ГУАП

КАФЕДРА № 43

ОТЧЕТ
ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

ассистент		И.М. Лозоватский
должность, уч. степень, звание	подпись, дата	инициалы, фамилия

ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ №8

Создание двумерного пользовательского интерфейса в среде Qt Creator с использованием расширенных виджетов Qt

по дисциплине: Проектирование человеко-машинного интерфейса

РАБОТУ ВЫПОЛНИЛ

СТУДЕНТ ГР.	4134к		Столяров Н.С.
		подпись, дата	инициалы, фамилия

Санкт-Петербург
2024

Цель работы:
Освоение использования расширенных виджетов Qt для создания пользовательского интерфейса

Задание:
Разработать и отладить приложение на языке C++ с графическим пользовательским интерфейсом, использующим расширенные 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 приложения представляет собой многостраничное окно, состоящее из следующих элементов:

1. **Главное окно**:

• Содержит **QStackedWidget**, который управляет переключением между различными страницами приложения.

• Кнопки навигации, такие как pushButtonAddCar, pushButtonAddClient, и pushButtonOperations, позволяют пользователю переходить на страницы для управления автомобилями, клиентами и операциями соответственно.

2. **Страница управления автомобилями (Page 1)**:

- Использует **QFormLayout** для ввода данных об автомобилях, включая такие элементы, как QLineEdit для ввода текста и QComboBox для выбора из предустановленных значений.
- **QPushButton** для добавления автомобилей и кнопки для возврата на главную страницу.
- **QTableView** для отображения списка автомобилей.

3. **Страница управления клиентами (Page 2)**:

- Использует **QFormLayout** для ввода данных о клиентах.
- **QPushButton** для добавления клиентов и кнопки для возврата на главную страницу.
- **QTableView** для отображения списка клиентов.

4. **Страница операций (Page 3)**:

• Использует **QFormLayout** для ввода информации о выдаче и возврате автомобилей.

• Кнопки для выполнения операций и возврата на главную страницу. • **QTableView** для отображения истории операций.

В интерфейсе используются кнопки навигации для удобного перехода между страницами и управления данными. Виджеты организованы в **QFormLayout** для структурированного ввода информации и **QTableView** для отображения данных в виде таблиц.

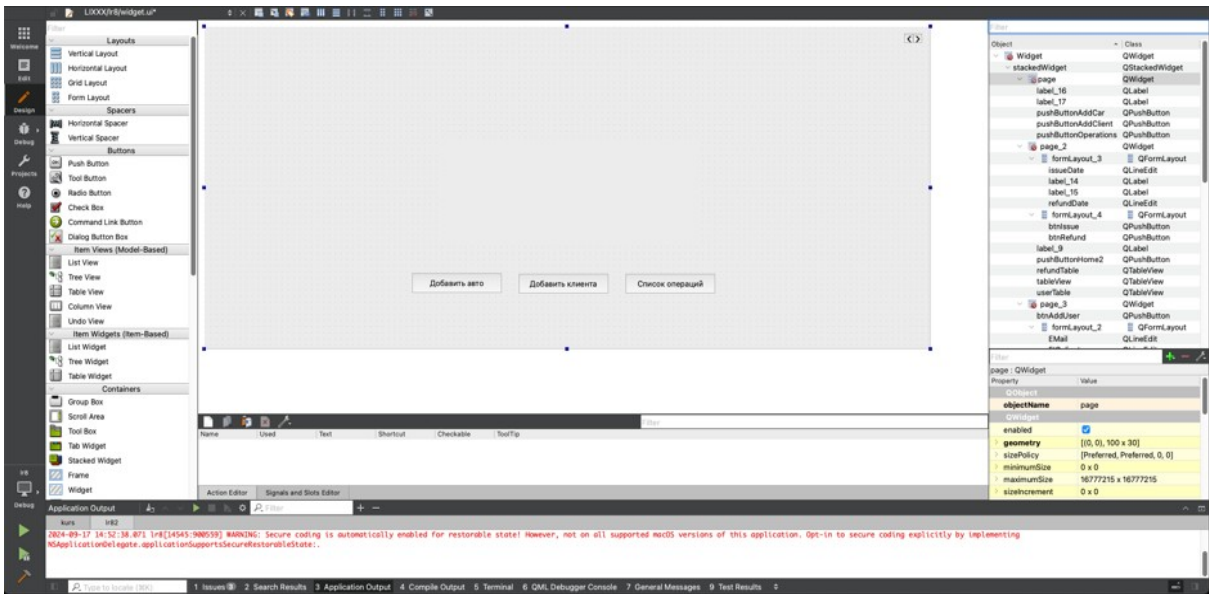

Рисунок 1 – Главный экран, редактор

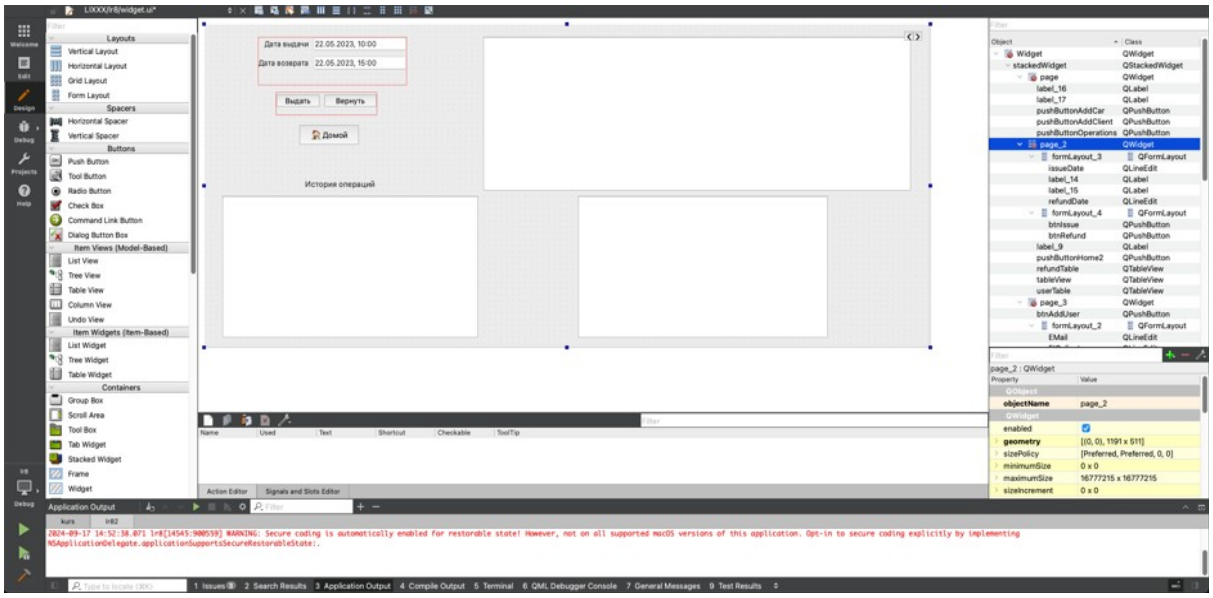

Рисунок 2 – страница с операциями, редактор

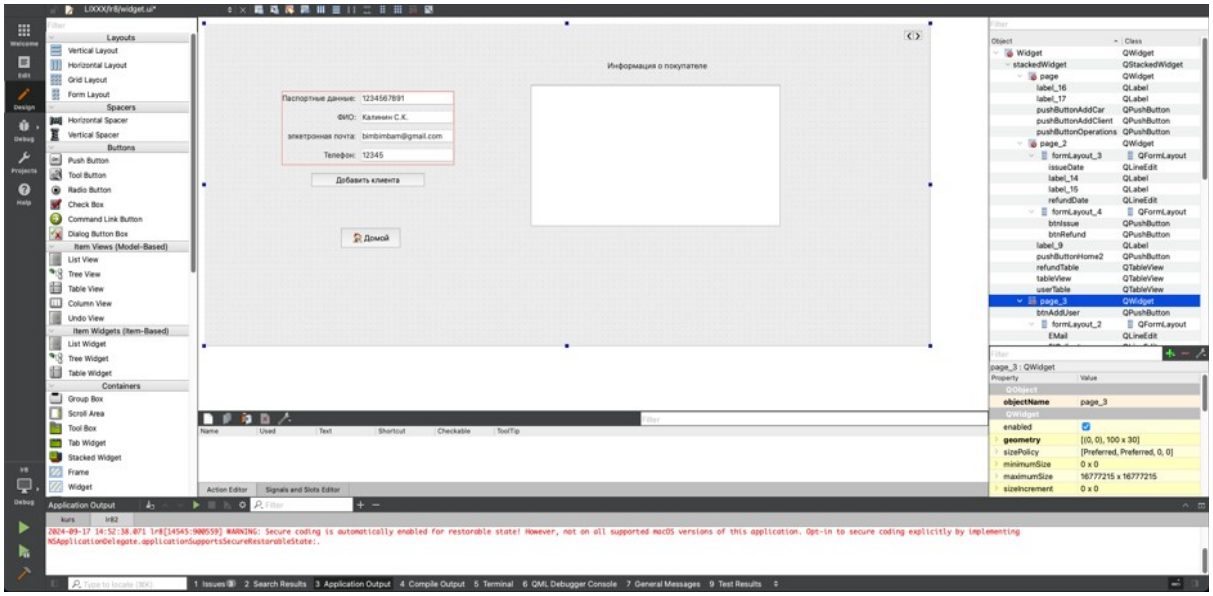

Рисунок 3 – страница с покупателями, редактор

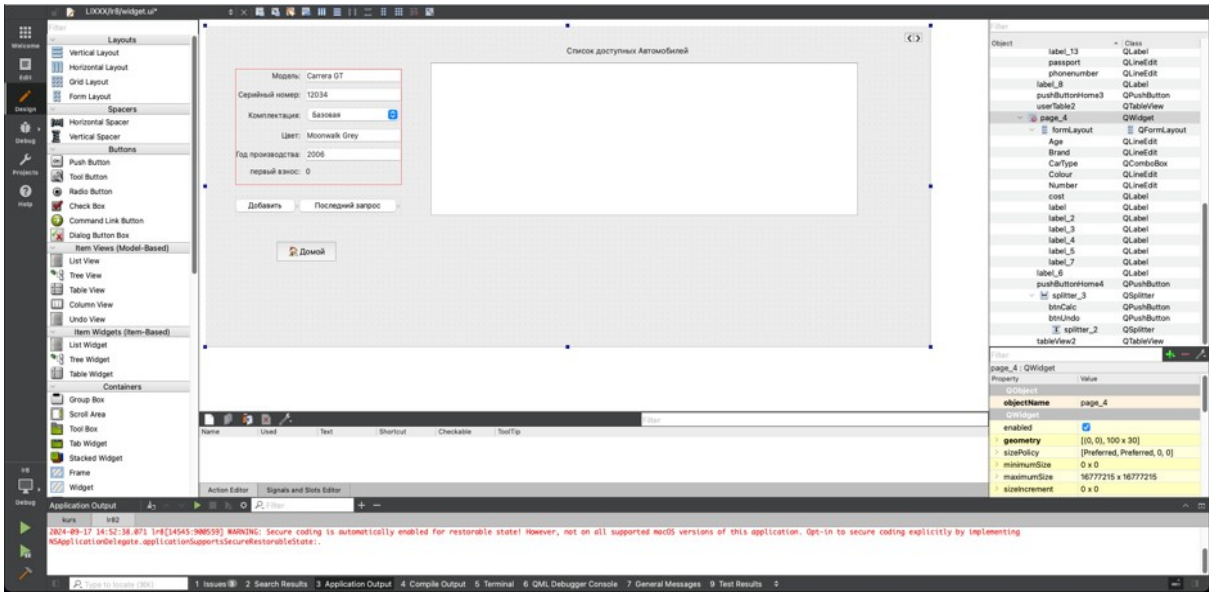

Рисунок 4 – страница с автомобилями, редактор

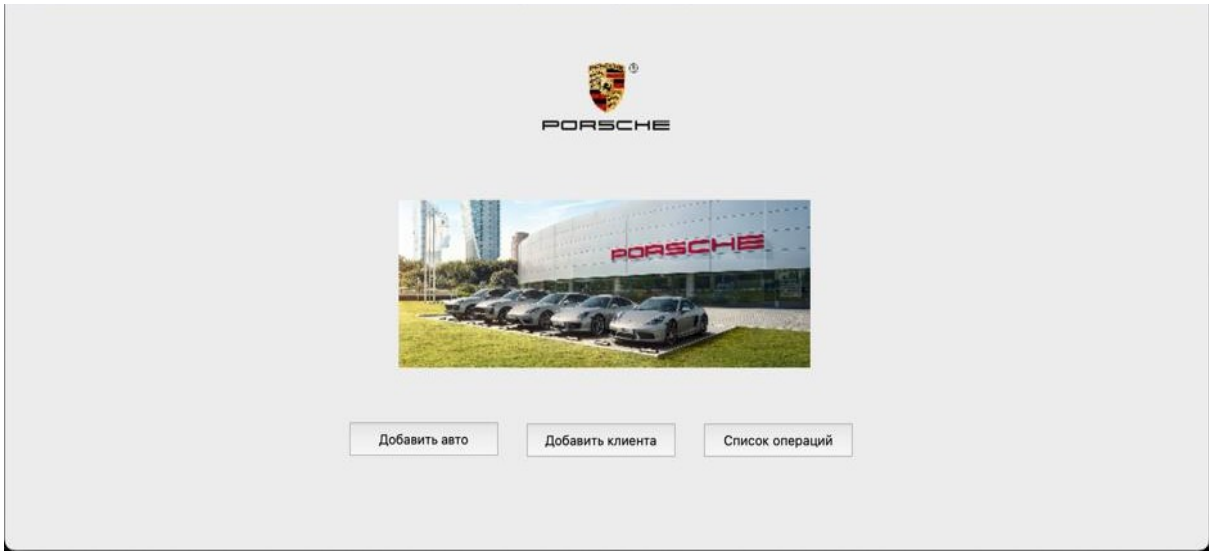

Рисунок 5 – главный экран

Рисунок 6 – страница с автомобилями

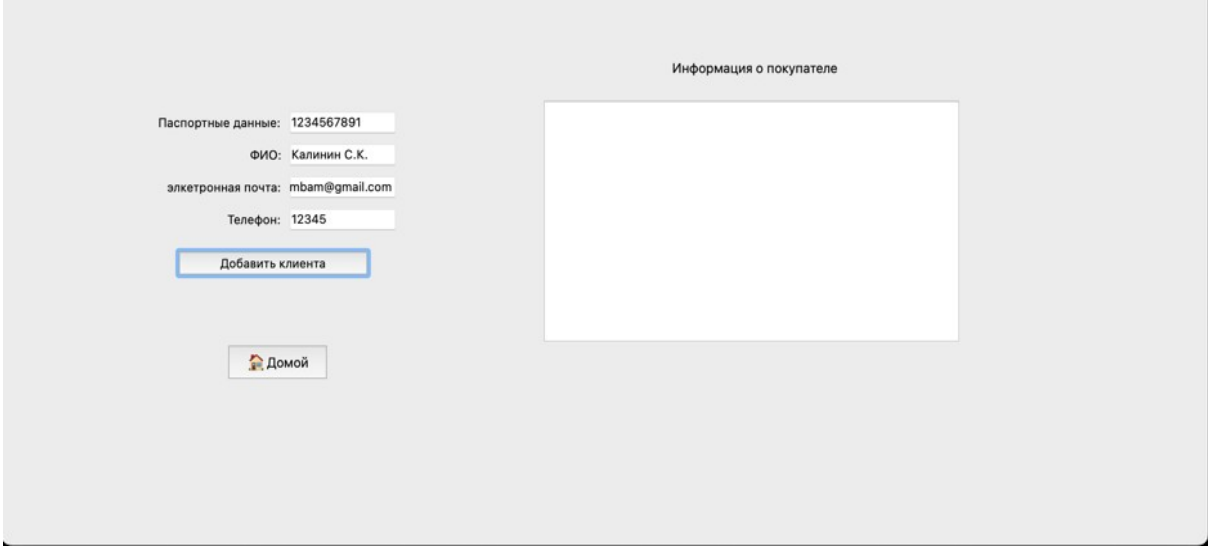

Рисунок 7 – страница с клиентами

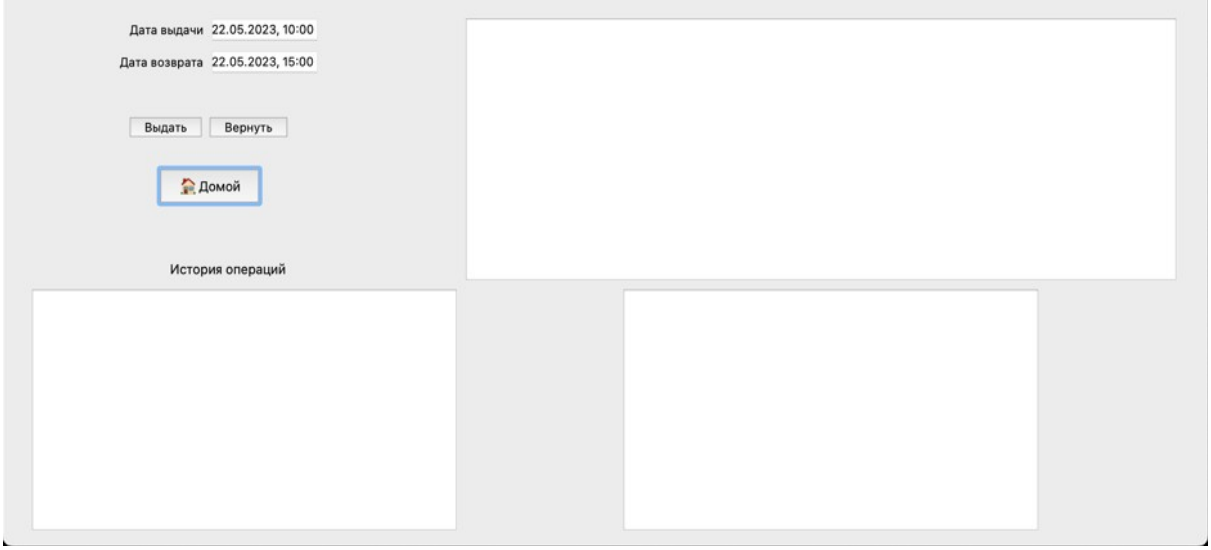

Рисунок 8 – страница с операциями

В данной работе были использованы следующие элементы интерфейса: Расширенные виджеты:

1. **QStackedWidget**

• **Использование**: Управление несколькими страницами внутри основного окна. Позволяет переключаться между разными виджетами (страницами) в одном месте.

2. **QTableView**

• **Использование**: Отображение данных в виде таблицы. В приложении используется для отображения таблиц с информацией о клиентах, автомобилях и операциях.

3. **QPushButton**

• **Использование**: Кнопки для выполнения действий, таких как добавление автомобилей, клиентов, обработка операций. Позволяют взаимодействовать с пользователем и запускать функции приложения.

Остальные виджеты:

1. **QWidget**

• **Использование**: Базовый виджет, который используется для создания других виджетов и окон.

2. **QFormLayout**

• **Использование**: Расположение виджетов в виде формы. Используется для организации ввода данных, таких как информация о клиентах и автомобилях.

3. **QLineEdit**

• **Использование**: Поля для ввода текста, например, для ввода информации о клиентах и автомобилях.

4. **QLabel**

• **Использование**: Метки для отображения текста и информации, такой как названия полей ввода.

5. **QComboBox**

• **Использование**: Выпадающий список для выбора из предустановленных значений, например, тип автомобиля.

6. **QTableView**

• **Использование**: Отображение данных в виде таблиц, например, информация о клиентах, автомобилях и истории операций.

7. **QPushButton**

• **Использование**: Кнопки для выполнения различных действий, таких как добавление записей и обработка операций.

8. **QSplitter**

• **Использование**: Разделение области окна на несколько секций, позволяя пользователю изменять их размер.

Исходный текст приложения, основной код для интерфейса:

View_controller:

```
#include <QMessageBox>
#include "view_controller.h"
#include <QMap>
View_Controller::View_Controller(QObject *parent)
   : QObject{parent}
{
}

QMap<QString, double> carRentalCosts;
```

```
void View_Controller::addToTableView(car* lastObject, QTableView* tableView, QTableView* tableView2)//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); tableView2->setModel(model);
    model->setHorizontalHeaderLabels({"Название","Комплектация","Номер","Цвет","Год выпуска","Ставка","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)));

   double cost = cur.getCost(lastObject);
    carRentalCosts[NumberItem->text()] = cost; // Сохраняем стоимость, используя номер автомобиля в качестве ключа
```

```
   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();
   tableView2->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
    QString personNumber = selectedPersonIndexes.at(0).data().toString();



    QString carNumber = selectedCarIndexes.at(2).data().toString();


              if  (carRentalCosts.contains(carNumber))  {              double  totalCost  =
carRentalCosts[carNumber]; // Получаем сохраненную стоимость      QStandardItem*
finalCost = new QStandardItem(QString::number(totalCost)); // Создаем элемент для
отображения стоимости

    // Добавляем остальные элементы строки
     QList<QStandardItem*> newRow;      // ...
[добавление остальных элементов] ...

    newRow.append(finalCost); // Добавляем элемент с итоговой стоимостью в строку

}



    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);
   //QStandardItem* finalCost = new QStandardItem(carNumber);  //********

   // Добавляем созданные элементы в новую строку
newRow.append(personNumberItem);
newRow.append(carNumberItem);
```

```
newRow.append(issueDateItem);
newRow.append(returnDateItem);
//newRow.append(finalCost); //********


        // 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
   // 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->stackedWidget->setCurrentIndex(0);
   connect(ui->pushButtonHome2,                SIGNAL(clicked()),                this,
SLOT(on_pushButtonHome2_clicked()));
   connect(ui->pushButtonHome3,                SIGNAL(clicked()),                this,
SLOT(on_pushButtonHome3_clicked()));
   connect(ui->pushButtonHome4,                SIGNAL(clicked()),                this,
SLOT(on_pushButtonHome4_clicked()));    ui-
>btnUndo->setEnabled(false);
   QPixmap
pix("/Users/andrey/Documents/SUAI/3.1/ООП/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/SUAI/3.1/ООП/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;
}


void Widget::on_pushButtonHome2_clicked() { ui->stackedWidget->setCurrentIndex(0); // Возвращение на главную страницу }

void Widget::on_pushButtonHome3_clicked() {
        ui->stackedWidget->setCurrentIndex(0); // Возвращение на главную страницу }

void Widget::on_pushButtonHome4_clicked() { ui->stackedWidget->setCurrentIndex(0); // Возвращение на главную страницу }

void Widget::on_pushButtonAddCar_clicked() {
        ui->stackedWidget->setCurrentIndex(3); // Перенаправление на page_4 }

void Widget::on_pushButtonAddClient_clicked() {    ui->stackedWidget->setCurrentIndex(2); // Перенаправление на page_3 }

void Widget::on_pushButtonOperations_clicked() {
   ui->stackedWidget->setCurrentIndex(1); // Перенаправление на page_2
}

//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, ui->tableView2);
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);
controller.addToClientTable(value, ui->userTable2); //*****
   //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 результате выполнения работы было создано приложение для управления автопарком и базой клиентов с использованием расширенных виджетов Qt. Интерфейс приложения включает многостраничное окно с функциями для добавления автомобилей, клиентов и управления операциями, что обеспечивает удобный и структурированный доступ к различным данным.