МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«ВЯТСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
Институт математики и информационных систем
Факультет автоматики и вычислительной техники
Кафедра электронных вычислительных машин

Отчет по лабораторной работе №3
по дисциплине
«Технологии программирования»

Выполнил студент гр. ИВТб-2301-05-00 _____/Макаров С.А./
Проверил преподаватель _____/Пащенко Д.Э./

Киров 2025

Цель

Цель: Освоить принципы разработки приложений на C++ с графическим интерфейсом, состоящих из нескольких окон, и организовать взаимодействие между ними.

Задание

Разработать оконное приложение на C++ с использованием WinAPI или Qt, содержащее как минимум одно главное окно и одно дочернее окно. Реализовать функционал передачи данных между ними.

Решение

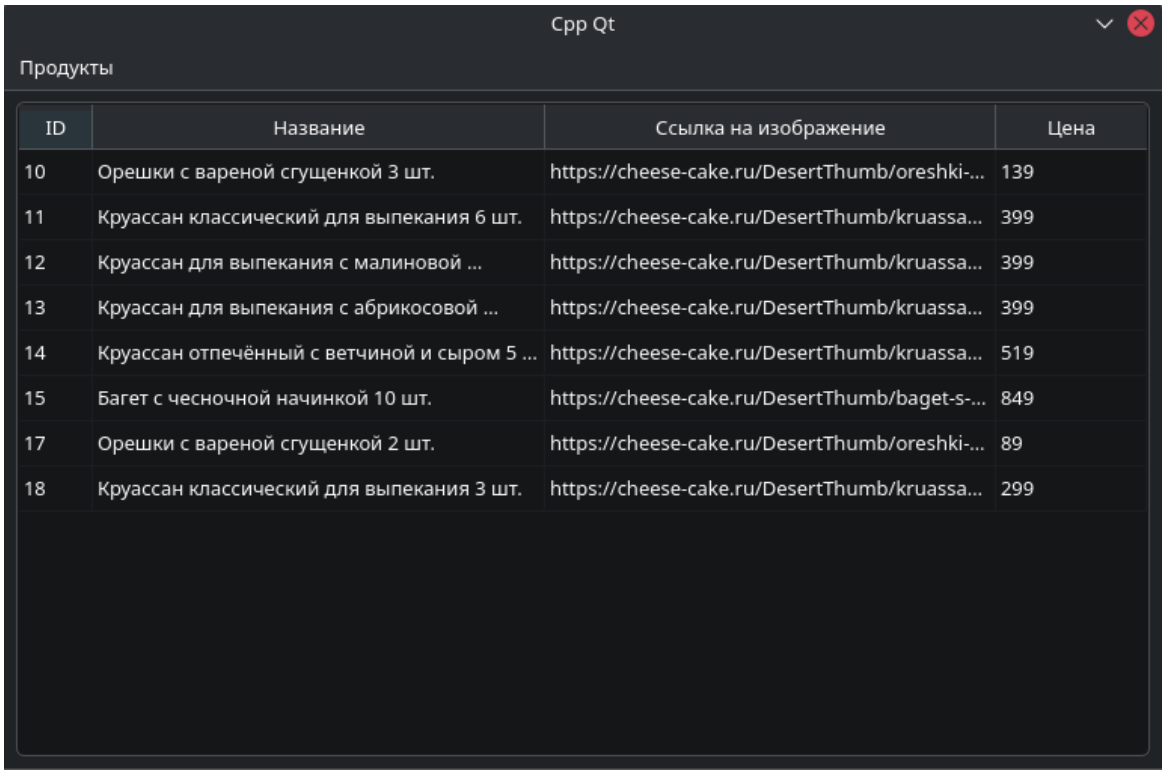

Рисунок 1 – Главное окно приложения

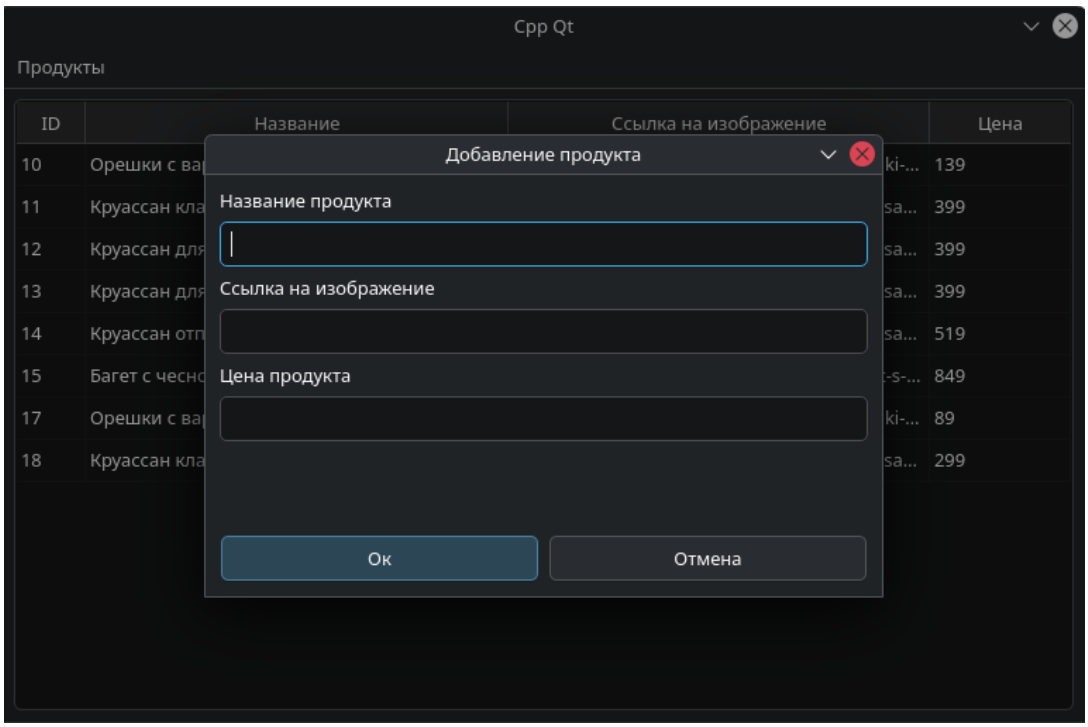

Рисунок 2 – Модальное окно добавления данных

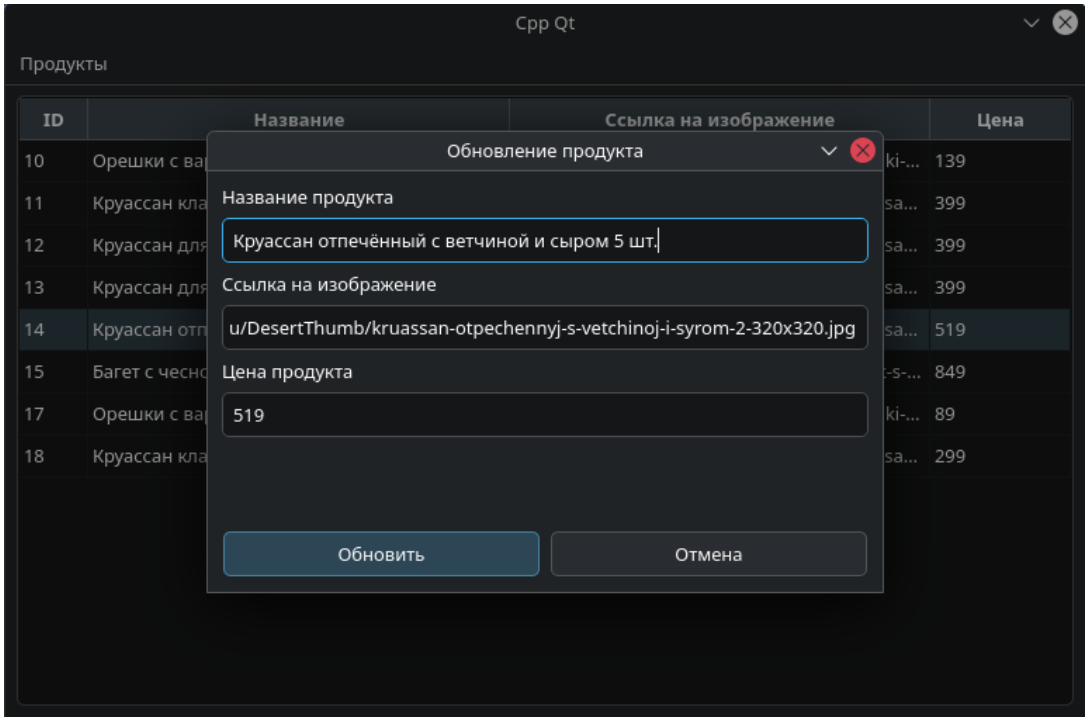

Рисунок 3 – Модальное окно обновления данных

Исходный код приложения представлен ниже:

```
#pragma once

#include <pqxx/pqxx>
#include <string>

class Database
{
private:
    std::shared_ptr<pqxx::connection> connection;

    bool connect(std::string dsn);

public:
    Database(std::string dsn);

    pqxx::result execute(std::string query);
};

#include "database.h"

#include <iostream>

Database::Database(std::string dsn)
{
    Database::connect(dsn);
}

bool Database::connect(std::string dsn)
{
    try
    {
        connection = std::make_shared<pqxx::connection>(dsn);

        if (connection->is_open()) {
            std::cout << "Connected to database: "
              << connection->dbname() << std::endl;

            return true;
        } else {
```

```
            std::cout << "Failed to connect to database: " << dsn << std::endl;

            return false;
        }
    }
    catch(const std::exception& e)
    {
        std::cerr << "Database connection error: " << e.what() << std::endl;

        return false;
    }
}

pqxx::result Database::execute(const std::string query)
{
    try
    {
        pqxx::work txn(*connection);

        auto result = txn.exec(query);

        txn.commit();

        return result;
    }
    catch(const std::exception& e)
    {
        std::cerr << "Query execution error: " << e.what() << std::endl;
    }
}

#pragma once

#include "../database/database.h"

#include <vector>
#include <string>
#include <format>
#include <iostream>
```

```
struct Product
{
    int id;
    std::string title;
    std::string image_url;
    int price;
};

struct ProductDto {
    std::string title;
    std::string image_url;
    int price;
};

class ProductService
{
private:
    Database &_db;

public:
    ProductService(Database &db);

    bool create(ProductDto dto);
    std::vector<Product> getAll();
    Product getById(int id);
    bool update(int id, ProductDto dto);
    bool remove(int id);
};

#include "product_service.h"

ProductService::ProductService(Database &db) : _db(db) {}

bool ProductService::create(ProductDto dto)
{
    try
    {
        _db.execute(
            std::format(
                "INSERT INTO products (title, image_url, price) "
```

```
                "VALUES ('{}', '{}', {})",
                dto.title,
                dto.image_url,
                std::to_string(dto.price)
            )
        );

        return true;
    }
    catch(const std::exception& e)
    {
        std::cerr << "Create product error: " << e.what() << std::endl;
        return false;
    }
}

std::vector<Product> ProductService::getAll()
{
    std::vector<Product> products;

    try
    {
        pqxx::result result = _db.execute(
            "SELECT * FROM products"
        );

        for (const auto& row : result) {
            Product product;

            product.id = row[0].as<int>();
            product.title = row[1].as<std::string>();
            product.image_url = row[2].as<std::string>();
            product.price = row[3].as<int>();

            products.push_back(product);
        }
    }
    catch(const std::exception& e)
    {
        std::cerr << "Get all products error: " << e.what() << std::endl;
```

```
    }

    return products;
}

Product ProductService::getById(int id)
{
    try
    {
        pqxx::result result = _db.execute(
            std::format("SELECT * FROM products WHERE id = {}", id)
        );

        if (!result.empty()) {
            const auto& row = result[0];
            Product product;
            product.id = row[0].as<int>();
            product.title = row[1].as<std::string>();
            product.image_url = row[2].as<std::string>();
            product.price = row[3].as<int>();
            return product;
        }
    }
    catch(const std::exception& e)
    {
        std::cerr << "Get product by id error: " << e.what() << std::endl;
    }

    return Product{-1, "", "", 0};
}

bool ProductService::update(int id, ProductDto dto)
{
    try
    {
        _db.execute(
            std::format(
                "UPDATE products SET "
                    "title = '{}', "
                    "image_url = '{}', "
```

```
                    "price = {} "
                "WHERE id = {}",
                dto.title,
                dto.image_url,
                dto.price,
                id
            )
        );

        return true;
    }
    catch(const std::exception& e)
    {
        std::cerr << "Update product error: " << e.what() << std::endl;
        return false;
    }
}

bool ProductService::remove(int id)
{
    try
    {
        _db.execute(
            std::format("DELETE FROM products WHERE id = {}", id)
        );

        return true;
    }
    catch(const std::exception& e)
    {
        std::cerr << "Remove product error: " << e.what() << std::endl;
        return false;
    }
}

#include "windows/main_window.h"

#include <QApplication>

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

```
{
    QApplication app(argc, argv);

    MainWindow window;
    window.show();

    return app.exec();
}

#include <vector>

#include <QApplication>
#include <QMainWindow>
#include <QVBoxLayout>
#include <QHBoxLayout>
#include <QMenuBar>
#include <QMenu>
#include <qaction.h>
#include <QTableWidget>
#include <QTableWidgetItem>
#include <QHeaderView>
#include <QMessageBox>

#include "../database/database.h"
#include "../modals/create_product_modal.h"
#include "../modals/update_product_modal.h"
#include "../services/product_service.h"

class MainWindow : public QMainWindow
{
private:
    Database db;

    ProductService product_service;

    int selected_product_id;

    std::vector<Product> products;

    QWidget *central_widget;
```

```
    QVBoxLayout *main_layout;

    QMenuBar *menu_bar;
    QMenu *product_menu;

    QAction *show_products_action;
    QAction *create_product_action;
    QAction *update_product_action;
    QAction *delete_product_action;

    QTableWidget *table = nullptr;

private slots:
    void onShowProducts();
    void onCreateProduct();
    void onUpdateProduct();
    void onDeleteProduct();

public:
    explicit MainWindow(QWidget *parent = nullptr);

    void setupTable();
    void updateTable();

    void selectProduct(int row, int column);
};

#include "main_window.h"

MainWindow::MainWindow(QWidget *parent)
    : QMainWindow(parent)
    , db("dbname=pt_db user=root password=root host=localhost port=5432")
    , product_service(db)
{
    central_widget = new QWidget();

    main_layout = new QVBoxLayout(central_widget);
    main_layout->setContentsMargins(8, 8, 8, 8);
    main_layout->addSpacing(0);
```

```
    setWindowTitle("Cpp Qt");
    setFixedSize(768, 480);
    setCentralWidget(central_widget);

    menu_bar = this->menuBar();

    product_menu = menu_bar->addMenu("Продукты");

    show_products_action = product_menu->addAction("Отобразить продукты");
    create_product_action = product_menu->addAction("Добавление продукта");
    update_product_action = product_menu->addAction("Обновление продукта");
    delete_product_action = product_menu->addAction("Удалить продкут");

    update_product_action->setDisabled(true);
    delete_product_action->setDisabled(true);

    connect(show_products_action, &QAction::triggered, this,
      &MainWindow::onShowProducts);
    connect(create_product_action, &QAction::triggered, this,
      &MainWindow::onCreateProduct);
    connect(update_product_action, &QAction::triggered, this,
      &MainWindow::onUpdateProduct);
    connect(delete_product_action, &QAction::triggered, this,
      &MainWindow::onDeleteProduct);
}

void MainWindow::onShowProducts()
{
    products = product_service.getAll();

    if (!table) {
        MainWindow::setupTable();
    }

    MainWindow::updateTable();
}

void MainWindow::onCreateProduct() {
    CreateProductModal *modal = new CreateProductModal(this);
```

```
    if (modal->exec() == QDialog::Accepted) {
        MainWindow::onShowProducts();
    }
}

void MainWindow::onUpdateProduct()
{
    UpdateProductModal *modal =
      new UpdateProductModal(selected_product_id, this);

    if (modal->exec() == QDialog::Accepted) {
        MainWindow::onShowProducts();
    }
}

void MainWindow::onDeleteProduct()
{
    QMessageBox::StandardButton reply = QMessageBox::question(
        this, "Подтверждение удаления",
        "Вы уверены, что хотите удалить запись?",
        QMessageBox::Yes | QMessageBox::No
    );

    if (reply == QMessageBox::Yes) {
        product_service.remove(selected_product_id);
        update_product_action->setDisabled(true);
        delete_product_action->setDisabled(true);
        selected_product_id = -1;
        MainWindow::onShowProducts();
    }
}

void MainWindow::setupTable()
{
    table = new QTableWidget();
    table->setColumnCount(4);
    table->setHorizontalHeaderLabels({"ID", "Название",
      "Ссылка на изображение", "Цена"});
    table->setSelectionBehavior(QTableWidget::SelectRows);
    table->verticalHeader()->setVisible(false);
```

```
    table->setColumnWidth(0, 20);
    table->setColumnWidth(3, 100);

    table->horizontalHeader()->setSectionResizeMode(0, QHeaderView::Fixed);
    table->horizontalHeader()->setSectionResizeMode(1, QHeaderView::Stretch);
    table->horizontalHeader()->setSectionResizeMode(2, QHeaderView::Stretch);
    table->horizontalHeader()->setSectionResizeMode(3, QHeaderView::Fixed);

    connect(table, &QTableWidget::cellClicked, this,
      &MainWindow::selectProduct);

    main_layout->addWidget(table);
}

void MainWindow::updateTable()
{
    table->setRowCount(products.size());

    for (int row = 0; row < products.size(); row++) {
        const auto& product = products[row];
        table->setItem(row, 0,
          new QTableWidgetItem(QString::number(product.id)));
        table->setItem(row, 1,
          new QTableWidgetItem(QString::fromStdString(product.title)));
        table->setItem(row, 2,
          new QTableWidgetItem(QString::fromStdString(product.image_url)));
        table->setItem(row, 3,
          new QTableWidgetItem(QString::number(product.price)));
    }
}

void MainWindow::selectProduct(int row, int column)
{
    selected_product_id = table->item(row, 0)->text().toInt();

    update_product_action->setDisabled(false);
    delete_product_action->setDisabled(false);
}
```

```
#include <vector>

#include <QWidget>
#include <QDialog>
#include <QVBoxLayout>
#include <QHBoxLayout>
#include <QPushButton>
#include <QLineEdit>
#include <QLabel>
#include <QMessageBox>

#include "../database/database.h"
#include "../services/product_service.h"

class UpdateProductModal : public QDialog
{
private:
    Database db;
    ProductService product_service;
    std::vector<Product> products;

    QVBoxLayout *main_layout;
    QHBoxLayout *button_layout;

    QLineEdit *title_input;
    QLineEdit *image_url_input;
    QLineEdit *price_input;

    QPushButton *submit_button;
    QPushButton *cancel_button;

    int product_id;

    void setupForm();
    void setupButtons();
    void handleSubmit();
    void validateIntInput(const QString &text);
    void loadProductData();

private slots:
```

```
    void onCancelClick();

public:
    explicit UpdateProductModal(int product_id, QWidget *parent = nullptr);
};

#include "update_product_modal.h"

UpdateProductModal::UpdateProductModal(int product_id, QWidget *parent)
    : QDialog(parent)
    , db("dbname=pt_db user=root password=root host=localhost port=5432")
    , product_service(db)
    , product_id(product_id)
{
    main_layout = new QVBoxLayout();

    setWindowTitle("Обновление продукта");
    setFixedSize(480, 300);
    setLayout(main_layout);

    products = product_service.getAll();

    UpdateProductModal::setupForm();
    UpdateProductModal::setupButtons();
    UpdateProductModal::loadProductData();
}

void UpdateProductModal::setupForm()
{
    title_input = new QLineEdit();
    image_url_input = new QLineEdit();
    price_input = new QLineEdit();

    connect(price_input, &QLineEdit::textChanged, this,
      &UpdateProductModal::validateIntInput);

    main_layout->addWidget(new QLabel("Название продукта"));
    main_layout->addWidget(title_input);
    main_layout->addWidget(new QLabel("Ссылка на изображение"));
    main_layout->addWidget(image_url_input);
```

```
    main_layout->addWidget(new QLabel("Цена продукта"));
    main_layout->addWidget(price_input);
}

void UpdateProductModal::setupButtons()
{
    button_layout = new QHBoxLayout();

    submit_button = new QPushButton("Обновить");
    cancel_button = new QPushButton("Отмена");

    connect(submit_button, &QPushButton::clicked, this,
      &UpdateProductModal::handleSubmit);
    connect(cancel_button, &QPushButton::clicked, this,
      &UpdateProductModal::onCancelClick);

    button_layout->addWidget(submit_button);
    button_layout->addWidget(cancel_button);

    main_layout->addStretch();
    main_layout->addLayout(button_layout);
}

void UpdateProductModal::loadProductData()
{
    auto product = product_service.getById(product_id);
    if (product.id != -1) {
        title_input->setText(QString::fromStdString(product.title));
        image_url_input->setText(QString::fromStdString(product.image_url));
        price_input->setText(QString::number(product.price));
    } else {
        QMessageBox::warning(this, "Ошибка", "Продукт не найден.");
        reject();
    }
}

void UpdateProductModal::handleSubmit()
{
    QString title = title_input->text().trimmed();
    QString image_url = image_url_input->text().trimmed();
```

```
    QString price = price_input->text().trimmed();

    if (title.isEmpty() || image_url.isEmpty() || price.isEmpty()) {
        QMessageBox::warning(this,
        "Ошибка обновления", "Все поля обязательны для заполнения.");
        return;
    }

    for (const auto& product : products) {
        if (product.title == title.toStdString() && product.id != product_id) {
            QMessageBox::warning(this,
            "Ошибка обновления", "Продукт с таким названием уже существует.");
            return;
        }
    }

    ProductDto dto;
    dto.title = title.toStdString();
    dto.image_url = image_url.toStdString();
    dto.price = price.toInt();

    bool success = product_service.update(product_id, dto);

    if (success) {
        accept();
    } else {
        QMessageBox::warning(this,
        "Ошибка обновления", "Не удалось обновить продукт.");
    }
}

void UpdateProductModal::validateIntInput(const QString &text)
{
    if (!text.isEmpty()) {
        for (const QChar &ch : text) {
            if (!ch.isDigit()) {
                QLineEdit *sender = qobject_cast<QLineEdit*>(this->sender());
                if (sender) {
                    QString currentText = sender->text();
                    sender->setText(currentText.left(currentText.length() - 1));
```

```
                    QMessageBox::warning(this,
                    "Ошибка ввода", "Разрешены только цифры");
                }
                break;
            }
        }
    }
}

void UpdateProductModal::onCancelClick()
{
    reject();
}

#include <vector>

#include <QWidget>
#include <QDialog>
#include <QVBoxLayout>
#include <QHBoxLayout>
#include <QPushButton>
#include <QLineEdit>
#include <QLabel>
#include <QMessageBox>

#include "../database/database.h"
#include "../services/product_service.h"

class CreateProductModal : public QDialog
{
private:
    Database db;

    ProductService product_service;

    std::vector<Product> products;

    QVBoxLayout *main_layout;
    QHBoxLayout *button_layout;
```

```
    QLineEdit *title_input;
    QLineEdit *image_url_input;
    QLineEdit *price_input;

    QPushButton *submit_button;
    QPushButton *cancel_button;

    void setupForm();
    void setupButtons();
    void handleSubmit();
    void validateIntInput(const QString &text);

private slots:
    void onCancelClick();

public:
    explicit CreateProductModal(QWidget *parent = nullptr);
};

#include "create_product_modal.h"

CreateProductModal::CreateProductModal(QWidget *parent)
    : QDialog(parent)
    , db("dbname=pt_db user=root password=root host=localhost port=5432")
    , product_service(db)
{
    main_layout = new QVBoxLayout();

    setWindowTitle("Добавление продукта");
    setFixedSize(480, 300);
    setLayout(main_layout);

    products = product_service.getAll();

    CreateProductModal::setupForm();
    CreateProductModal::setupButtons();
}

void CreateProductModal::setupForm()
{
```

```
    title_input = new QLineEdit();
    image_url_input = new QLineEdit();
    price_input = new QLineEdit();

    connect(price_input, &QLineEdit::textChanged, this,
      &CreateProductModal::validateIntInput);

    main_layout->addWidget(new QLabel("Название продукта"));
    main_layout->addWidget(title_input);
    main_layout->addWidget(new QLabel("Ссылка на изображение"));
    main_layout->addWidget(image_url_input);
    main_layout->addWidget(new QLabel("Цена продукта"));
    main_layout->addWidget(price_input);
}

void CreateProductModal::setupButtons()
{
    button_layout = new QHBoxLayout();

    submit_button = new QPushButton("Ок");
    cancel_button = new QPushButton("Отмена");

    connect(submit_button, &QPushButton::clicked, this,
      &CreateProductModal::handleSubmit);
    connect(cancel_button, &QPushButton::clicked, this,
      &CreateProductModal::onCancelClick);

    button_layout->addWidget(submit_button);
    button_layout->addWidget(cancel_button);

    main_layout->addStretch();
    main_layout->addLayout(button_layout);
}

void CreateProductModal::handleSubmit()
{
    QString title = title_input->text().trimmed();
    QString image_url = image_url_input->text().trimmed();
    QString price = price_input->text().trimmed();
```

```
    if (title.isEmpty() || image_url.isEmpty() || price.isEmpty()) {
        QMessageBox::warning(this,
        "Ошибка добавления", "Все поля обязательны для заполнения.");
        return;
    }

    for (const auto& product : products) {
        if (product.title == title) {
            QMessageBox::warning(this,
            "Ошибка добавления", "Продукт с таким названием уже существует.");
            return;
        }
    }

    ProductDto dto;

    dto.title = title.toStdString();
    dto.image_url = image_url.toStdString();
    dto.price = price.toInt();

    product_service.create(dto);

    accept();
}

void CreateProductModal::validateIntInput(const QString &text)
{
    if (!text.isEmpty()) {
        for (const QChar &ch : text) {
            if (!ch.isDigit()) {
                QLineEdit *sender = qobject_cast<QLineEdit*>(this->sender());
                if (sender) {
                    QString currentText = sender->text();
                    sender->setText(currentText.left(currentText.length() - 1));
                    QMessageBox::warning(
                      this, "Ошибка ввода", "Разрешены только цифры");
                }
                break;
            }
        }
```

```
    }
}

void CreateProductModal::onCancelClick()
{
    reject();
}
```

Вывод

В ходе выполнения лабораторной работы освоены принципы разработки приложений на C++, состоящее из нескольких окон. Разработано приложение, позволяющее создавать и редактировать записи в базе данных определенной предметной области.