторе действий нажмите два раза на определенный пункт и поместите в него иконку. Нажимая Choose Resource выберите файлы из папки img.

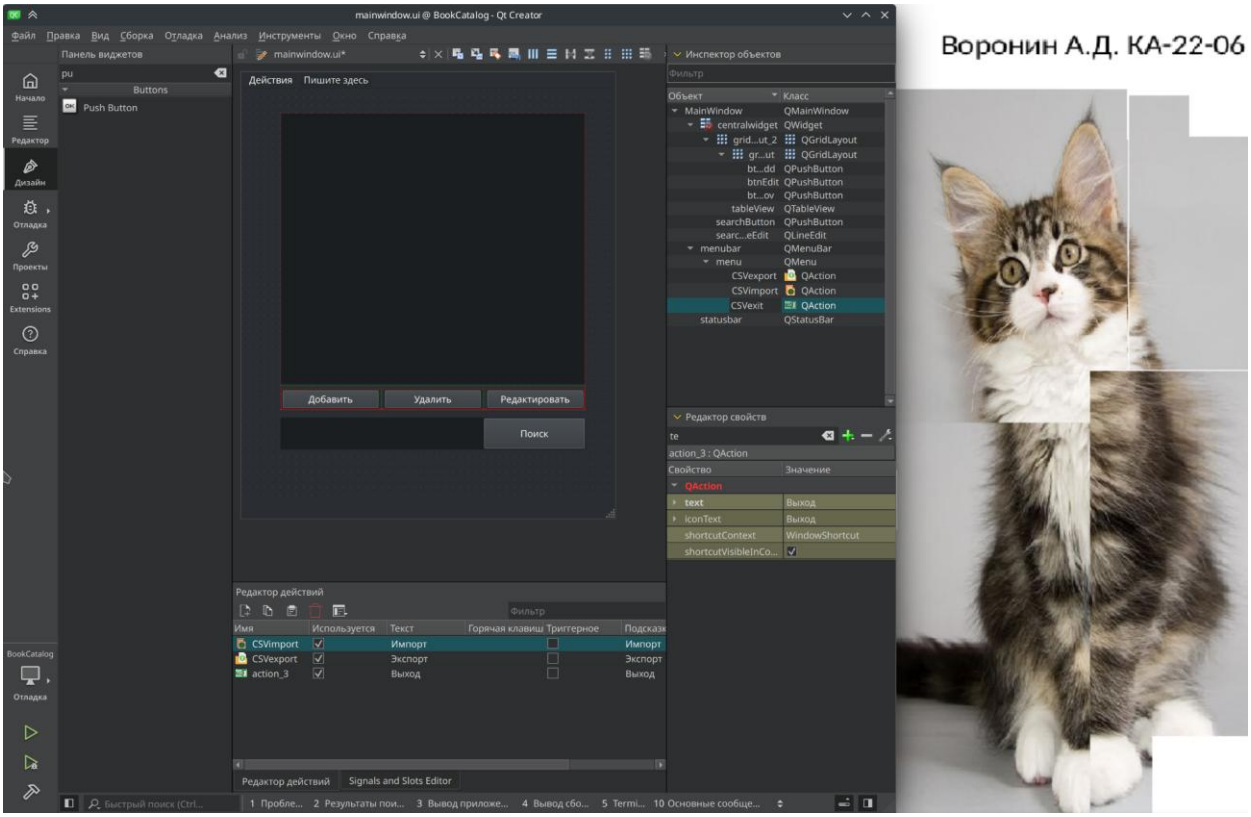

Рисунок 4 – Добавление иконок

Далее внесем изменения в код.

Код mainwindow.h

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H
#include <QMainWindow>
#include <QStandardItemModel>
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 addBook();
    void editBook();
    void removeBook();

private:
    Ui::MainWindow *ui;
    QStandardItemModel *model;
    void setupModel();
};
#endif // MAINWINDOW_H
```

Код mainwindow.cpp

```
#include "mainwindow.h"
#include "./ui_mainwindow.h"
#include <QInputDialog>
#include <QMessageBox>

MainWindow::MainWindow(QWidget *parent)
    : QMainWindow(parent)
    , ui(new Ui::MainWindow)
```

```
{
    ui->setupUi(this);
    setupModel();

        connect(ui->btnAdd, &QPushButton::clicked, this, &MainWindow::addBook);
        connect(ui->btnEdit, &QPushButton::clicked, this, &MainWindow::editBook);
        connect(ui->btnRemove, &QPushButton::clicked, this, &MainWindow::removeBook);
}

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

void MainWindow::setupModel()
{
    model = new QStandardItemModel(this);
    model->setColumnCount(4);
    model->setHeaderData(0, Qt::Horizontal, "Автор");
    model->setHeaderData(1, Qt::Horizontal, "Название");
    model->setHeaderData(2, Qt::Horizontal, "Год");
    model->setHeaderData(3, Qt::Horizontal, "Жанр");

    ui->tableView->setModel(model);
}
```

```
void MainWindow::addBook()
{
    QString author = QInputDialog::getText(this, "Добавить книгу", "Автор:");
    if (author.isEmpty()) return;

    QString title = QInputDialog::getText(this, "Добавить книгу", "Название:");
    if (title.isEmpty()) return;

    int year = QInputDialog::getInt(this, "Добавить книгу", "Год издания:", 2000, 0, 2100);

    QString genre = QInputDialog::getText(this, "Добавить книгу", "Жанр:");
    if (genre.isEmpty()) return;

    QList<QStandardItem *> rowItems;
    rowItems << new QStandardItem(author)
        << new QStandardItem(title)
        << new QStandardItem(QString::number(year))
        << new QStandardItem(genre);

    model->appendRow(rowItems);
}

void MainWindow::editBook()
```

```
{
    QModelIndex index = ui->tableView->currentIndex();
    if (!index.isValid()) {
        QMessageBox::warning(this, "Ошибка", "Выберите книгу для редактирования");
        return;
    }

    QString newText = QInputDialog::getText(this, "Редактировать книгу", "Новое значение:", QLineEdit::Normal, model->item(index.row(), index.column())->text());
    if (!newText.isEmpty()) {
        model->setItem(index.row(), index.column(), new QStandardItem(newText));
    }
}

void MainWindow::removeBook()
{
    QModelIndex index = ui->tableView->currentIndex();
    if (!index.isValid()) {
        QMessageBox::warning(this, "Ошибка", "Выберите книгу для удаления");
        return;
    }

    model->removeRow(index.row());
}
```

ЧАСТЬ 2: Реализация поиска по каталогу

Включите в интерфейс строку поиска (QLineEdit) и кнопку "Поиск".

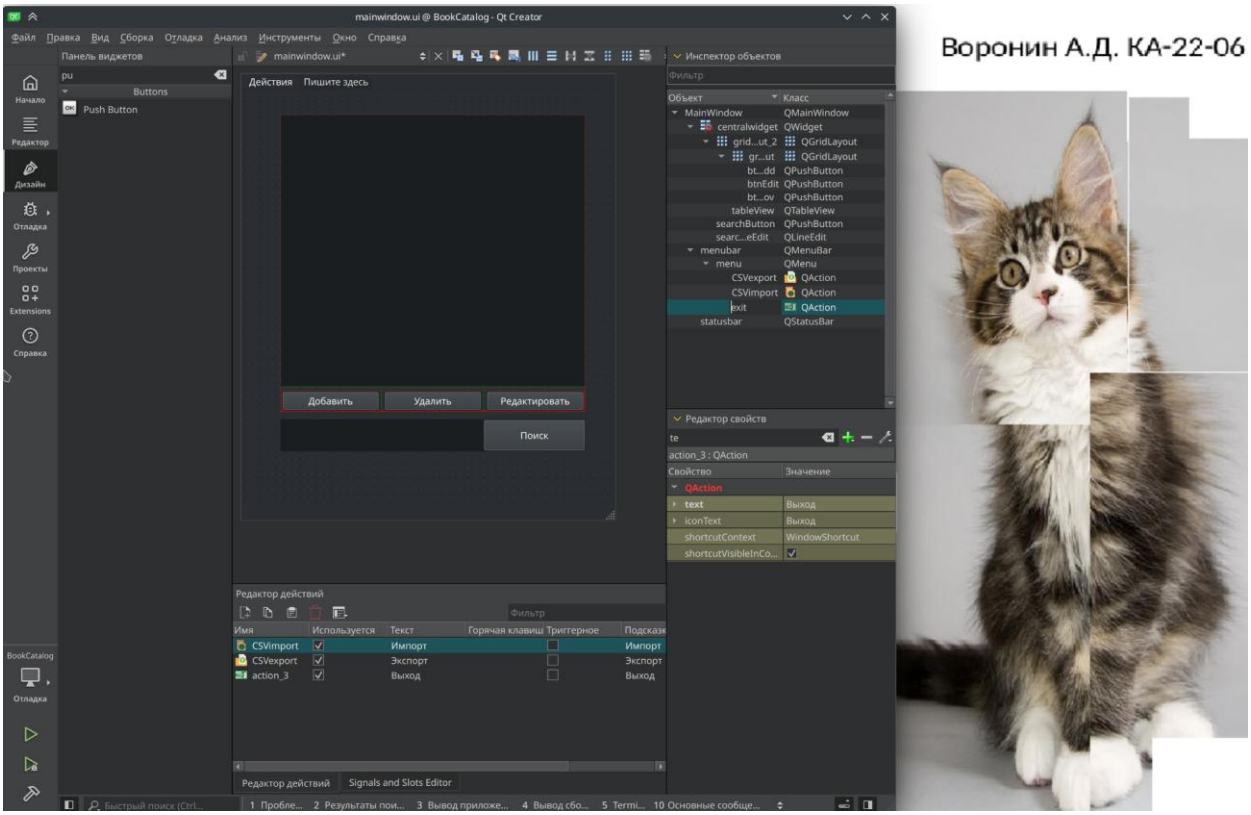

Рисунок 5 – Добавление поиска

Реализуйте фильтрацию отображаемых записей в QTableView, по ключевым словам, введенным в строку поиска.

Обновленный mainwindow.h

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H
#include <QMainWindow>
#include <QStandardItemModel>
#include <QSortFilterProxyModel>

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 addBook();
    void editBook();
    void removeBook();
    void searchBooks();

private:
    Ui::MainWindow *ui;
    QStandardItemModel *model;
    QSortFilterProxyModel *proxyModel;
    void setupModel();
    void setupSearch();
};
#endif // MAINWINDOW_H
```

Добавляем поддержку поиска в mainwindow.cpp

```
#include "mainwindow.h"
#include "./ui_mainwindow.h"
#include <QInputDialog>
#include <QMessageBox>
#include <QFileDialog>
```

```
#include <QTextStream>

MainWindow::MainWindow(QWidget *parent)
    : QMainWindow(parent)
    , ui(new Ui::MainWindow)
{
    ui->setupUi(this);
    setupModel();
    setupSearch();

    connect(ui->btnAdd, &QPushButton::clicked, this, &MainWindow::addBook);
    connect(ui->btnEdit, &QPushButton::clicked, this, &MainWindow::editBook);
    connect(ui->btnRemove, &QPushButton::clicked, this, &MainWindow::removeBook);
    connect(ui->searchButton, &QPushButton::clicked, this, &MainWindow::searchBooks);
}

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

void MainWindow::setupModel()
{
    model = new QStandardItemModel(this);
```

```
    model->setColumnCount(4);
    model->setHeaderData(0, Qt::Horizontal, "Автор");
    model->setHeaderData(1, Qt::Horizontal, "Название");
    model->setHeaderData(2, Qt::Horizontal, "Год");
    model->setHeaderData(3, Qt::Horizontal, "Жанр");

    proxyModel = new QSortFilterProxyModel(this);
    proxyModel->setSourceModel(model);
    proxyModel->setFilterCaseSensitivity(Qt::CaseInsensitive);
    proxyModel->setFilterKeyColumn(-1);

    ui->tableView->setModel(proxyModel);
}

void MainWindow::addBook()
{
    QString author = QInputDialog::getText(this, "Добавить книгу", "Автор:");
    if (author.isEmpty()) return;

    QString title = QInputDialog::getText(this, "Добавить книгу", "Название:");
    if (title.isEmpty()) return;

    int year = QInputDialog::getInt(this, "Добавить книгу", "Год издания:", 2000, 0, 2100);
```

```
        QString genre = QInputDialog::getText(this, "Добавить книгу", "Жанр:");
        if (genre.isEmpty()) return;

        QList<QStandardItem *> rowItems;
        rowItems << new QStandardItem(author)
                 << new QStandardItem(title)
                 << new QStandardItem(QString::number(year))
                 << new QStandardItem(genre);

        model->appendRow(rowItems);
    }

    void MainWindow::editBook()
    {
        QModelIndex index = ui->tableView->currentIndex();
        if (!index.isValid()) {
            QMessageBox::warning(this, "Ошибка", "Выберите книгу для редактирования");
            return;
        }

        QString newText = QInputDialog::getText(this, "Редактировать книгу", "Новое значение:", QLineEdit::Normal, model->item(index.row(), index.column())->text());
        if (!newText.isEmpty()) {
            model->setItem(index.row(), index.column(), new QStandardItem(newText));
```

```
    }
}

void MainWindow::removeBook()
{
    QModelIndex index = ui->tableView->currentIndex();
    if (!index.isValid()) {
        QMessageBox::warning(this, "Ошибка", "Выберите книгу для удаления");
        return;
    }

    model->removeRow(index.row());
}

void MainWindow::setupSearch()
{
}

void MainWindow::searchBooks()
{
    QString searchText = ui->searchLineEdit->text();
    proxyModel->setFilterFixedString(searchText);
}
```

ЧАСТЬ 3: Импорт и экспорт каталога

Реализуйте возможность сохранения каталога книг в файл и загрузки из файла через пункты меню в QMenuBar.

Объявите слот для экспорта и импорта CSV в mainwindow.h

```
private slots:
    void importCSV();
    void exportCSV();
```

Добавьте код для обработки пунктов экспорта, импорта и выхода в mainwindow.cpp

Для экспорта.

```
#include <QFileDialog>
#include <QTextStream>
void MainWindow::exportCSV()
{
    QString fileName = QFileDialog::getSaveFileName(this, "Экспорт в CSV", "", "CSV Files (*.csv)");
    if (fileName.isEmpty()) {

        return;
    }

    if (!fileName.endsWith(".csv", Qt::CaseInsensitive)) {
        fileName += ".csv";
    }

    QFile file(fileName);
    if (!file.open(QIODevice::WriteOnly | QIODevice::Text)) {
```

```
        QMessageBox::warning(this, "Ошибка", "Не удалось сохранить файл");
        return;
    }

    QTextStream out(&file);

    for (int row = 0; row < model->rowCount(); ++row) {
        QStringList fields;
        for (int col = 0; col < model->columnCount(); ++col) {
            QString text = model->item(row, col)->text();
            text.replace("\"", "\"\"");
            fields.append("" + text + "");
        }
        out << fields.join(",") << "\n";
    }

    file.close();
    QMessageBox::information(this, "Экспорт", "Файл сохранён успешно!");
}
```

Для импорта

```
void MainWindow::importCSV()
{
    QString fileName = QFileDialog::getOpenFileName(this, "Импорт CSV", "", "CSV Files (*.csv)");
    if (fileName.isEmpty()) {
```

```
        return;
    }

    QFile file(fileName);
    if (!file.open(QIODevice::ReadOnly | QIODevice::Text)) {
        QMessageBox::warning(this, "Ошибка", "Не удалось открыть файл");
        return;
    }

    QTextStream in(&file);
    model->clear();
    setupModel();

    int rowCount = 0;
    while (!in.atEnd()) {
        QString line = in.readLine();
        QStringList fields = line.split(",");

        if (fields.size() != 4) {
            continue;
        }

        QList<QStandardItem *> rowItems;
        for (const QString &field : fields)
            rowItems.append(new QStandardItem(field.trimmed()));
```

```
            model->appendRow(rowItems);
            rowCount++;
        }

        file.close();

        if (rowCount == 0) {
            QMessageBox::information(this, "Импорт CSV", "Файл загружен, но он пуст.");
        } else {
            QMessageBox::information(this, "Импорт CSV", "Файл успешно загружен: " + fileName);
        }
    }
```

Соедините сигналы наших пунктов

```
    connect(ui->actionImportCSV, &QAction::triggered, this, &MainWindow::importCSV);
    connect(ui->actionExportCSV, &QAction::triggered, this, &MainWindow::exportCSV);
    connect(ui->actionExit, &QAction::triggered, this, &QApplication::quit);
```

Протестируем приложение.

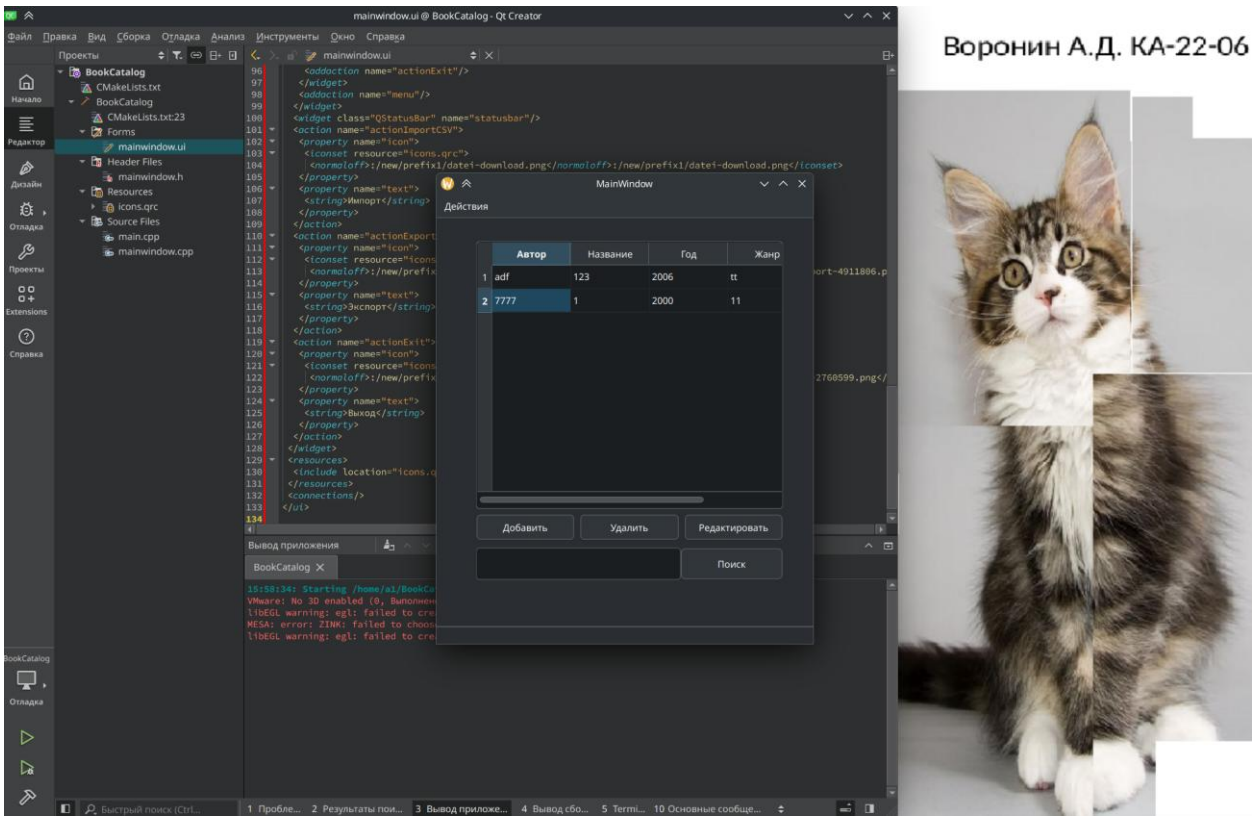

Рисунок 6 – Тестирование

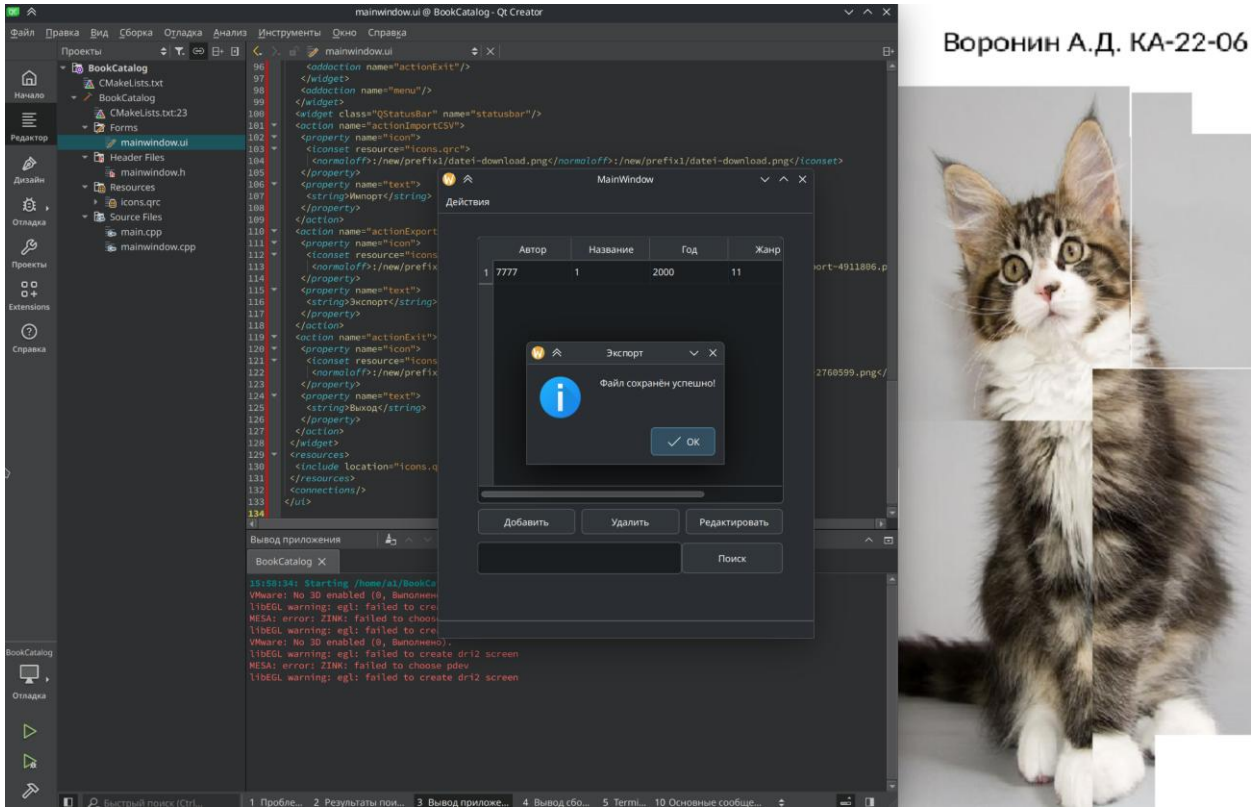

Рисунок 7 – Тестирование

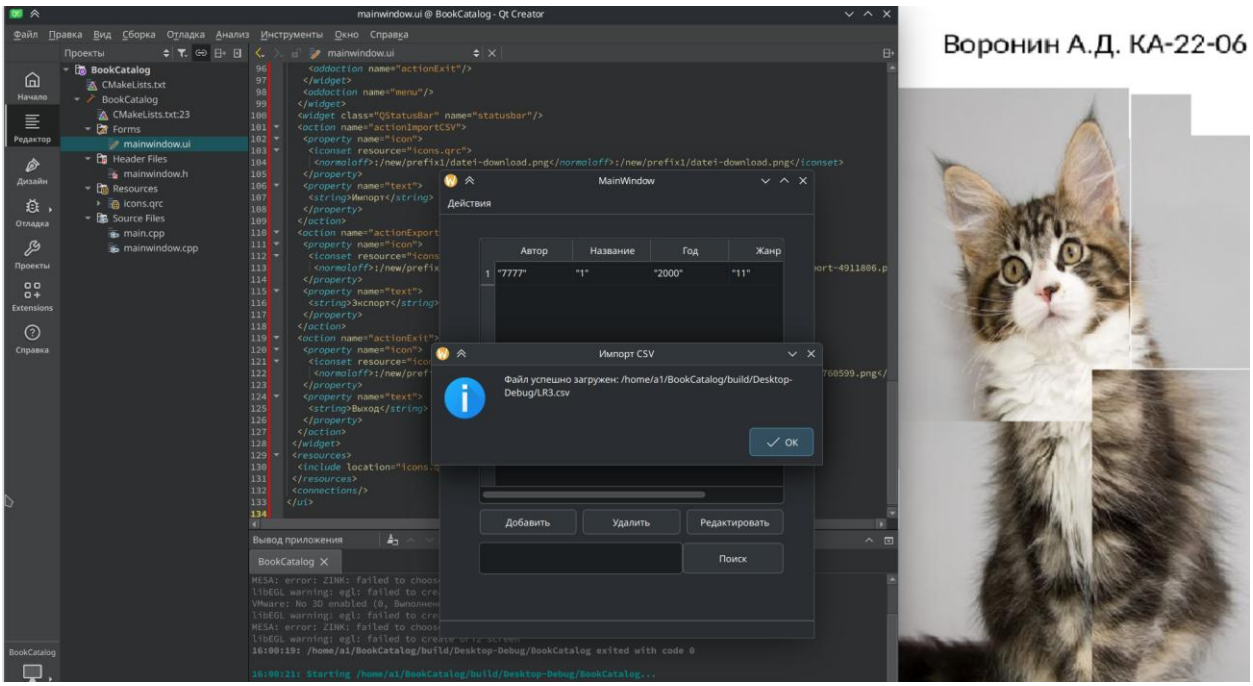

Рисунок 8 – Тестирование

Самостоятельная работа

Расширение функциональности поиска и сортировки:

- Реализуйте дополнительную фильтрацию, позволяющую искать книги не только по общему ключевому слову, но и по отдельным полям (например, автор или жанр).
- Добавьте возможность сортировки данных по столбцам (например, по году издания или по алфавиту) с использованием виджетов, таких как QHeaderView.

Редактирование через диалоговое окно:

- Создайте отдельное диалоговое окно для редактирования данных книги, в котором пользователь сможет изменять все поля записи.
- Обеспечьте валидацию вводимых данных (например, проверку корректности года издания или обязательность заполнения полей).

Работа с альтернативными форматами данных:

- Расширьте функционал импорта/экспорта, добавив поддержку формата JSON для сохранения и загрузки каталога.
- Сравните особенности работы с CSV и JSON, продемонстрируйте преимущества каждого подхода в зависимости от задачи.

Добавление новых параметров для книги:

- Модифицируйте модель данных, добавив дополнительные поля, например, «Издательство», «ISBN» или «Количество страниц».
- Обновите интерфейс, обеспечив корректное отображение и редактирование новых данных, а также добавьте функционал поиска по этим параметрам.

1) Внесем изменения в mainwindow.h

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H
#include <QMainWindow>
#include <QStandardItemModel>
```

```
#include <QSortFilterProxyModel>
#include <QComboBox>
#include "edit_book_dialog.h"

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 addBook();
    void editBook();
    void removeBook();
    void searchBooks();
    void importCSV();
    void exportCSV();
    void importJSON();
    void exportJSON();

private:
    Ui::MainWindow *ui;
```

```
    QStandardItemModel *model;
    QSortFilterProxyModel *proxyModel;
    QComboBox *filterComboBox;
    void setupModel();
    void setupSearch();
};
#endif // MAINWINDOW_H
```

2. Внесем изменения в mainwindow.cpp

```
#include "mainwindow.h"
#include "ui_mainwindow.h"
#include <QInputDialog>
#include <QMessageBox>
#include <QFileDialog>
#include <QTextStream>
#include <QFileDialog>
#include <QTextStream>
#include <QDialog>
#include <QJsonDocument>
#include <QJsonObject>
#include <QFile>
#include <QJsonArray>

MainWindow::MainWindow(QWidget *parent)
    : QMainWindow(parent)
    , ui(new Ui::MainWindow)
{
    ui->setupUi(this);
    setupModel();
```

```
    setupSearch();

    connect(ui->btnAdd, &QPushButton::clicked, this, &MainWindow::addBook);
    connect(ui->btnEdit, &QPushButton::clicked, this, &MainWindow::editBook);
    connect(ui->btnRemove, &QPushButton::clicked, this, &MainWindow::removeBook);
    connect(ui->searchButton, &QPushButton::clicked, this, &MainWindow::searchBooks);
    connect(ui->actionImportCSV, &QAction::triggered, this, &MainWindow::importCSV);
    connect(ui->actionExportCSV, &QAction::triggered, this, &MainWindow::exportCSV);
    connect(ui->actionExit, &QAction::triggered, this, &QApplication::quit);
    connect(ui->actionImportJSON, &QAction::triggered, this, &MainWindow::importJSON);
    connect(ui->actionExportJSON, &QAction::triggered, this, &MainWindow::exportJSON);
}

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

void MainWindow::exportCSV()
{
```

```
        QString fileName = QFileDialog::getSaveFileName(this, "Экспорт в CSV", "", "CSV Files (*.csv)");
        if (fileName.isEmpty()) {
            return;
        }

        if (!fileName.endsWith(".csv", Qt::CaseInsensitive)) {
            fileName += ".csv";
        }

        QFile file(fileName);
        if (!file.open(QIODevice::WriteOnly | QIODevice::Text)) {
            QMessageBox::warning(this, "Ошибка", "Не удалось сохранить файл");
            return;
        }

        QTextStream out(&file);

        for (int row = 0; row < model->rowCount(); ++row) {
            QStringList fields;
            for (int col = 0; col < model->columnCount(); ++col) {
                QString text = model->item(row, col)->text();

                // text.replace("\"", "\"\"");
                text.replace("\"", "");
                text.replace(";", ",");
                // fields.append("" + text + "");
```

```
            fields.append(text);
        }
        out << fields.join(";") << "\n";
    }

    file.close();
    QMessageBox::information(this, "Экспорт", "Файл сохранён успешно!");
}

void MainWindow::importCSV()
{
    QString fileName = QFileDialog::getOpenFileName(this, "Импорт CSV", "", "CSV Files (*.csv)");
    if (fileName.isEmpty()) {
        return;
    }

    QFile file(fileName);
    if (!file.open(QIODevice::ReadOnly | QIODevice::Text)) {
        QMessageBox::warning(this, "Ошибка", "Не удалось открыть файл");
        return;
    }

    QTextStream in(&file);
    model->clear();
    setupModel();
```

```
        int rowCount = 0;
        while (!in.atEnd()) {
            QString line = in.readLine();
            QStringList fields = line.split(";");

            if (fields.size() != 7) {
                continue;
            }

            QList<QStandardItem *> rowItems;
            for (const QString &field : fields) {
                rowItems.append(new QStandardItem(field.trimmed()));
            }

            model->appendRow(rowItems);
            rowCount++;
        }

        file.close();

        if (rowCount == 0) {
            QMessageBox::information(this, "Импорт CSV", "Файл загружен, но он пуст.");
        } else {
            QMessageBox::information(this, "Импорт CSV", "Файл успешно загружен: " + fileName);
        }
    }
```

```
void MainWindow::exportJSON()
{
    QString fileName = QFileDialog::getSaveFileName(this, "Экспорт в JSON", "", "JSON Files (*.json)");
    if (fileName.isEmpty()) {
        return;
    }

    if (!fileName.endsWith(".json", Qt::CaseInsensitive)) {
        fileName += ".json";
    }

    QFile file(fileName);
    if (!file.open(QIODevice::WriteOnly | QIODevice::Text)) {
        QMessageBox::warning(this, "Ошибка", "Не удалось сохранить файл");
        return;
    }

    QJsonArray jsonArray;
    for (int row = 0; row < model->rowCount(); ++row) {
        QJsonObject jsonObject;
        jsonObject["author"] = model->item(row, 0)->text();
        jsonObject["title"] = model->item(row, 1)->text();
        jsonObject["year"] = model->item(row, 2)->text();
        jsonObject["genre"] = model->item(row, 3)->text();
        jsonObject["publisher"] = model->item(row, 4)->text();
```

```
            jsonObject["isbn"] = model->item(row, 5)->text();
            jsonObject["pageCount"] = model->item(row, 6)->text();
            jsonArray.append(jsonObject);
        }

        QJsonDocument jsonDoc(jsonArray);
        file.write(jsonDoc.toJson());
        file.close();

        QMessageBox::information(this, "Экспорт", "Файл сохранён успешно!");
    }

    void MainWindow::importJSON()
    {
        QString fileName = QFileDialog::getOpenFileName(this, "Импорт JSON", "", "JSON Files (*.json)");
        if (fileName.isEmpty()) {
            return;
        }

        QFile file(fileName);
        if (!file.open(QIODevice::ReadOnly | QIODevice::Text)) {
            QMessageBox::warning(this, "Ошибка", "Не удалось открыть файл");
            return;
        }

        QByteArray jsonData = file.readAll();
```

```
        QJsonDocument jsonDoc = QJsonDocument::fromJson(jsonData);
        if (!jsonDoc.isArray()) {
            QMessageBox::warning(this, "Ошибка", "Неверный формат JSON");
            return;
        }

        model->clear();
        setupModel();

        QJsonArray jsonArray = jsonDoc.array();
        for (const QJsonValue &value : jsonArray) {
            if (value.isObject()) {
                QJsonObject jsonObject = value.toObject();
                QList<QStandardItem *> rowItems;
                rowItems.append(new
QStandardItem(jsonObject["author"].toString()));
                rowItems.append(new
QStandardItem(jsonObject["title"].toString()));
                rowItems.append(new
QStandardItem(jsonObject["year"].toString()));
                rowItems.append(new
QStandardItem(jsonObject["genre"].toString()));
                rowItems.append(new
QStandardItem(jsonObject["publisher"].toString()));
                rowItems.append(new
QStandardItem(jsonObject["isbn"].toString()));
                rowItems.append(new
QStandardItem(jsonObject["pageCount"].toString()));
```

```
            model->appendRow(rowItems);
        }
    }

    file.close();
    QMessageBox::information(this, "Импорт JSON", "Файл успешно загружен: " + fileName);
}

void MainWindow::setupModel()
{
    model = new QStandardItemModel(this);
    model->setColumnCount(7);
    model->setHeaderData(0, Qt::Horizontal, "Автор");
    model->setHeaderData(1, Qt::Horizontal, "Название");
    model->setHeaderData(2, Qt::Horizontal, "Год");
    model->setHeaderData(3, Qt::Horizontal, "Жанр");
    model->setHeaderData(4, Qt::Horizontal, "Издательство");
    model->setHeaderData(5, Qt::Horizontal, "ISBN");
    model->setHeaderData(6, Qt::Horizontal, "Количество страниц");

    proxyModel = new QSortFilterProxyModel(this);
    proxyModel->setSourceModel(model);
    proxyModel->setFilterCaseSensitivity(Qt::CaseInsensitive);
    proxyModel->setFilterKeyColumn(-1);

    ui->tableView->setModel(proxyModel);
    ui->tableView->setSortingEnabled(true);
```

```
    ui->tableView->horizontalHeader()->setSortIndicatorShown(true);
}

void MainWindow::addBook()
{
    QString author = QInputDialog::getText(this, "Добавить книгу", "Автор:");
    if (author.isEmpty()) return;

    QString title = QInputDialog::getText(this, "Добавить книгу", "Название:");
    if (title.isEmpty()) return;

    int year = QInputDialog::getInt(this, "Добавить книгу", "Год издания:", 1000, 0, 2100);

    QString genre = QInputDialog::getText(this, "Добавить книгу", "Жанр:");
    if (genre.isEmpty()) return;

    QString publisher = QInputDialog::getText(this, "Добавить книгу", "Издательство:");
    if (publisher.isEmpty()) return;

    QString isbn = QInputDialog::getText(this, "Добавить книгу", "ISBN:");
    if (isbn.isEmpty()) return;
```

```
        int pageCount = QInputDialog::getInt(this, "Добавить книгу", "Количество страниц:", 100, 1);

        QList<QStandardItem *> rowItems;
        rowItems << new QStandardItem(author)
                 << new QStandardItem(title)
                 << new QStandardItem(QString::number(year))
                 << new QStandardItem(genre)
                 << new QStandardItem(publisher)
                 << new QStandardItem(isbn)
                 << new QStandardItem(QString::number(pageCount));

        model->appendRow(rowItems);
    }

    void MainWindow::editBook()
    {
        QModelIndex index = ui->tableView->currentIndex();
        if (!index.isValid()) {
            QMessageBox::warning(this, "Ошибка", "Выберите книгу для редактирования");
            return;
        }

        QString author = model->item(index.row(), 0)->text();
        QString title = model->item(index.row(), 1)->text();
        int year = model->item(index.row(), 2)->text().toInt();
        QString genre = model->item(index.row(), 3)->text();
```

```
        QString publisher = model->item(index.row(), 4)->text();
        QString isbn = model->item(index.row(), 5)->text();
        int pageCount = model->item(index.row(), 6)->text().toInt();

        EditBookDialog dialog(this);
        dialog.setBookData(author, title, year, genre, publisher, isbn, pageCount);

        if (dialog.exec() == QDialog::Accepted) {
            model->setItem(index.row(), 0, new QStandardItem(dialog.getAuthor()));
            model->setItem(index.row(), 1, new QStandardItem(dialog.getTitle()));
            model->setItem(index.row(), 2, new QStandardItem(QString::number(dialog.getYear())));
            model->setItem(index.row(), 3, new QStandardItem(dialog.getGenre()));
            model->setItem(index.row(), 4, new QStandardItem(dialog.getPublisher()));
            model->setItem(index.row(), 5, new QStandardItem(dialog.getISBN()));
            model->setItem(index.row(), 6, new QStandardItem(QString::number(dialog.getPageCount())));
        }
    }

    void MainWindow::removeBook()
    {
        QModelIndex index = ui->tableView->currentIndex();
        if (!index.isValid()) {
```

```
            QMessageBox::warning(this, "Ошибка", "Выберите книгу для удаления");
            return;
        }

        model->removeRow(index.row());
    }

    void MainWindow::setupSearch()
    {
        filterComboBox = new QComboBox(this);
        filterComboBox->addItem("Все поля");
        filterComboBox->addItem("Автор");
        filterComboBox->addItem("Название");
        filterComboBox->addItem("Год");
        filterComboBox->addItem("Жанр");
        filterComboBox->addItem("Издательство");
        filterComboBox->addItem("ISBN");
        filterComboBox->addItem("Количество страниц");

        ui->searchBoxLayout->addWidget(filterComboBox);

        connect(ui->searchLineEdit, &QLineEdit::textChanged, this, &MainWindow::searchBooks);
        connect(filterComboBox, &QComboBox::currentIndexChanged, this, &MainWindow::searchBooks);
    }
```

```
void MainWindow::searchBooks()
{
    QString searchText = ui->searchLineEdit->text();
    int filterColumn = -1;

    switch (filterComboBox->currentIndex()) {
        case 1: // Автор
            filterColumn = 0;
            break;
        case 2: // Название
            filterColumn = 1;
            break;
        case 3: // Год
            filterColumn = 2;
            break;
        case 4: // Жанр
            filterColumn = 3;
            break;
        case 5: // Издательство
            filterColumn = 4;
            break;
        case 6: // ISBN
            filterColumn = 5;
            break;
        case 7: // Количество страниц
            filterColumn = 6;
            break;
        default: // Все поля
```

```
            filterColumn = -1;
            break;
        }

        proxyModel->setFilterKeyColumn(filterColumn);
        proxyModel->setFilterFixedString(searchText);
    }
```

3) Создадим новый класс edit_book_dialog и внесем в edit_book_dialog.h

```
#ifndef EDITBOOKDIALOG_H
#define EDITBOOKDIALOG_H

#include <QDialog>

namespace Ui {
class EditBookDialog;
}

class EditBookDialog : public QDialog
{
    Q_OBJECT

public:
    explicit EditBookDialog(QWidget *parent = nullptr);
    ~EditBookDialog();

    void setBookData(const QString &author, const QString &title, int year, const QString &genre, const QString &publisher, const QString &isbn, int pageCount);
```

```
    QString getAuthor() const;
    QString getTitle() const;
    int getYear() const;
    QString getGenre() const;
    QString getPublisher() const;
    QString getISBN() const;
    int getPageCount() const;

private slots:
    void on_buttonBox_accepted();

private:
    Ui::EditBookDialog *ui;
};

#endif // EDITBOOKDIALOG_H
```

4) внесем изменения в edit_book_dialog.cpp

```
#include "edit_book_dialog.h"
#include "ui_editbookdialog.h"
#include <QMessageBox>

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

```
EditBookDialog::~EditBookDialog()
{
    delete ui;
}

void EditBookDialog::setBookData(const QString &author, const QString &title, int year, const QString &genre, const QString &publisher, const QString &isbn, int pageCount)
{
    ui->authorLineEdit->setText(author);
    ui->titleLineEdit->setText(title);
    ui->yearSpinBox->setValue(year);
    ui->genreLineEdit->setText(genre);
    ui->publisherLineEdit->setText(publisher);
    ui->isbnLineEdit->setText(isbn);
    ui->pageCountSpinBox->setValue(pageCount);
}

QString EditBookDialog::getPublisher() const
{
    return ui->publisherLineEdit->text();
}

QString EditBookDialog::getISBN() const
{
    return ui->isbnLineEdit->text();
}
```

```
int EditBookDialog::getPageCount() const
{
    return ui->pageCountSpinBox->value();
}

QString EditBookDialog::getAuthor() const
{
    return ui->authorLineEdit->text();
}

QString EditBookDialog::getTitle() const
{
    return ui->titleLineEdit->text();
}

int EditBookDialog::getYear() const
{
    return ui->yearSpinBox->value();
}

QString EditBookDialog::getGenre() const
{
    return ui->genreLineEdit->text();
}

void EditBookDialog::on_buttonBox_accepted()
{
```

```
        if (getAuthor().isEmpty() || getTitle().isEmpty() || getGenre().isEmpty() || getPublisher().isEmpty() || getISBN().isEmpty()) {
            QMessageBox::warning(this, "Ошибка", "Все поля должны быть заполнены.");
            return;
        }

        int year = getYear();
        if ((year < 0) || (year > 2100)) {
            QMessageBox::warning(this, "Ошибка", "Введите корректный год издания.");
            return;
        }

        int pageCount = getPageCount();
        if (pageCount <= 0) {
            QMessageBox::warning(this, "Ошибка", "Количество страниц должно быть положительным.");
            return;
        }

        accept();
    }
```

5) Добавим интерфейс в editbookdialog.ui

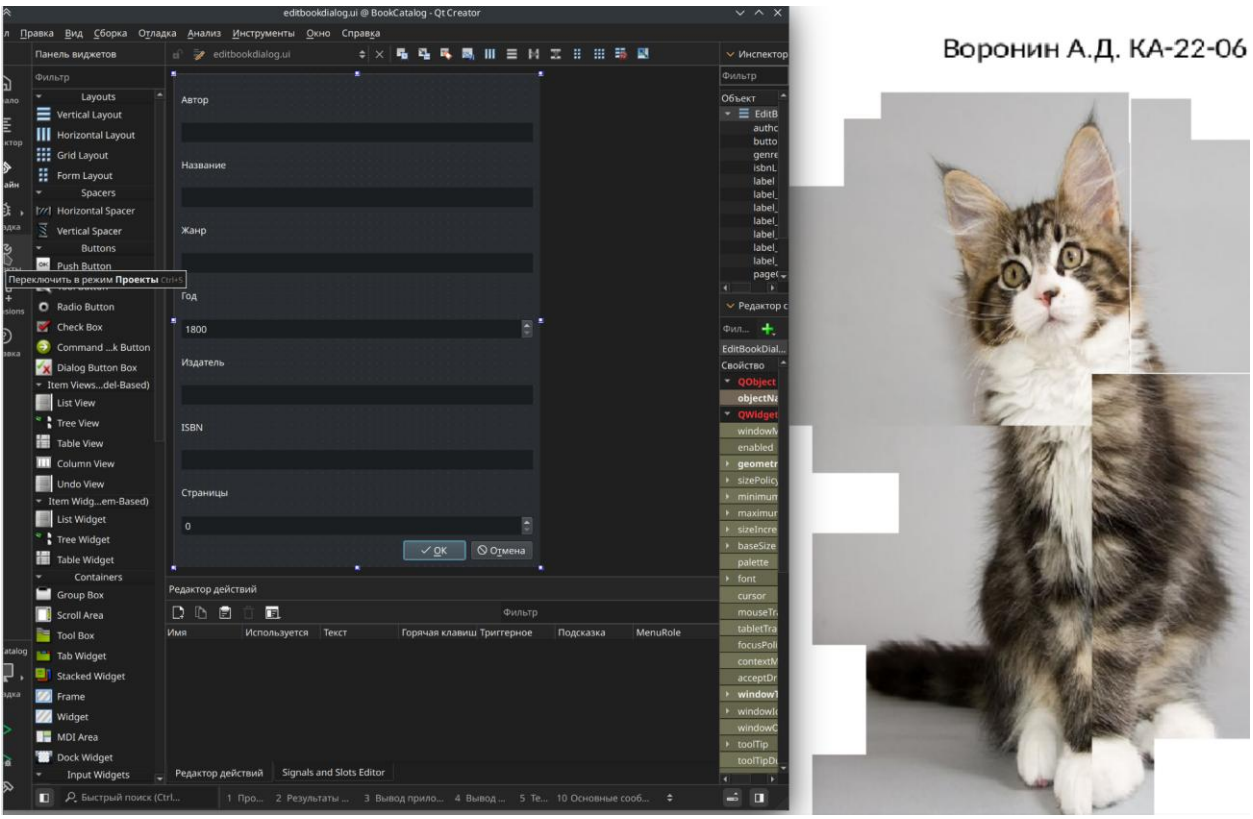

Рисунок 9 – Окно редактирования

Протестируем приложение.

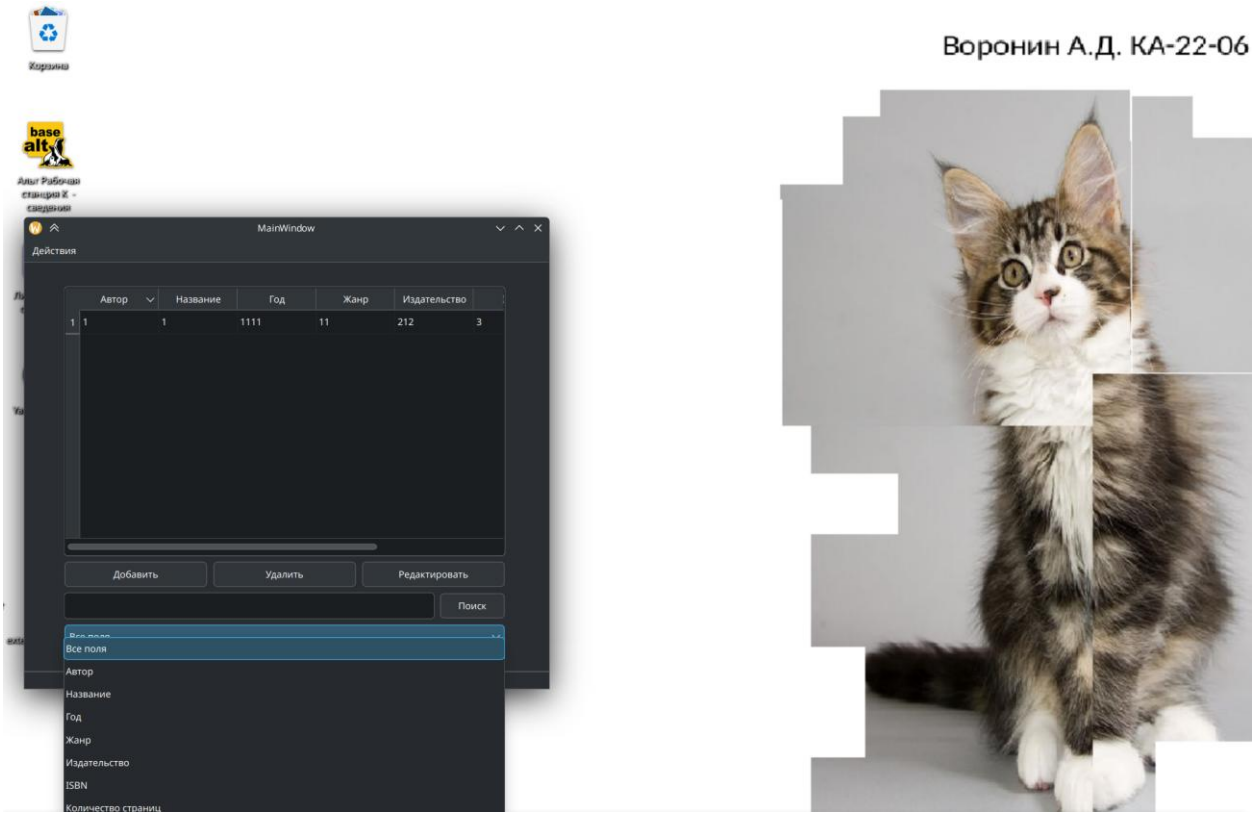

Рисунок 10 – Тестирование

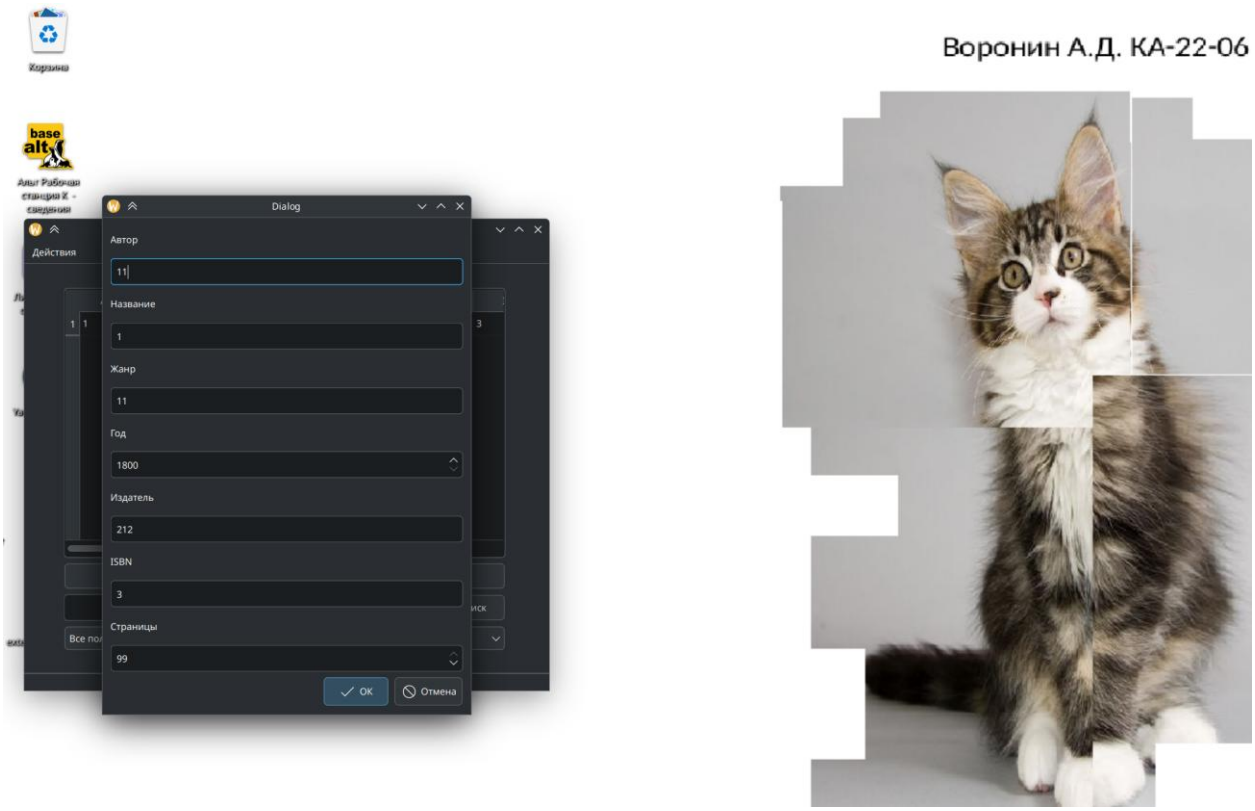

Рисунок 11 – Тестирование

ЗАКЛЮЧЕНИЕ

В заключение, подводя итог проделанной работе, все поставленные в начале цели и задания данной лабораторной работы были достигнуты в полном объеме.

Контрольные вопросы

1. Какова роль модели QStandardItemModel в реализации книжного каталога и чем она отличается от других моделей MVC в Qt?

QStandardItemModel в Qt используется как универсальная модель для представления табличных или иерархических данных и хорошо подходит для реализации книжного каталога.

В такой модели каждая строка может представлять книгу, а столбцы — её характеристики: название, автор, год, жанр, ISBN. Данные добавляются с помощью объектов QStandardItem, что упрощает работу с моделью без необходимости реализовывать собственные классы.

По сравнению с другими моделями Qt:

QStringListModel — самая простая модель, подходит только для списка строк (например, названий книг). QAbstractTableModel и QAbstractListModel требуют ручной реализации, но дают больше гибкости и лучше подходят для интеграции с собственными структурами данных и для работы с большими объёмами информации.

QStandardItemModel — оптимальный выбор, когда нужна простота, наглядность и быстрое построение интерфейса.

Она особенно удобна при работе с небольшими и средними объёмами данных, где не требуется высокая производительность или сложная логика.

2. Что представляет собой QSortFilterProxyModel и как он используется для реализации поиска и сортировки данных?

QSortFilterProxyModel — это вспомогательная модель в Qt, которая используется для сортировки и фильтрации данных, не изменяя исходную модель. Она работает как "прослойка" между моделью-источником (например, QStandardItemModel) и представлением (например, QTableView), позволяя динамически отображать только те данные, которые соответствуют определённым условиям.

Как работает QSortFilterProxyModel:

Фильтрация — позволяет отображать только те строки, которые соответствуют заданному шаблону.

Сортировка — позволяет сортировать строки модели по возрастанию или убыванию значений в выбранном столбце.

3. Какие преимущества и недостатки существуют у форматов CSV и JSON для хранения и обмена данными?

CSV

Преимущества:

- Простой и лёгкий формат — легко читается человеком и обрабатывается программами.
- Очень компактный — не содержит лишних структур, минимальный объём.
- Поддерживается большинством табличных редакторов (Excel, Google Sheets).

Быстрый парсинг — особенно для табличных данных.

Недостатки:

- Ограничен таблицей — только строки и столбцы, нет вложенных структур.
- Нет поддержки типов данных — всё строки; числа, даты и логика должны обрабатываться отдельно.
- Проблемы с символами-разделителями — например, если данные содержат запятые, нужны кавычки, и это может привести к ошибкам.
- Нет стандартной поддержки кодировки или метаданных.

JSON

- Преимущества:
- Поддержка вложенных структур — удобно хранить сложные объекты и списки.
- Читаемый формат — легко интерпретируется человеком.

- Широко поддерживается — во всех языках программирования и веб приложениях.
- Типизированность — различает строки, числа, логические значения, списки и объекты.
- Универсальность — подходит и для конфигураций, и для обмена данными, и для хранения.

Недостатки:

- Чуть тяжелее и медленнее — больше символов.
- Неэффективен для больших объёмов табличных данных — громоздкий по сравнению с CSV.
- Менее удобен для импорта в табличные редакторы — хотя современные системы уже умеют с ним работать.

4. Какие меры можно предпринять для валидации вводимых пользователем данных в диалоговом окне редактирования, и почему это важно?

Основные методы валидации в Qt:

Встроенные валидаторы Используются с QLineEdit и другими полями ввода:

QIntValidator — только целые числа.

QDoubleValidator — числа с плавающей точкой.

QRegExpValidator или QRegularExpressionValidator — для форматов, например, email.

Ограничение форматов ввода Например, маски ввода (setInputMask) для дат, телефонов и т.п.

Проверка вручную при нажатии «OK»