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 №4
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 модульной архитектуры приложений, изучив механизмы явной и неявной загрузки библиотек. Сформировать понимание различий в подходах к интеграции компонентов программного обеспечения.

Задание

Модифицировать приложение из предыдущей лабораторной работы №3, разделив его на три независимых модуля с различными механизмами загрузки:

1. Вычислительный модуль – библиотека явной загрузки. Функционал:
   - Реализация алгоритмов, связанных с предметной областью.
   - Обработка данных, полученных от пользовательского интерфейса.
   - Одна из вычислительных функций должна быть на ассемблере.
2. Модуль пользовательского интерфейса – библиотека неявной загрузки. Функционал:
   - Главное и дочернее окна с сохранением всего функционала.
   - Элементы управления и обработчики событий.
   - Зависимость от интерфейса вычислительного модуля.
3. Основное приложение (загрузчик). Функционал:
   - Автоматическая загрузка библиотеки пользовательского интерфейса.
   - Динамическая загрузка и выгрузка вычислительной библиотеки.
   - Организация взаимодействия между модулями.

   Требования к реализации:

- Сохранить работоспособность всего функционала из предыдущей лабораторной работы.

– Обеспечить чёткое разделение ответственности между модулями.

– Предусмотреть механизмы обработки ошибок.

– Если у вас используется таблица, она должны иметь какие-то входные данные при запуске программы (например, их можно сохранять в json)

Решение

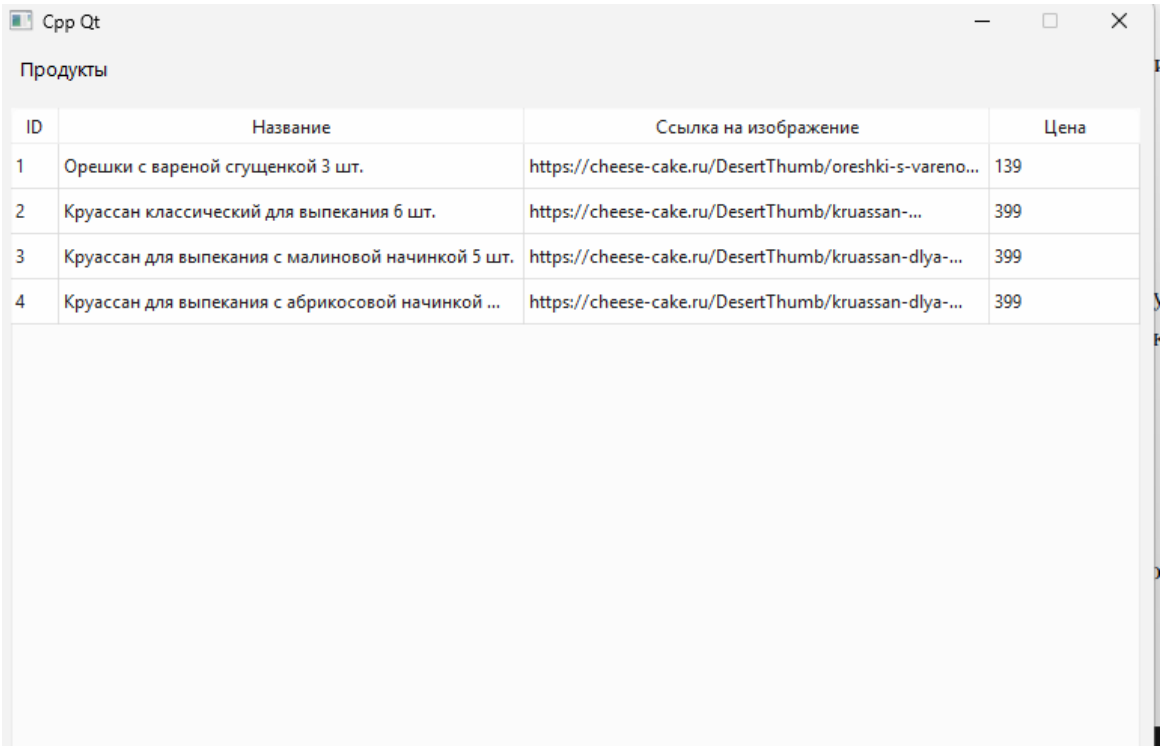

Рисунок 1 – Главное окно приложения

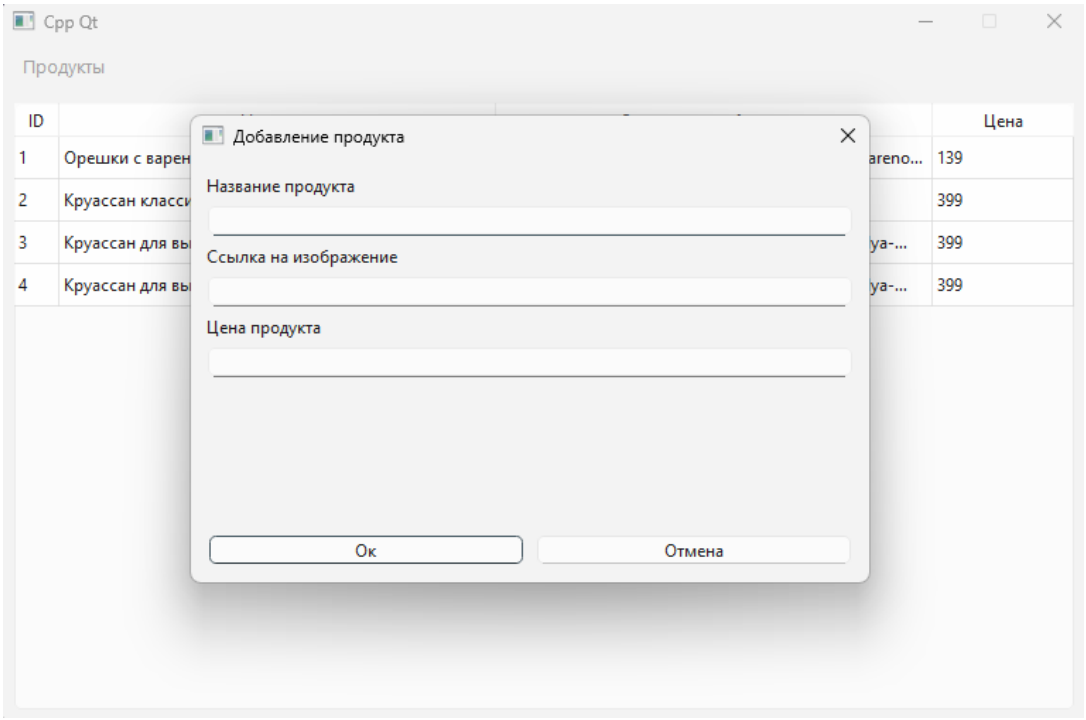

Рисунок 2 – Модальное окно добавления данных

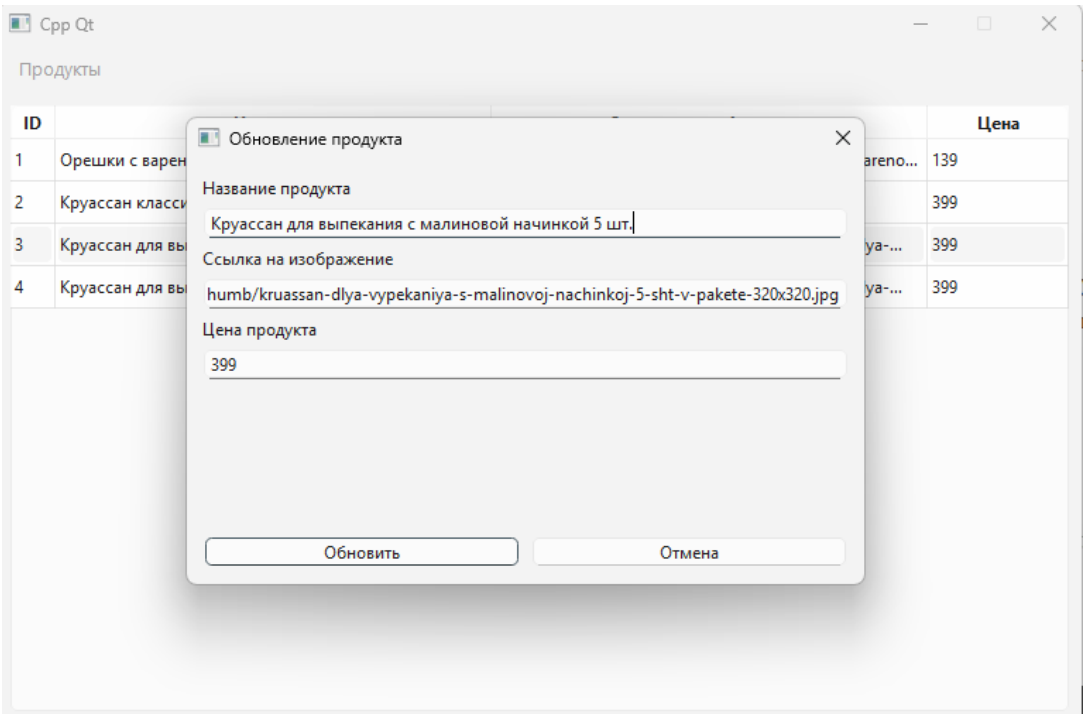

Рисунок 3 – Модальное окно обновления данных

Исходный код модуля проверки ввода целого числа, написанаая на ассемблере:

```
; src/asm/validate_int_input.asm
bits 64
default rel

section .text

global validateIntInputAsm
validateIntInputAsm:
    ; rcx = const char* str
    test rcx, rcx
    jz .return_false

.check_loop:
    mov al, [rcx]
    test al, al
    jz .return_true

    cmp al, '0'
    jl .return_false
    cmp al, '9'
    jg .return_false

    inc rcx
    jmp .check_loop

.return_true:
    mov rax, 1
    ret

.return_false:
    xor rax, rax
    ret
```

Исходный код модуля пользовательского интерфейса:

```
#ifndef PRODUCT_QT_LIB_H
#define PRODUCT_QT_LIB_H
```

```
#ifdef PRODUCT_QT_LIB_EXPORTS
    #define PRODUCT_QT_LIB_API __declspec(dllexport)
#else
    #define PRODUCT_QT_LIB_API __declspec(dllimport)
#endif

#include <QApplication>
#include <QMainWindow>
#include <QString>

struct PRODUCT_QT_LIB_API ProductDto {
    QString title;
    QString image_url;
    int price;

    ProductDto() : price(0) {}
    ProductDto(const QString& t, const QString& i, int p)
        : title(t), image_url(i), price(p) {}
};

struct PRODUCT_QT_LIB_API Product {
    int id;
    QString title;
    QString image_url;
    int price;

    Product() : id(-1), price(0) {}
    Product(int i, const QString& t, const QString& img, int p)
        : id(i), title(t), image_url(img), price(p) {}
};

class ProductRepositoryWrapper;

extern "C" {
    PRODUCT_QT_LIB_API int run_product_app(
        int argc, char *argv[], const char* dsn);

    PRODUCT_QT_LIB_API QMainWindow* create_main_window(
        const char* dsn, QWidget* parent = nullptr);
}
```

```
#endif

#include "../include/product_qt_lib.h"
#include "windows/main_window.h"
#include <QApplication>

extern "C" PRODUCT_QT_LIB_API int run_product_app(
    int argc, char *argv[], const char* dsn)
{
    QApplication app(argc, argv);

    MainWindow window(nullptr, dsn);
    window.show();

    return app.exec();
}

extern "C" PRODUCT_QT_LIB_API QMainWindow* create_main_window(
    const char* dsn, QWidget* parent)
{
    return new MainWindow(parent, dsn);
}

#ifndef PRODUCT_REPOSITORY_WRAPPER_H
#define PRODUCT_REPOSITORY_WRAPPER_H

#ifdef PRODUCT_QT_LIB_EXPORTS
    #define PRODUCT_QT_LIB_API __declspec(dllexport)
#else
    #define PRODUCT_QT_LIB_API __declspec(dllimport)
#endif


#include <string>
#include <vector>
#include <windows.h>
#include <QString>

struct InternalProductDto {
```

```
    std::string title;
    std::string image_url;
    int price;
};

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

class PRODUCT_QT_LIB_API ProductRepositoryWrapper {
private:
    HMODULE hLib = nullptr;
    void* repo = nullptr;

    struct ProductDto_C {
        const char* title;
        const char* image_url;
        int price;
    };

    struct Product_C {
        int id;
        char* title;
        char* image_url;
        int price;
    };

    using CreateRepoFunc = void*(__cdecl*)(const char*);
    using DestroyRepoFunc = void(__cdecl*)(void*);
    using CreateFunc = bool(__cdecl*)(void*, const void*);
    using GetAllFunc = int(__cdecl*)(void*, void**, size_t*);
    using GetByIdFunc = bool(__cdecl*)(void*, int, void*);
    using UpdateFunc = bool(__cdecl*)(void*, int, const void*);
    using RemoveFunc = bool(__cdecl*)(void*, int);
    using FreeProductsFunc = void(__cdecl*)(void*, size_t);
    using FreeProductFunc = void(__cdecl*)(void*);
```

```
    CreateRepoFunc create_product_repository = nullptr;
    DestroyRepoFunc destroy_product_repository = nullptr;
    CreateFunc product_repo_create = nullptr;
    GetAllFunc product_repo_get_all = nullptr;
    GetByIdFunc product_repo_get_by_id = nullptr;
    UpdateFunc product_repo_update = nullptr;
    RemoveFunc product_repo_remove = nullptr;
    FreeProductsFunc product_repo_free_products = nullptr;
    FreeProductFunc product_repo_free_product = nullptr;

    bool loadLibrary();

public:
    ProductRepositoryWrapper(const std::string& dsn);
    ~ProductRepositoryWrapper();

    bool create(const InternalProductDto& dto);
    std::vector<InternalProduct> getAll();
    InternalProduct getById(int id);
    bool update(int id, const InternalProductDto& dto);
    bool remove(int id);

    bool isConnected() const { return repo != nullptr; }
};

#endif

#include "product_repository_wrapper.h"
#include <iostream>
#include <cstring>

bool ProductRepositoryWrapper::loadLibrary() {
    hLib = LoadLibraryA("libproduct_db_lib.dll");

    if (!hLib) {
        std::cerr << "Library loading error: " << GetLastError() << std::endl;

        return false;
    }
```

```
    create_product_repository =
        (CreateRepoFunc)GetProcAddress(hLib, "create_product_repository");
    destroy_product_repository =
        (DestroyRepoFunc)GetProcAddress(hLib, "destroy_product_repository");
    product_repo_create =
        (CreateFunc)GetProcAddress(hLib, "product_repo_create");
    product_repo_get_all =
        (GetAllFunc)GetProcAddress(hLib, "product_repo_get_all");
    product_repo_get_by_id =
        (GetByIdFunc)GetProcAddress(hLib, "product_repo_get_by_id");
    product_repo_update =
        (UpdateFunc)GetProcAddress(hLib, "product_repo_update");
    product_repo_remove =
        (RemoveFunc)GetProcAddress(hLib, "product_repo_remove");
    product_repo_free_products =
        (FreeProductsFunc)GetProcAddress(hLib, "product_repo_free_products");
    product_repo_free_product =
        (FreeProductFunc)GetProcAddress(hLib, "product_repo_free_product");

    if (!create_product_repository || !destroy_product_repository ||
        !product_repo_create || !product_repo_get_all ||
        !product_repo_get_by_id || !product_repo_update ||
        !product_repo_remove || !product_repo_free_products) {

        std::cerr << "Failed to load functions from library" << std::endl;
        FreeLibrary(hLib);
        hLib = nullptr;

        return false;
    }

    return true;
}

ProductRepositoryWrapper::ProductRepositoryWrapper(const std::string& dsn) {
    if (!loadLibrary()) {
        return;
    }

    repo = create_product_repository(dsn.c_str());
```

```
    if (!repo) {
        std::cerr << "Failed to connect to the database" << std::endl;
        FreeLibrary(hLib);
        hLib = nullptr;
    }
}

ProductRepositoryWrapper::~ProductRepositoryWrapper() {
    if (repo) {
        destroy_product_repository(repo);
    }

    if (hLib) {
        FreeLibrary(hLib);
    }
}

bool ProductRepositoryWrapper::create(const InternalProductDto& dto) {
    if (!repo) return false;

    ProductDto_C dto_c;
    dto_c.title = dto.title.c_str();
    dto_c.image_url = dto.image_url.c_str();
    dto_c.price = dto.price;

    return product_repo_create(repo, &dto_c);
}

std::vector<InternalProduct> ProductRepositoryWrapper::getAll() {
    std::vector<InternalProduct> result;

    if (!repo) return result;

    Product_C* products_c = nullptr;
    size_t count = 0;

    if (product_repo_get_all(repo, (void**)&products_c, &count)
        == 0 && products_c) {
        for (size_t i = 0; i < count; ++i) {
```

```
            InternalProduct product;
            product.id = products_c[i].id;
            product.title = products_c[i].title
                ? products_c[i].title : "";
            product.image_url = products_c[i].image_url
                ? products_c[i].image_url : "";
            product.price = products_c[i].price;
            result.push_back(product);
        }

        product_repo_free_products(products_c, count);
    }

    return result;
}

InternalProduct ProductRepositoryWrapper::getById(int id) {
    InternalProduct product;
    product.id = -1;

    if (!repo) return product;

    Product_C product_c;
    if (product_repo_get_by_id(repo, id, &product_c)) {
        product.id = product_c.id;
        product.title = product_c.title ? product_c.title : "";
        product.image_url = product_c.image_url ? product_c.image_url : "";
        product.price = product_c.price;

        product_repo_free_product(&product_c);
    }

    return product;
}

bool ProductRepositoryWrapper::update(int id, const InternalProductDto& dto) {
    if (!repo) return false;

    ProductDto_C dto_c;
    dto_c.title = dto.title.c_str();
```

```
    dto_c.image_url = dto.image_url.c_str();
    dto_c.price = dto.price;

    return product_repo_update(repo, id, &dto_c);
}

bool ProductRepositoryWrapper::remove(int id) {
    if (!repo) return false;

    return product_repo_remove(repo, id);
}

#ifndef MAIN_WINDOW_H
#define MAIN_WINDOW_H

#ifdef PRODUCT_QT_LIB_EXPORTS
    #define PRODUCT_QT_LIB_API __declspec(dllexport)
#else
    #define PRODUCT_QT_LIB_API __declspec(dllimport)
#endif

#include "../product_repository_wrapper.h"
#include "../include/product_qt_lib.h"
#include "../modals/create_product_modal.h"
#include "../modals/update_product_modal.h"

#include <vector>

#include <QMainWindow>
#include <QWidget>
#include <QVBoxLayout>
#include <QHBoxLayout>
#include <QMenuBar>
#include <QMenu>
#include <QAction>
#include <QTableWidget>
#include <QTableWidgetItem>
#include <QHeaderView>
#include <QMessageBox>
```

```
class PRODUCT_QT_LIB_API MainWindow : public QMainWindow
{
    Q_OBJECT

private:
    ProductRepositoryWrapper product_repo;

    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;

    Product convertToProduct(const InternalProduct& internal);

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

public:
    explicit MainWindow(QWidget *parent = nullptr, const char* dsn = nullptr);
    ~MainWindow();

    void setupTable();
    void updateTable();

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

```
};

#endif

#include "main_window.h"

Product MainWindow::convertToProduct(const InternalProduct& internal) {
    return Product(
        internal.id,
        QString::fromStdString(internal.title),
        QString::fromStdString(internal.image_url),
        internal.price
    );
}

MainWindow::MainWindow(QWidget *parent, const char* dsn)
    : QMainWindow(parent)
    , product_repo(dsn ? dsn
        : "dbname=pt_db user=root password=root host=localhost port=5432")
{
    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);
}

MainWindow::~MainWindow() {
}

void MainWindow::onShowProducts()
{
    if (!product_repo.isConnected()) {
        QMessageBox::warning(this, "Ошибка", "Нет подключения к базе данных");
        return;
    }

    auto internalProducts = product_repo.getAll();
    products.clear();

    for (const auto& internal : internalProducts) {
        products.push_back(convertToProduct(internal));
    }

    if (!table) {
        setupTable();
    }

    updateTable();
}

void MainWindow::onCreateProduct() {
    CreateProductModal *modal = new CreateProductModal(&product_repo, this);

    if (modal->exec() == QDialog::Accepted) {
```

```
        onShowProducts();
    }
}

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

    if (modal->exec() == QDialog::Accepted) {
        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_repo.remove(selected_product_id);
        update_product_action->setDisabled(true);
        delete_product_action->setDisabled(true);
        selected_product_id = -1;
        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(product.title));
        table->setItem(row, 2,
            new QTableWidgetItem(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);
}

#ifndef CREATE_PRODUCT_MODAL_H
```

```
#define CREATE_PRODUCT_MODAL_H

#ifdef PRODUCT_QT_LIB_EXPORTS
#define PRODUCT_QT_LIB_API __declspec(dllexport)
#else
#define PRODUCT_QT_LIB_API __declspec(dllimport)
#endif

#include "../product_repository_wrapper.h"
#include "../include/product_qt_lib.h"

#include <vector>
#include <windows.h>

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

class PRODUCT_QT_LIB_API CreateProductModal : public QDialog
{
    Q_OBJECT

private:
    ProductRepositoryWrapper *product_repo;
    std::vector<InternalProduct> products;

    QVBoxLayout *main_layout;
    QHBoxLayout *button_layout;

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

    QPushButton *submit_button;
    QPushButton *cancel_button;
```

```
    HMODULE hValidateLib = nullptr;
    bool (*validateIntInputAsm)(const char *) = nullptr;

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

private slots:
    void onCancelClick();

public:
    explicit CreateProductModal(
        ProductRepositoryWrapper *repo, QWidget *parent = nullptr);
    ~CreateProductModal();
};

#endif

#include "create_product_modal.h"

CreateProductModal::CreateProductModal(
    ProductRepositoryWrapper *repo, QWidget *parent)
    : QDialog(parent), product_repo(repo)
{
    hValidateLib = LoadLibraryA("validate_int_lib.dll");
    if (hValidateLib)
    {
        validateIntInputAsm = reinterpret_cast<bool (*)(const char *)>(
            GetProcAddress(hValidateLib, "validateIntInputAsm"));
    }

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

    if (product_repo && product_repo->isConnected())
    {
        products = product_repo->getAll();
```

```
    }

    setupForm();
    setupButtons();
}

CreateProductModal::~CreateProductModal()
{
    if (hValidateLib)
    {
        FreeLibrary(hValidateLib);
    }
}

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()
{
    if (!product_repo || !product_repo->isConnected())
    {
        QMessageBox::warning(this, "Ошибка", "Нет подключения к базе данных");
        return;
    }

    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.toStdString())
        {
            QMessageBox::warning(this, "Ошибка добавления",
                "Продукт с таким названием уже существует.");
            return;
        }
    }

    InternalProductDto dto;
```

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

    if (product_repo->create(dto))
    {
        accept();
    }
    else
    {
        QMessageBox::warning(this, "Ошибка добавления",
            "Не удалось добавить продукт");
    }
}

void CreateProductModal::validateIntInput(const QString &text)
{
    std::string stdStr = text.toStdString();
    if (!validateIntInputAsm(stdStr.c_str()))
    {
        QLineEdit *sender = qobject_cast<QLineEdit *>(this->sender());
        if (sender && !text.isEmpty())
        {
            sender->setText(text.left(text.length() - 1));
            QMessageBox::warning(this, "Ошибка ввода",
                "Разрешены только цифры");
        }
    }
}

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

#ifndef UPDATE_PRODUCT_MODAL_H
#define UPDATE_PRODUCT_MODAL_H

#ifdef PRODUCT_QT_LIB_EXPORTS
#define PRODUCT_QT_LIB_API __declspec(dllexport)
```

```
#else
#define PRODUCT_QT_LIB_API __declspec(dllimport)
#endif

#include "../product_repository_wrapper.h"
#include "../include/product_qt_lib.h"

#include <vector>
#include <windows.h>

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

class PRODUCT_QT_LIB_API UpdateProductModal : public QDialog
{
    Q_OBJECT

private:
    ProductRepositoryWrapper *product_repo;
    std::vector<InternalProduct> 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;

    HMODULE hValidateLib = nullptr;
    bool (*validateIntInputAsm)(const char *) = nullptr;
```

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

private slots:
    void onCancelClick();

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

#endif

#include "update_product_modal.h"

UpdateProductModal::UpdateProductModal(
    ProductRepositoryWrapper *repo, int product_id, QWidget *parent)
    : QDialog(parent), product_repo(repo), product_id(product_id)
{
    hValidateLib = LoadLibraryA("validate_int_lib.dll");
    if (hValidateLib)
    {
        validateIntInputAsm = reinterpret_cast<bool (*)(const char *)>(
            GetProcAddress(hValidateLib, "validateIntInputAsm"));
    }

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

    if (product_repo && product_repo->isConnected())
    {
        products = product_repo->getAll();
    }
```

```
    setupForm();
    setupButtons();
    loadProductData();
}

UpdateProductModal::~UpdateProductModal()
{
    if (hValidateLib)
    {
        FreeLibrary(hValidateLib);
    }
}

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()
{
    if (!product_repo || !product_repo->isConnected())
    {
        QMessageBox::warning(this, "Ошибка",
            "Нет подключения к базе данных");
        reject();
        return;
    }

    auto product = product_repo->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()
{
    if (!product_repo || !product_repo->isConnected())
    {
        QMessageBox::warning(this, "Ошибка", "Нет подключения к базе данных");
        return;
```

```
    }

    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;
        }
    }

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

    bool success = product_repo->update(product_id, dto);

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

```
void UpdateProductModal::validateIntInput(const QString &text)
{
    std::string stdStr = text.toStdString();
    if (!validateIntInputAsm(stdStr.c_str()))
    {
        QLineEdit *sender = qobject_cast<QLineEdit *>(this->sender());
        if (sender && !text.isEmpty())
        {
            sender->setText(text.left(text.length() - 1));
            QMessageBox::warning(this, "Ошибка ввода",
                "Разрешены только цифры");
        }
    }
}

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

Исходный код модуля обращения к базе данных:

```
#pragma once

#include <cstddef>

struct ProductDto {
    const char* title;
    const char* image_url;
    int         price;
};

struct Product {
    int    id;
    char*  title;
    char*  image_url;
    int    price;
};

#ifdef __cplusplus
```

```
extern "C" {
#endif

void* create_product_repository(const char* dsn);
void destroy_product_repository(void* repo);
bool product_repo_create(void* repo, const ProductDto* dto);
int  product_repo_get_all(void* repo,
    Product** out_products, size_t* out_count);
bool product_repo_get_by_id(void* repo, int id, Product* out_product);
bool product_repo_update(void* repo, int id, const ProductDto* dto);
bool product_repo_remove(void* repo, int id);
void product_repo_free_products(Product* products, size_t count);
void product_repo_free_product(Product* product);

#ifdef __cplusplus
}
#endif

#include "../include/product_db_lib.h"

#include <pqxx/pqxx>
#include <iostream>
#include <format>
#include <vector>
#include <cstring>
#include <memory>

static char* cstr_dup(const std::string& s)
{
    if (s.empty()) return nullptr;
    char* p = static_cast<char*>(std::malloc(s.size() + 1));
    if (p) std::memcpy(p, s.c_str(), s.size() + 1);
    return p;
}

class ProductRepository
{
private:
    std::shared_ptr<pqxx::connection> conn;
```

```
    bool connect(const std::string& dsn)
    {
        if (dsn.empty()) return false;
        try {
            conn = std::make_shared<pqxx::connection>(dsn);
            if (conn->is_open()) {
                std::cout << "Connected to database: "
                    << conn->dbname() << std::endl;
                return true;
            }
        }
        catch (const std::exception& e) {
            std::cerr << "DB connection failed: " << e.what() << std::endl;
        }
        return false;
    }

    void exec(const std::string& sql)
    {
        pqxx::work txn(*conn);
        txn.exec(sql);
        txn.commit();
    }

public:
    explicit ProductRepository(const char* dsn)
    {
        connect(dsn ? dsn : "");
    }

    bool is_connected() const noexcept
    {
        return conn && conn->is_open();
    }

    bool create(const char* title, const char* image_url, int price)
    {
        if (!title || !image_url) return false;
        try {
            exec(std::format(
```

```
            "INSERT INTO products (title, image_url, price)
                VALUES ({}, {}, {})",
            conn->quote(title),
            conn->quote(image_url),
            price
        ));
        return true;
    }
    catch (...) { return false; }
}

std::vector<Product> getAll()
{
    std::vector<Product> result;
    try {
        pqxx::result r = pqxx::work(*conn).exec(
            "SELECT id, title, image_url, price
                FROM products ORDER BY id");
        for (const auto& row : r) {
            Product p{};
            p.id        = row["id"].as<int>();
            p.title     = cstr_dup(row["title"].as<std::string>());
            p.image_url = cstr_dup(row["image_url"].as<std::string>());
            p.price     = row["price"].as<int>();
            result.emplace_back(std::move(p));
        }
    }
    catch (...) {}
    return result;
}

bool getById(int id, Product& out)
{
    try {
        pqxx::result r = pqxx::work(*conn).exec(std::format(
            "SELECT id, title, image_url, price FROM products
            WHERE id = {}", id));
        if (r.empty()) return false;
        const auto& row = r[0];
        out.id        = row["id"].as<int>();
```

```
            out.title     = cstr_dup(row["title"].as<std::string>());
            out.image_url = cstr_dup(row["image_url"].as<std::string>());
            out.price     = row["price"].as<int>();
            return true;
        }
        catch (...) { return false; }
    }

    bool update(int id, const char* title, const char* image_url, int price)
    {
        if (!title || !image_url) return false;
        try {
            exec(std::format(
                "UPDATE products SET title = {}, image_url = {}, price = {}
                WHERE id = {}",
                conn->quote(title),
                conn->quote(image_url),
                price, id
            ));
            return true;
        }
        catch (...) { return false; }
    }

    bool remove(int id)
    {
        try {
            exec(std::format("DELETE FROM products WHERE id = {}", id));
            return true;
        }
        catch (...) { return false; }
    }
};

extern "C" {

__declspec(dllexport) void* create_product_repository(const char* dsn)
{
    auto* repo = new ProductRepository(dsn);
    if (repo->is_connected())
```

```
        return repo;
    delete repo;
    return nullptr;
}

__declspec(dllexport) void destroy_product_repository(void* repo)
{
    delete static_cast<ProductRepository*>(repo);
}

__declspec(dllexport) bool product_repo_create(
    void* repo, const ProductDto* dto)
{
    return repo && dto && static_cast<ProductRepository*>(repo)
        ->create(dto->title, dto->image_url, dto->price);
}

__declspec(dllexport) int product_repo_get_all(
    void* repo, Product** out_products, size_t* out_count)
{
    if (!repo || !out_products || !out_count) return -1;
    auto vec = static_cast<ProductRepository*>(repo)->getAll();
    if (vec.empty()) {
        *out_products = nullptr;
        *out_count = 0;
        return 0;
    }
    *out_products = static_cast<Product*>(
        std::malloc(vec.size() * sizeof(Product)));
    if (!*out_products) return -1;
    std::memcpy(*out_products, vec.data(), vec.size() * sizeof(Product));
    *out_count = vec.size();
    return 0;
}

__declspec(dllexport) bool product_repo_get_by_id(
    void* repo, int id, Product* out)
{
    return repo && out && static_cast<ProductRepository*>(repo)
        ->getById(id, *out);
```

```
}

__declspec(dllexport) bool product_repo_update(
    void* repo, int id, const ProductDto* dto)
{
    return repo && dto && static_cast<ProductRepository*>(repo)
        ->update(id, dto->title, dto->image_url, dto->price);
}

__declspec(dllexport) bool product_repo_remove(void* repo, int id)
{
    return repo && static_cast<ProductRepository*>(repo)->remove(id);
}

__declspec(dllexport) void product_repo_free_products(
    Product* products, size_t count)
{
    if (!products) return;
    for (size_t i = 0; i < count; ++i) {
        std::free(products[i].title);
        std::free(products[i].image_url);
    }
    std::free(products);
}

__declspec(dllexport) void product_repo_free_product(Product* p)
{
    if (!p) return;
    std::free(p->title);
    std::free(p->image_url);
}

}
```

Исходный код главного модуля приложения:

```
#include <product_qt_lib.h>

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

```
    return run_product_app(argc, argv,
        "dbname=pt_db user=root password=root host=localhost port=5432");
}
```

Вывод

В ходе выполнения лабораторной работы освоены принципы модульной архитектуры приложений, изучены механизмы явной и неявной загрузки. Разделено приложение из лабораторной работы №3 на модули пользовательского интерфейса, модуль сообщения с базой данных, а также модуль обработки ввода целочисленного числа переписан на ассемблер.