КАФЕДРА

ОТЧЕТ
ЗАЩИЩЕН С ОЦЕНКОЙ

РУКОВОДИТЕЛЬ

должность, уч. степень, звание	подпись, дата	инициалы, фамилия

Отчет о лабораторной работе №7

Создание двумерного пользовательского интерфейса в среде Qt Creator

По дисциплине: Проектирование человеко-машинного интерфейса

РАБОТУ ВЫПОЛНИЛИ

СТУДЕНТЫ ГР. № _____

	подпись, дата	инициалы, фамилия

Санкт-Петербург 2024

Цель работы:
Освоение кросс-платформенной интегрированной среды разработки, изучение базовых возможностей разработки и отладки приложения с двумерным пользовательским интерфейсом с использованием библиотеки Qt и компилятора MinGW

Задание:
Разработать и отладить приложение на языке C++ с графическим пользовательским интерфейсом, использующим основные виджеты Qt

Название и версия используемой среды моделирования:
QT Creator 11.0.3
Based on Qt 6.4.3 (Clang 13.0 (Apple), arm64)
Built on Sep 27 2023 06:47:35

Описание структуры интерфейса:
Интерфейс программы разделен на несколько логически связанных областей, каждая из которых обслуживает ключевые функции приложения для проката автомобилей. Основные компоненты включают:

1. Заголовок и логотип: В верхней части окна размещены изображения, состоящие из логотипа Porsche и фотографий автомобилей. Эти элементы создают визуальное оформление интерфейса, поддерживая тематику автопроката.
2. Поле для ввода текста: В интерфейсе присутствуют текстовые поля, позволяющие пользователю вводить данные о клиенте и автомобилях. Например, поле для имени клиента, модель автомобиля, и другие данные, необходимые для регистрации аренды.
3. Выпадающие списки: Для выбора предопределенных значений, таких как марки автомобилей или существующие клиенты, использованы выпадающие списки (ComboBox). Это позволяет ускорить процесс выбора из заранее внесенных данных.
4. Таблицы: Три таблицы отображают данные о:
    - Автомобилях в автопарке (модель, марка, регистрационный номер и статус).
    - Клиентах (имя, фамилия, контактные данные).
    - Истории операций (какой клиент, какой автомобиль и когда арендовал/вернул).
5. Кнопки: Несколько кнопок позволяют взаимодействовать с приложением. Среди них кнопки для добавления нового клиента, внесения нового автомобиля в базу, передачи автомобиля клиенту и возврата автомобиля.
6. Интерактивные элементы для дат: В интерфейсе также имеются виджеты для ввода дат начала аренды и даты возврата автомобилей, что упрощает управление временными данными.

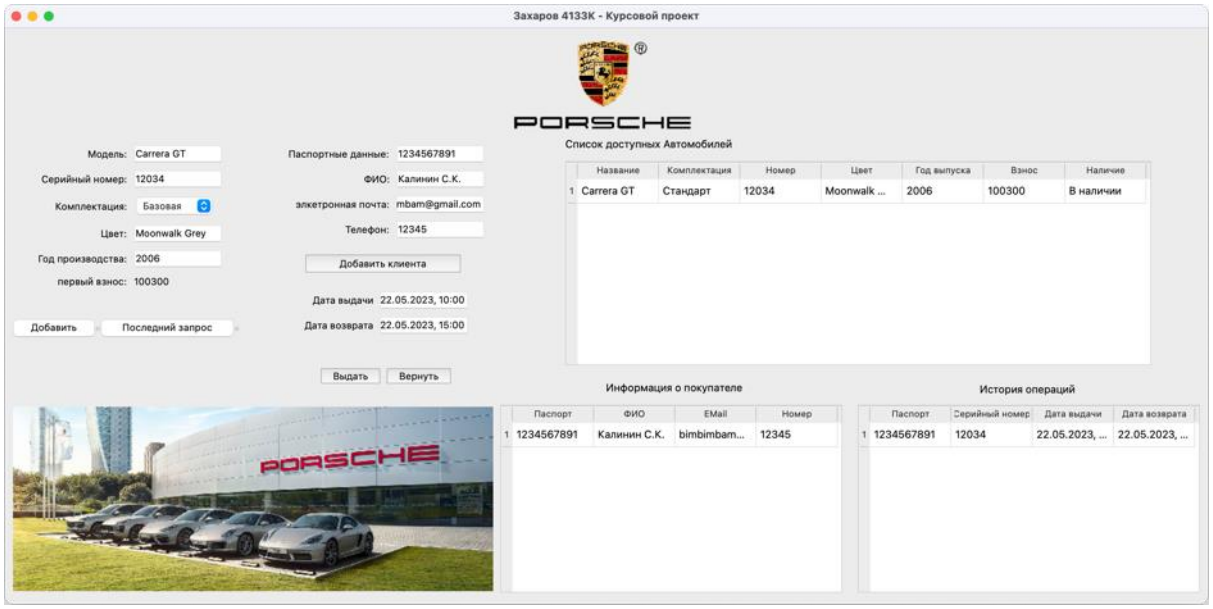

Рисунок 1 – общий вид приложения

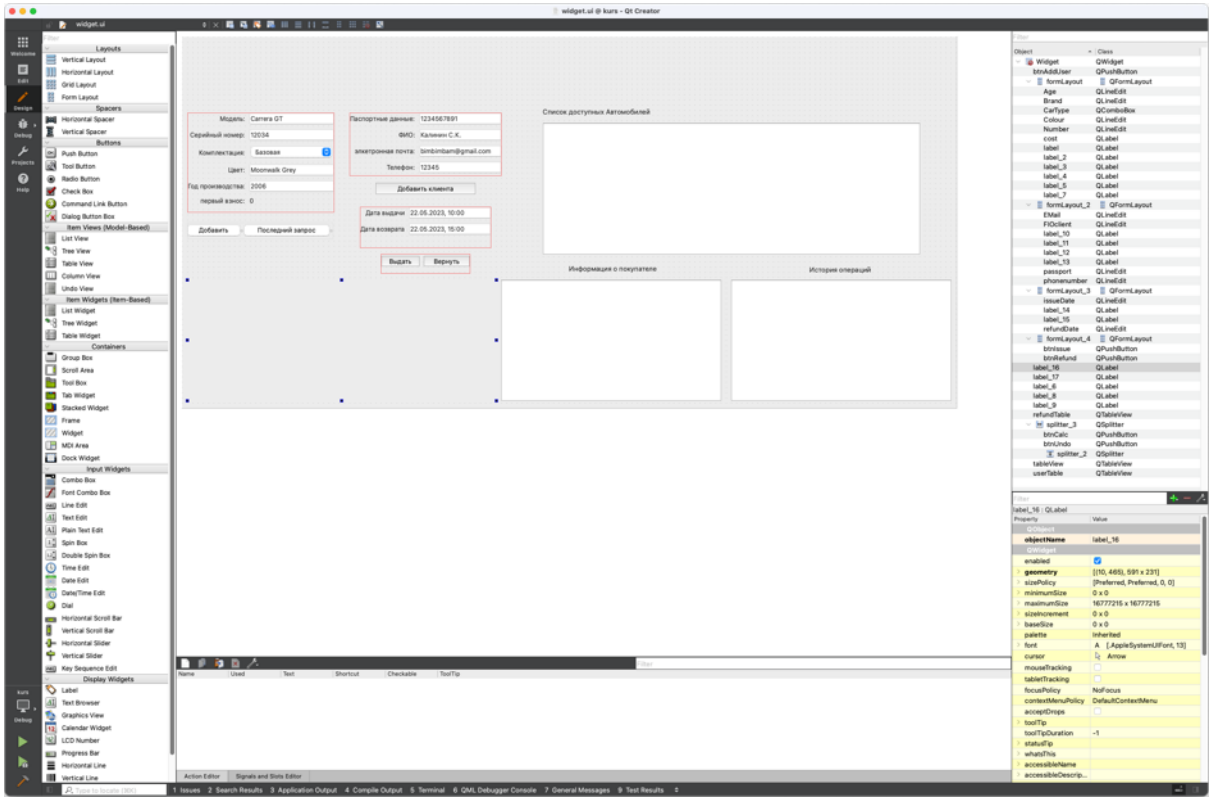

Рисунок 2 – вид редактора, используемые объекты

В данной работе были использованы следующие элементы интерфейса:

- **QWidget** — основной контейнер для виджетов.
- **QPushButton (btnAddUser, btnIssue, btnRefund, btnCalc, btnUndo)** — кнопки для добавления пользователей, выдачи и возврата автомобилей, выполнения расчетов и отмены действий.
- **QFormLayout** — компоновка элементов интерфейса для ввода данных.

- **QLineEdit (Age, Brand, CarType, Colour, Number, EMail, FIOclient, Passport, PhoneNumber, IssueDate, RefundDate)** — текстовые поля для ввода информации о клиентах и машинах.
- **QComboBox (CarType)** — выпадающий список для выбора типа автомобиля.
- **QLabel** — текстовые метки для обозначения полей ввода и отображения информации.
- **QTableView (RefundTable, TableView, UserTable)** — таблицы для отображения данных об автомобилях, клиентах и истории операций.
- **Splitter** — разделители интерфейса для более удобного размещения элементов.

Эти компоненты были использованы для создания удобного и функционального интерфейса приложения, обеспечивающего работу с клиентами, автопарком и операциями проката.

Исходный текст приложения, основной код для интерфейса:
View_controller:

```
#include <QMessageBox>
#include "view_controller.h"
View_Controller::View_Controller(QObject *parent)
   : QObject{parent}
{
}
void View_Controller::addToTableView(car* lastObject, QTableView* tableView)//cool
{
   CalculationFacade cur;
   // Получаем указатель на модель данных
   QStandardItemModel* model = dynamic_cast<QStandardItemModel*>(tableView->model());
   if (!model)
   {
   // Если модель данных не является типом QStandardItemModel, создаем новую модель
   model = new QStandardItemModel(tableView);
   tableView->setModel(model);
   model->setHorizontalHeaderLabels({"Название","Комплектация","Номер","Цвет","Год выпуска","Взнос","Наличие"});
   }
   // Получаем количество строк в таблице
   int rowCount = model->rowCount();
   // Создаем новую строку в модели данных
   QList<QStandardItem*> newRow;
   // Создаем элементы для каждого столбца таблицы
   QString condition = "В наличии";
   QStandardItem* typeItem = new QStandardItem(lastObject->getTypeString());
   QStandardItem* ageItem = new QStandardItem(QString::number(lastObject->getAge()));
   //QStandardItem* residentsItem = new QStandardItem(QString::number(lastObject->getResidents()));
   //QStandardItem* monthsItem = new QStandardItem(QString::number(lastObject->getMonthsForMVC()));
```

```
    //QStandardItem* priceItem = new QStandardItem(QString::number(lastObject->getPrice()));
    QStandardItem* BrandItem = new QStandardItem(lastObject->getBrand());
    QStandardItem* ColourItem = new QStandardItem(lastObject->getColour());
    QStandardItem* NumberItem = new QStandardItem(QString::number(lastObject->getNumber()));
    //QStandardItem* TitleItem = new QStandardItem(lastObject->getTitle());
    QStandardItem* CostItem = new QStandardItem(QString::number(cur.getCost(lastObject)));
    QStandardItem* conditionItem = new QStandardItem(condition);
    // Добавляем созданные элементы в новую строку
    newRow.append(BrandItem);
    newRow.append(typeItem);
    //newRow.append(priceItem);
    newRow.append(NumberItem);
    newRow.append(ColourItem);
    newRow.append(ageItem);
    newRow.append(CostItem);
    newRow.append(conditionItem);
   // Добавляем новую строку в модель данных
    model->insertRow(rowCount, newRow);
    // Обновляем таблицу
    tableView->viewport()->update();
}
void View_Controller::addToClientTable(client* Object,QTableView* tableClient){//cool
    // Получаем указатель на модель данных
    QStandardItemModel* model = dynamic_cast<QStandardItemModel*>(tableClient->model());
    if (!model)
    {
    // Если модель данных не является типом QStandardItemModel, создаем новую модель
    model = new QStandardItemModel(tableClient);
    tableClient->setModel(model);
    model->setHorizontalHeaderLabels({"Паспорт","ФИО","EMail","Номер"});
    }
    int rowCount = model->rowCount();
    // Создаем новую строку в модели данных
    QList<QStandardItem*> newRow;
    // Создаем элементы для каждого столбца таблицы
    QStandardItem* PassportItem = new QStandardItem(QString::number(Object->getPassport()));
    QStandardItem* FIOItem = new QStandardItem(Object->getFIO());
    QStandardItem* EMailItem = new QStandardItem(Object->getEMail());
    QStandardItem* PhoneNumberItem = new QStandardItem(QString::number(Object->getPhoneNumber()));
    // Добавляем созданные элементы в новую строку
    newRow.append(PassportItem);
    newRow.append(FIOItem);
    //newRow.append(priceItem);
    newRow.append(EMailItem);
    newRow.append(PhoneNumberItem);
```

```
   // Добавляем новую строку в модель данных
    model->insertRow(rowCount, newRow);
    // Обновляем таблицу
    tableClient->viewport()->update();
}
rent*       View_Controller::addToTableRefund(QTableView*       tableRefund,QTableView*
tableClient,QTableView* tableView,cars& carsinfo,QLineEdit* issueDate){//остальные две
таблицы тоже нужны, и список книг и лайн эдит с датой
    int curid;
    QString curids;
// Получаем выделенные элементы из двух предыдущих таблиц
    QModelIndexList       selectedPersonIndexes       =       tableClient->selectionModel()-
>selectedIndexes();
    QModelIndexList selectedCarIndexes = tableView->selectionModel()->selectedIndexes();
    QModelIndex index = tableView->currentIndex();
    //получаем указатель на модель данных для замены поля наличия
    QStandardItemModel*  model1  =  dynamic_cast<QStandardItemModel*>(tableView-
>model());
    // Изменяем значение в четвертом столбце выбранной строки
    QStandardItem* item = model1->itemFromIndex(index.sibling(index.row(), 6)); // 3 -
индекс четвертого столбца
// Получаем номер читательского билета и идентификатор книги из выделенных
элементов
    QString personNumber = selectedPersonIndexes.at(0).data().toString();
    QString carNumber = selectedCarIndexes.at(2).data().toString();
    for(int i =0;i<carsinfo.array.size();i++){
        curid=carsinfo.array[i]->Number;
        curids=QString::number(curid);
        if(curids==carNumber){
            if (carsinfo.array[i]->condition==false){
                QMessageBox::warning(nullptr, "Ошибка", "Выбранный автомобиль уже выдан
другому клиенту.");
                //проверяете состояние автомобиля, используя condition
                                        //Если автомобиль уже взят в аренду (т.е. condition
== true),
                                        //не разрешаем его повторное взятие, и функция
возвращает управление, и новая запись о выдаче в таблицу не добавляется.
                return nullptr;
            }
            else{
                carsinfo.array[i]->condition=false;
                item->setData("Выдана", Qt::DisplayRole);
                //selectedCarIndexes.at(6).data()="Выдана";
                //ui->clientTable->selectionModel()->selectedIndexes().at(6).data()=cond;
                break;
            }
        }
    }
    // Получаем данные для поля "Дата выдачи" из LineEdit
    QString issueDate1 = issueDate->text();
    //extradition* value (personNumber.toInt(),carNumber.toInt(),issueDate);
```

```
   QStandardItemModel* model = dynamic_cast<QStandardItemModel*>(tableRefund->model());
   if (!model)
   {
   // Если модель данных не является типом QStandardItemModel, создаем новую модель
   model = new QStandardItemModel(tableRefund);
   tableRefund->setModel(model);
   model->setHorizontalHeaderLabels({"Паспорт", "Серийный номер", "Дата выдачи", "Дата возврата"});
   }
   int rowCount = model->rowCount();
   // Создаем новую строку в модели данных
   QList<QStandardItem*> newRow;
   // Создаем объекты для хранения данных
   QString returnDate = ""; // Поле "Дата возврата" не заполняется
   QStandardItem* personNumberItem = new QStandardItem(personNumber);
   QStandardItem* carNumberItem = new QStandardItem(carNumber);
   QStandardItem* issueDateItem = new QStandardItem(issueDate1);
   QStandardItem* returnDateItem = new QStandardItem(returnDate);
   // Добавляем созданные элементы в новую строку
   newRow.append(personNumberItem);
   newRow.append(carNumberItem);
   newRow.append(issueDateItem);
   newRow.append(returnDateItem);
   // Добавляем новую строку в модель данных
   model->insertRow(rowCount, newRow);
   tableRefund->viewport()->update();
   tableView->viewport()->update();

   return new rent(personNumber.toInt(),carNumber.toInt(),issueDate1);
}
void View_Controller::setRefundData(QTableView* refundtable,QTableView* booktable,rented& rentinfo,QLineEdit* refundDate,cars& carsinfo){//лайн эдит с датой, список выдач, тблица книг
   // Получаем индекс выбранной строки в таблице
   QModelIndex index = refundtable->currentIndex();
   // Получаем значение из LineEdit
   QString value = refundDate->text();//////////////////////////////////////////////////////////////
   // Получаем модель данных, которая отображается в таблице
   QStandardItemModel* model = dynamic_cast<QStandardItemModel*>(refundtable->model());
   QModelIndexList selectedCarIndexes = refundtable->selectionModel()->selectedIndexes();
   QString carNumber1 = selectedCarIndexes.at(1).data().toString();
   int curid;
   for(int i =0;i<rentinfo.array.size();i++){
      curid=rentinfo.array[i]->CarNumber;
      //curids=QString::number(curid);
      if(curid==carNumber1.toInt()){
         if (rentinfo.array[i]->getRefund()==" "){
            rentinfo.array[i]->setRefund(value);
            //item->setData("Выдана", Qt::DisplayRole);
```

```
            //selectedBookIndexes.at(6).data()="Выдана";
            //ui->userTable->selectionModel()->selectedIndexes().at(6).data()=cond;
            break;
        }
    }
}
curid=0;
//QModelIndexList selectedBookIndexes = booktable->selectionModel()->selectedIndexes();
//QString bookId = selectedBookIndexes.at(2).data().toString();
for(int i =0;i<carsinfo.array.size();i++){
    curid=carsinfo.array[i]->Number;
    //curids=QString::number(curid);
    if(curid==carNumber1.toInt()){
        if (carsinfo.array[i]->condition==true){                    //после возврата автомобиля обновляется состояние автомобиля
                                                                    //Теперь, этот автомобиль можно вновь выдать в аренду.
            return;
        }
        else{
            carsinfo.array[i]->condition=true;
            //item->setData("Выдана", Qt::DisplayRole);
            //selectedBookIndexes.at(6).data()="Выдана";
            //ui->userTable->selectionModel()->selectedIndexes().at(6).data()=cond;
            break;
        }
    }
}
// Изменяем значение в четвертом столбце выбранной строки
QStandardItem* item = model->itemFromIndex(index.sibling(index.row(), 3)); // 3 - индекс четвертого столбца
item->setData(value, Qt::DisplayRole);
// Получаем идентификатор машины из выбранной строки в таблице выдач
QModelIndex indexIssued = refundtable->currentIndex();
QString carNumber2 = indexIssued.sibling(indexIssued.row(), 1).data().toString(); // 1 - индекс второго столбца
// Находим элемент в таблице машин с таким же идентификатором
QStandardItemModel* modelCars = dynamic_cast<QStandardItemModel*>(booktable->model());
int rowCount = modelCars->rowCount();
int bookRow = -1;
for (int i = 0; i < rowCount; i++) {
    QModelIndex indexBook = modelCars->index(i, 2); // 0 - индекс первого столбцаъ
    if (indexBook.data().toString() == carNumber2) {
        bookRow = i;
        break;
    }
}
// Если элемент найден, то изменяем значение в шестом столбце
if (bookRow != -1) {
```

```
        QStandardItem* item = modelCars->itemFromIndex(modelCars->index(bookRow, 6)); //
5 - индекс шестого столбца (наличие)
        item->setData("В наличии", Qt::DisplayRole);

        // Обновляем модель данных в таблице машин
        booktable->setModel(modelCars);
    }
    // Обновляем модель данных в таблице
    refundtable->setModel(model);
}
```

Widget.cpp:

```
#include "widget.h"
#include "ui_widget.h"
#include <QPixmap>
#include <QMessageBox>
#include <QFile>
#include <QTextStream>
#include <QTableWidgetItem>
#include <QFileDialog>

Widget::Widget(QWidget *parent)
    : QWidget(parent)
    , ui(new Ui::Widget)
    , info(this)
{
    ui->setupUi(this);
    ui->btnUndo->setEnabled(false);
    QPixmap pix("/Users/andrey/Documents/3.1/ООП/LIXXX/porsche.jpg");
    int w = ui->label_16->width();
    int h = ui->label_16->height();
    ui->label_16->setPixmap(pix.scaled(w,h,Qt::KeepAspectRatio));
    QPixmap pixx("/Users/andrey/Documents/3.1/ООП/LIXXX/Porsche-Logo.png");
    int ww = ui->label_17->width();
    int hh = ui->label_17->height();
    ui->label_17->setPixmap(pixx.scaled(ww,hh,Qt::KeepAspectRatio));
    // регистрация слушателя
    connect(&info, SIGNAL(notifyObservers()), this, SLOT(update()));//включаем сигнал для
наблюдателя включающийся при изменении данных
    connect(ui->btnCalc, SIGNAL(pressed()), this, SLOT(btnCalcPressed()));//включаем
сигналы включающиеся при нажатии кнопок
    connect(ui->btnUndo, SIGNAL(pressed()), this, SLOT(btnUndoPressed()));
    connect(ui->btnAddUser, &QPushButton::clicked, this, &Widget::btnUserPressed);
    connect(ui->btnIssue, &QPushButton::clicked, this, &Widget::btnIssuePressed);
    connect(ui->btnRefund, &QPushButton::clicked, this, &Widget::btnSetRefund);


    //connect(ui->moveToUsedButton,                &QPushButton::clicked,                this,
&Widget::moveToUsedTable);
    //connect(ui->moveToTableButton,               &QPushButton::clicked,                this,
&Widget::moveToTableView);
```

```
    //Widget обновляет свое состояние и затем уведомляет о этом изменении, вызывая метод update(). В ответ на этот вызов метода, наблюдатели реагируют на изменение.
}
Widget::~Widget()
{
    delete ui;
}
//public slots
void Widget::update(){
    auto value = info.getActualData();//получаем актуальную информацию
    if(value != nullptr){//если значение не пустое
        fillForm(value);//выводим на форму
    }
    //update btnUndo state
    ui->btnUndo->setEnabled(info.hasCars());
    //seting value to null
    value=nullptr;
}
//private slots


void Widget::btnCalcPressed(){                    //функции добавляют новый автомобиль
    auto value=processForm();//создаем объект класса
    showCost(value);//вычисляем стоимость и выводим ее
    controller.addToTableView(value,ui->tableView);
    info.add(value);//добавляем объект в коллекцию предыдущих запросов
    carsinfo.add(value);
    ui->btnUndo->setEnabled(info.hasCars());
    //seting value to null
    value=nullptr;

    saveTableViewDataToFullRefundFile();
}

void Widget::btnUndoPressed(){
    info.undo();//запрос на получение информации о прошлом запросе
    ui->cost->setText("0");//стоимость 0
}

void Widget::btnUserPressed(){                    //функции добавляют нового пользователя
    auto value=processClientForm();
    controller.addToClientTable(value, ui->userTable);
    //addToUserTable(value,ui->userTable);
    clientinfo.add(value);
    value=nullptr;

    saveUserTableDataToFullRefundFile();
}

void Widget::btnIssuePressed(){
    // Проверяем, выбраны ли клиент и автомобиль
```

```
    QModelIndex userIndex = ui->userTable->currentIndex();
    QModelIndex carIndex = ui->tableView->currentIndex();

    if (!userIndex.isValid() || !carIndex.isValid()) {
        QMessageBox::warning(this, "Ошибка", "Пожалуйста, выберите клиента и автомобиль перед выдачей.");
        return;
    }

    // Получаем состояние выбранного автомобиля из массива carsinfo, используя выбранный индекс строки
    int carRowIndex = carIndex.row();
    if (carRowIndex < 0 || carRowIndex >= carsinfo.array.size()) {
        QMessageBox::warning(this, "Ошибка", "Произошла ошибка при выборе автомобиля. Пожалуйста, попробуйте снова.");
        return;
    }
    bool carCondition = carsinfo.array[carRowIndex]->condition;

    if (!carCondition) {
        QMessageBox::warning(this, "Ошибка", "Выбранный автомобиль уже выдан другому клиенту.");
        return;
    }

    // Добавляем информацию о выдаче автомобиля
    rent* result = controller.addToTableRefund(ui->refundTable, ui->userTable, ui->tableView, carsinfo, ui->issueDate);

    // Проверяем, успешно ли добавлена запись
    if (!result) {
        QMessageBox::warning(this, "Ошибка", "Не удалось выдать автомобиль. Пожалуйста, проверьте введенные данные и попробуйте снова.");
        return;
    }
    // Проверяем, есть ли хотя бы одна строка в refundTable
    if (ui->refundTable->model()->rowCount() > 0) {
        // Проверяем, выбраны ли клиент и автомобиль
        if (!ui->userTable->currentIndex().isValid() || !ui->tableView->currentIndex().isValid()) {
            QMessageBox::warning(this, "Ошибка", "Пожалуйста, выберите клиента и автомобиль перед выдачей.");
            return;
        }
    }


    // Если все проверки пройдены и автомобиль успешно выдан, добавляем информацию в список аренд
    rentinfo.add(result);//добавляет информацию об аренде в таблицу возвратов (refundTable)
```

```
    saveToFullRefundData();
    saveToCurrentRefundData();

}


void Widget::btnSetRefund(){
    // Проверяем, выбраны ли клиент и автомобиль
    if (!ui->userTable->currentIndex().isValid() || !ui->tableView->currentIndex().isValid()) {
        QMessageBox::warning(this, "Ошибка", "Пожалуйста, выберите клиента и
автомобиль перед возвратом.");
        return;
    }
    //setRefundData(ui->refundTable);
    controller.setRefundData(ui->refundTable,ui->tableView,rentinfo,ui-
>refundDate,carsinfo);//вносит изменения в таблицу возвратов (refundtable), изменяя
данные о дате возврата и
                                        //обновляя статус автомобиля в таблице авто
(cartable). В конце функции обновляется модель данных в обеих таблицах.
    saveToFullRefundData();
    saveToCurrentRefundData();


}


// Path to the files
const QString fullRefundDataPath =
"/Users/andrey/Documents/3.1/ООП/LIXXX/full_refund_data.txt";
const QString currentRefundDataPath =
"/Users/andrey/Documents/3.1/ООП/LIXXX/current_refund_data.txt";

void Widget::saveToFullRefundData() {
    QFile file(fullRefundDataPath);
    if (!file.open(QIODevice::Append | QIODevice::Text))
        return;

    QTextStream out(&file);

    // Iterate through all rows in refundTable and save data to the file
    for (int row = 0; row < ui->refundTable->model()->rowCount(); ++row) {
        QStringList rowData;
        for (int col = 0; col < ui->refundTable->model()->columnCount(); ++col) {
            QString cellData = ui->refundTable->model()->data(ui->refundTable->model()-
>index(row, col)).toString();
            rowData << cellData;
        }
        out << rowData.join("\t") << "\n";  // Tab-separated values
    }

    file.close();
```

```
}

void Widget::saveToCurrentRefundData() {
    QFile file(currentRefundDataPath);
    if (!file.open(QIODevice::WriteOnly | QIODevice::Text))
        return;

    QTextStream out(&file);

    // Iterate through all rows in refundTable and save data to the file
    for (int row = 0; row < ui->refundTable->model()->rowCount(); ++row) {
        QStringList rowData;
        for (int col = 0; col < ui->refundTable->model()->columnCount(); ++col) {
            QString cellData = ui->refundTable->model()->data(ui->refundTable->model()-
>index(row, col)).toString();
            rowData << cellData;
        }
        out << rowData.join("\t") << "\n";  // Tab-separated values
    }

    file.close();
}

void Widget::saveTableViewDataToFullRefundFile() {
    QFile file("/Users/andrey/Documents/3.1/ООП/LIXXX/full_refund_data.txt");
    if (!file.open(QIODevice::Append | QIODevice::Text))
        return;

    QTextStream out(&file);

    // Save a header to identify the section of the data
    out << "=== tableView Data ===\n";

    // Iterate through all rows in tableView and save data to the file
    for (int row = 0; row < ui->tableView->model()->rowCount(); ++row) {
        QStringList rowData;
        for (int col = 0; col < ui->tableView->model()->columnCount(); ++col) {
            QString cellData = ui->tableView->model()->data(ui->tableView->model()-
>index(row, col)).toString();
            rowData << cellData;
        }
        out << rowData.join("\t") << "\n";  // Tab-separated values
    }

    file.close();
}



void Widget::saveUserTableDataToFullRefundFile() {
    QFile file("/Users/andrey/Documents/3.1/ООП/LIXXX/full_refund_data.txt");
```

```
    if (!file.open(QIODevice::Append | QIODevice::Text))
        return;

    QTextStream out(&file);

    // Save a header to identify the section of the data
    out << "=== userTable Data ===\n";

    // Iterate through all rows in userTable and save data to the file
    for (int row = 0; row < ui->userTable->model()->rowCount(); ++row) {
        QStringList rowData;
        for (int col = 0; col < ui->userTable->model()->columnCount(); ++col) {
            QString cellData = ui->userTable->model()->data(ui->userTable->model()-
>index(row, col)).toString();
            rowData << cellData;
        }
        out << rowData.join("\t") << "\n";  // Tab-separated values
    }

    file.close();
}


//private
car *Widget::processForm(){//берем данные с формы и создаем новый объект класса
    int age = ui->Age->text().toInt();
    int ID = ui->Number->text().toInt();
    QString Colour = ui->Colour->text();
    car::CarType type = static_cast<car::CarType>(ui->CarType->currentIndex());
    QString Brand = ui->Brand->text();
    return new car(age, ID, Colour, type, Brand);
}
client *Widget::processClientForm(){
    int Passport = ui->passport->text().toInt();
    int PhoneNumber = ui->phonenumber->text().toInt();
    QString FIOclient = ui->FIOclient->text();
    QString EMail = ui->EMail->text();
    return new client (Passport,PhoneNumber,FIOclient,EMail);
}
void Widget::fillForm(car *value){//заполняем форму актуальной информацией
    QString str=value->getBrand();
    ui->Brand->setText(str);

    str=QString::number(value->getAge());
    ui->Age->setText(str);

    if (value->getType() == car::CarType::STANDARD) {
    ui->CarType->setCurrentIndex(0);
    } else if (value->getType() == car::CarType::COMFORT) {
    ui->CarType->setCurrentIndex(1);
    } else if (value->getType() == car::CarType::LUXURY) {
```

```
    ui->CarType->setCurrentIndex(2);
    } else if (value->getType() == car::CarType::ELECTRIC) {
    ui->CarType->setCurrentIndex(3);
    }
    str=value->getColour();
    ui->Colour->setText(str);
    str=QString::number(value->getNumber());
    ui->Number->setText(str);
}
QString Widget::showCost(car *value){
    CalculationFacade cur;//создаем объект фасада вычисления
    int rating=cur.getCost(value);//получаем стоимость от фасада
    QString str=QString::number(rating);//переводим тип данных стоимости из str в qstring
    ui->cost->setText(str);//выводим стоимость на форму
    return str;
}
```

Выводы:

В ходе работы было разработано приложение для управления прокатом автомобилей с использованием библиотеки Qt, позволяющее вести учет клиентов, автомобилей и операций аренды. Программа успешно продемонстрировала основные возможности графического интерфейса, включая работу с таблицами, кнопками и визуальными элементами.