СОДЕРЖАНИЕ

[**ВВЕДЕНИЕ 5**](#_Toc9786374)

[**1. ОПИСАНИЕ ПРОГРАММЫ ДЛЯ ПРОГРАММИСТА 6**](#_Toc9786375)

[**1.1. Постановка задачи 6**](#_Toc9786376)

[**1.2. Разработка спецификаций требований 6**](#_Toc9786377)

[**1.3. Разработка архитектуры программного средства 6**](#_Toc9786378)

[**1.4. Классы программы 8**](#_Toc9786379)

[**1.5. Разработка собственной системы базы данных 9**](#_Toc9786380)

[**2. ОПИСАНИЕ АЛГОРИТМОВ РЕШЕНИЯ ЗАДАЧИ 11**](#_Toc9786381)

[**3. ОПИСАНИЕ ПРОГРАММЫ 13**](#_Toc9786382)

[**4. ТЕСТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА 18**](#_Toc9786383)

[**ЗАКЛЮЧЕНИЕ 19**](#_Toc9786384)

[**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 20**](#_Toc9786385)

[**ПРИЛОЖЕНИЕ А. Листинг программы 21**](#_Toc9786386)

[**ПРИЛОЖЕНИЕ Б. Скриншот работы программы 30**](#_Toc9786387)

**ВВЕДЕНИЕ**

В настоящее время автоматизация производства проникает всё глубже и глубже во все сферы услуг, не является исключением и сфера ремонта компьютерной техники.

Последнее время, все больший интерес к автоматизации ощущается со стороны активно развивающихся сервисов ремонта компьютерной техники, желающих минимизировать потери рабочего времени персонала, сэкономить на его численности и поднять уровень сервиса в заведениях.

АРМ «Учета заказов ремонта компьютерной техники» - это большой шаг на пути к более качественной, быстрой работе, приносящей прибыль. Конечно, программы автоматизации актуальны практически для любых организаций, при этом, для многих принципы их работы схожи. Однако автоматизация работы ремонта компьютерной техники имеет свои особенности, которые значительно отличают её от остальных.

Сервис ремонта компьютеров работают с компьютерами, а значит, качество обслуживания стоит на одном из первых мест. Сюда входит не только опытный персонал и качественные запчасти, но и скорость в том числе. Быстрое оперативное обслуживание - это всегда большой плюс, а обеспечить его поможет автоматизация работы по учету заказов ремонта компьютерной техники.

Быстрое управление заказами, оперативный расчет с клиентами - все это позволяет поднять обслуживание на новый уровень. Автоматизация работы  сервиса ускорит работу, а значит, количество обслуживаемых клиентов увеличится, поднимая прибыль.

Кроме того, не стоит забывать и про удобство управления. Руководящему составу станет гораздо проще принимать решения и грамотно организовывать работу. Таким образом, АРМ по учету заказов ремонта компьютерной техники становится полезным инструментом, с помощью которого работа становится проще и лучше.

1. **ОПИСАНИЕ ПРОГРАММЫ ДЛЯ ПРОГРАММИСТА**
   1. **Постановка задачи**

Целью данного проекта является разработка автоматизированного рабочего места по учёту заказов ремонта компьютерной техники.

Для достижения поставленной цели необходимо реализовать следующие задачи:

* организация поиска данных по списку оформленных заказов.
* организовать разработку собственной системы базы данных
* организация добавления и удаления данных;
* организация просмотра всей информации находящейся в базе данных;
* организация печати гарантийного чека на оказанную услугу.

Таким образом, реализовав описанные выше задачи, будет получено полнофункциональное приложение, позволяющее вести базу данных.

* 1. **Разработка спецификаций требований**

Для база данных будет разрабатываться библиотека для работы с файлами. Клиентский интерфейс прикладной программы будет разрабатываться в среде разработке программного обеспечения Visual Studia C++. Взаимодействие интерфейсной части системы с базой данных должно осуществляться на основе технологии ADO. Взаимодействие с пользователем будет осуществляться через меню и диалоговые окна. Результаты работы программы будут выводиться в виде документа.[1]

* 1. **Разработка архитектуры программного средства**

Программное средство представляет совокупность 6 программных модулей.[2] Каждый модуль отвечает за хранение информации или выполнению функций над данными.

Программные модули представлены на (рисунке 1.1).

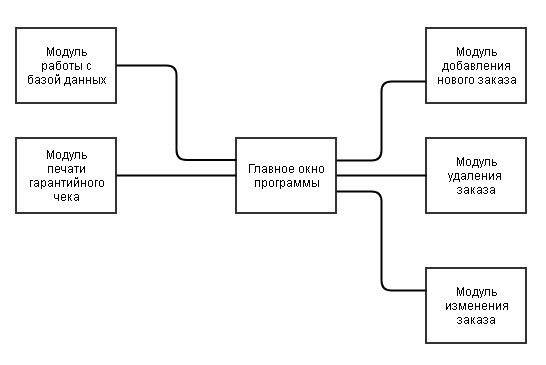


Рисунок 1.1 Программные модули

Модуль главного окна описывает главное окно программы, интерфейс пользователя. Данный модуль предоставляет доступ к другим модулям программы.

Модуль работы с базой данных позволяет создавать новую базу данных с нуля. А именно создание и удаление базы данных и таблиц.

Модуль добавления нового заказа позволяет работнику оформить заказ и ввести начальные данные заказа.

Модуль удаления заказа позволяет пользователю удалить новый заказ в случае его не пригодности.

Модуль изменения заказа позволяет пользователю изменять данные в ходе работы над заказом.

Модуль печати гарантийного чека позволяет пользователю распечатать гарантийной чек с информацией о заказе.

* 1. **Классы программы**

Программа, АРМ «Учета заказов ремонта компьютерной техники», будет иметь следующие классы: [3]

* User содержащий информацию о пользователе;
* Order содержащий информацию о заказе.

Также для организации хранения данных была разработана собственная система баз данных, которая имеет следующие классы:[4]

* TxtDataBaseLib содержащий вспомогательные функции для создания базы данных;[5]
* CommandsTableTxt содержащий вспомогательные функции для работы с таблицами базы данных.[6]

Программа, моделирующая работу АРМ «Учета заказов ремонта компьютерной техники», будет иметь следующие классы (рисунок 1.2).

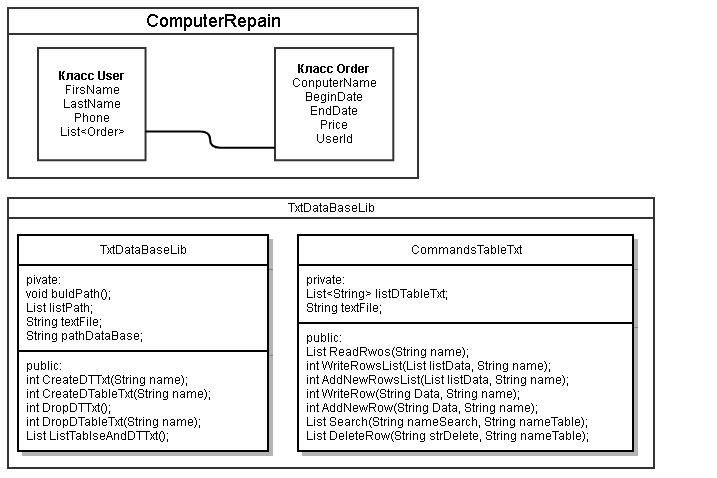


Рисунок 1.2 «Классы проекта»

Таким образом, можно увидеть, что были разработаны два проекта. Один для взаимодействия с пользователем, второй для работы с базой.

* 1. **Разработка собственной системы базы данных**

Хранение данных должно производиться в текстовые файлы. Для создания собственной системы базы данных, были поставлены задачи:

* система должна создавать базу данных;
* система должна создавать таблицы базы данных;
* система должна выдавать и сохранять идентификационный номер строки каждой из таблиц;
* система должна предусматривать создание, удаление и редактирование срок в таблице;
* система должна предусматривать поиск в базе данных.

Исходя из выше перечисленных пунктов, был создан, новы проект для реализации собственной системы базы данных. В котором имеется два класса, каждый из которых отвечает за собственный раздел системы.

Класс TxtDataBaseLib предназначен для создания, удаления базы данных, а также таблиц в ней. Предусматривает методы:

* метод CreateDTTxt принимает строку имени для создания базы данных и при успешном выполнении возвращает ноль, противном случае минус один;
* метод DropDTTxt удаляет базу данных вместе с таблицами и при успешном выполнении возвращает ноль, противном случае минус один;
* метод CreateDTableTxt принимает строку имени таблицы, создает новую таблицу в базе данных и при успешном выполнении возвращает ноль, противном случае минус один;
* метод DropDTableTxt принимает строку имени таблицы, удаляет таблицу из базы данных и при успешном выполнении возвращает ноль, противном случае минус один;
* метод ListTablseAndDTTxt возвращает лист таблиц и название базы данных.

Данный класс наглядно показывает работу с базой данных и таблицами на уровне создания и удаления.

Класс CommandsTableTxt реализует методы для работы со строками и предусматривает такие методы как:

* метод ReadRows принимает строку имени таблицы и возвращает лист трок из таблицы;
* метод WriteRowsList принимает строку имени таблицы, лист строк для записи в базу данных и при успешном выполнении возвращает ноль, противном случае минус один;
* метод WriteRow принимает строку имени таблицы, строку данных для записи в базу данных и при успешном выполнении возвращает ноль, противном случае минус один;
* метод AddNewRows принимает строку имени таблицы, лист данных для перезаписи всей таблицы и при успешном выполнении возвращает ноль, противном случае минус один;
* метод AddNewRow принимает строку имени таблицы, строку данных для перезаписи таблицы и при успешном выполнении возвращает ноль, противном случае минус один;
* метод Search принимает строку имени таблицы, строку дынных для поиска в таблице и при успешном выполнении возвращает лист строк;
* метод DeleteRow принимает строку имени таблицы, строку данных для удаления из таблицы и при успешном выполнении возвращает лист строк новой базы данных.

Методы, определенные в классе CommandsTableTxt в разы облегчают работу с системой базы данных для разработчика.[7] Помогая сэкономить время при разработке нового интерфейса для пользователя.

1. **ОПИСАНИЕ АЛГОРИТМОВ РЕШЕНИЯ ЗАДАЧИ**

Данный программный продукты имеет ряд функций, которые решают проблемы пользователя. При разработке программы, моделирующей работу АРМ «Учета заказов ремонта компьютерной техники», наиболее важными для создания и понимания являются, например, схемы программы методов (блок-схемы): алгоритм «Создание заказа» (рисунок 2.1) и «Печать гарантийного чека» (рисунок 2.2).[8]

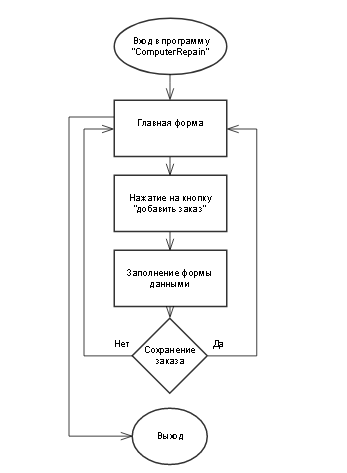


Рисунок 2.1 «Создание заказа»

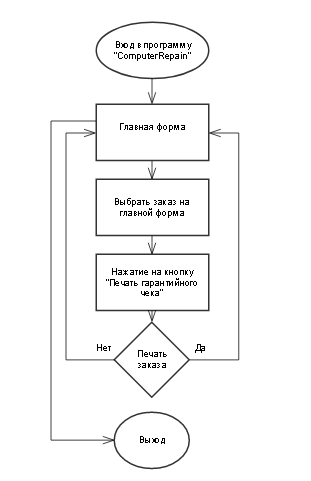


Рисунок 2.2 «Печать гарантийного чека»

Видно, что алгоритмы просты в понимании и легко решают поставленные задачи.

1. **ОПИСАНИЕ ПРОГРАММЫ**

После загрузки приложения появится главное окно программы, в котором есть таблица с заказами (рисунок 3.1):

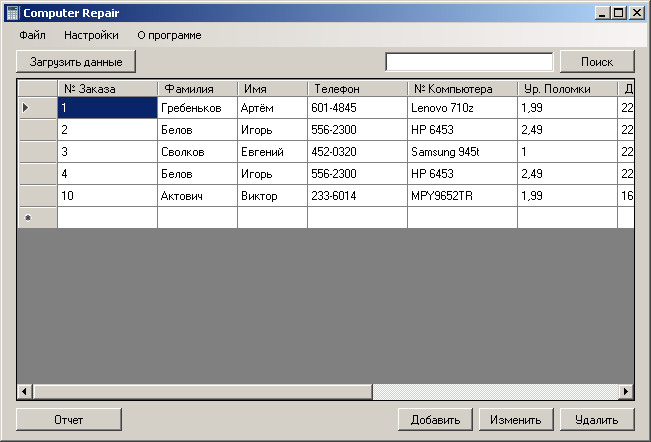


Рисунок 3.1 «Главное окно программы»

При первой загрузки для работы с программной необходимо создать базу данных. Для это в программе на панели меню есть кнопка «Настройки» (рисунок 3.2).

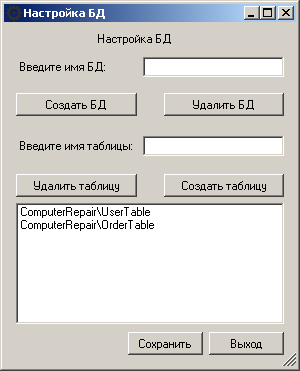


Рисунок 3.2 «Форма настройки базы данных»

Создаем базу данных и таблицы в ней. После нажатие кнопки «Создать таблицу» в лист боксе появятся название созданной таблицы (рисунок 3.3). После создание базы данных закрываем окно, и переходи на главную форму.

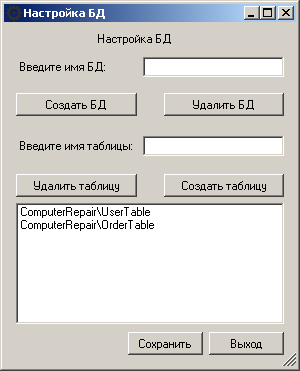


Рисунок 4.3 «Создание таблиц»

Для добавления нового заказа нужно перейти на форму заказа нажатием кнопки «Добавить» (рисунок 3.4).

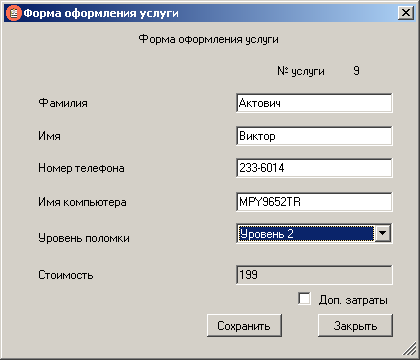


Рисунок 3.4 «Форма добавления заказа»

Заполнив все поля, нажимаем кнопку «Добавить». Нас возвращает обратно на главную форму и в таблице появляется новый заказ (рисунок 3.5).

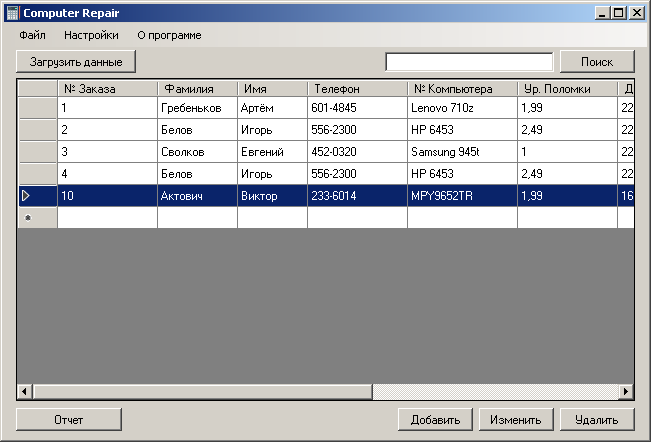


Рисунок 3.5 «Форма с добавленным заказом»

Если заказ был отменен или была допущена ошибка заказ можно удалить. Выделив его в таблице и нажав кнопку «Удалить».

При готовности заказа нужно перейти на форму изменения заказа, для подтверждения итоговой суммы (рисунок 3.6).

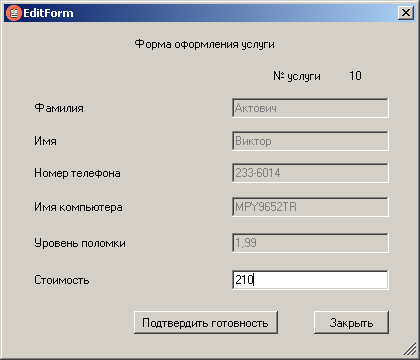


Рисунок 3.6 «Форма изменения заказа»

После корректировки цены нажимаем кнопку «Подтвердить готовность», автоматический ставится дата окончания заказа (рисунок 3.7).

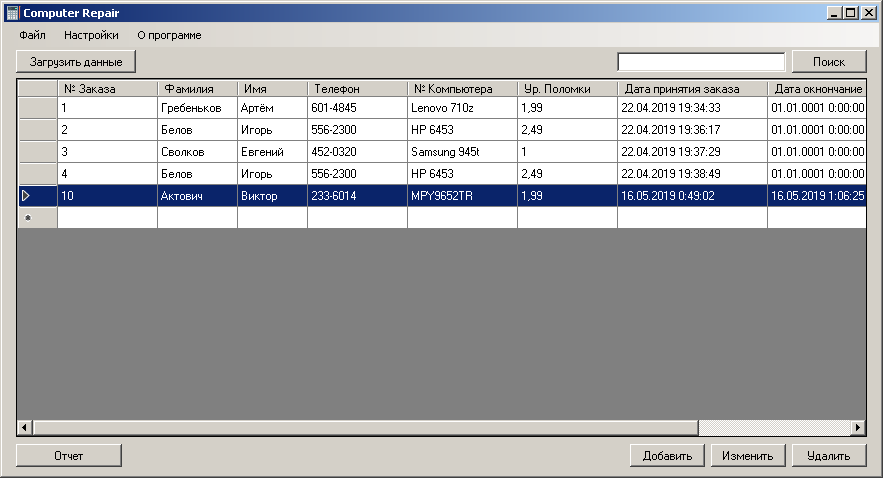


Рисунок 3.7 «Изменение данных на главной форме»

При необходимости найти нужный заказ по информации из таблицы. Можно прибегнуть к поиску по таблице (рисунок 3.8).

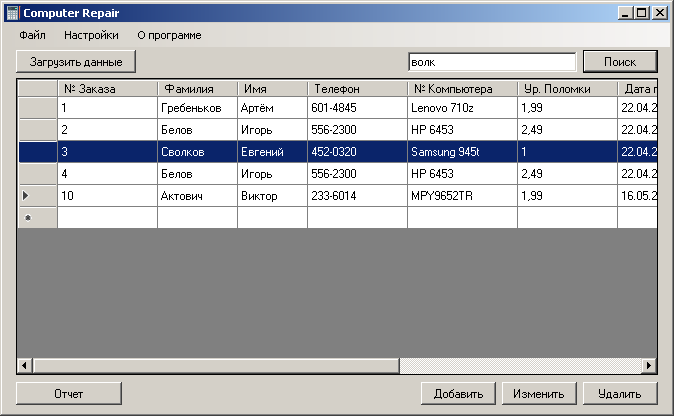


Рисунок 3.8 «Поиск по таблице»

При передаче отремонтированной техники владельцу. Печатается гарантийный чек. Выбираем нужный заказ из таблицы и нажимаем кнопку отчет. Появится окно предварительной печати (рисунок 3.9).

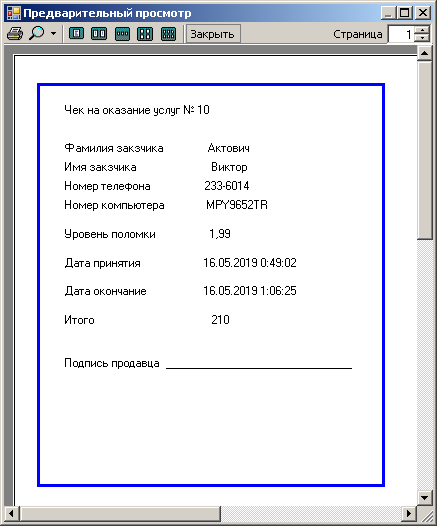


Рисунок 3.9 «Просмотр предварительной печати»

После завершения работы взываем вкладку меню «Файл» и нажимаем кнопку выход (рисунок 3.10).

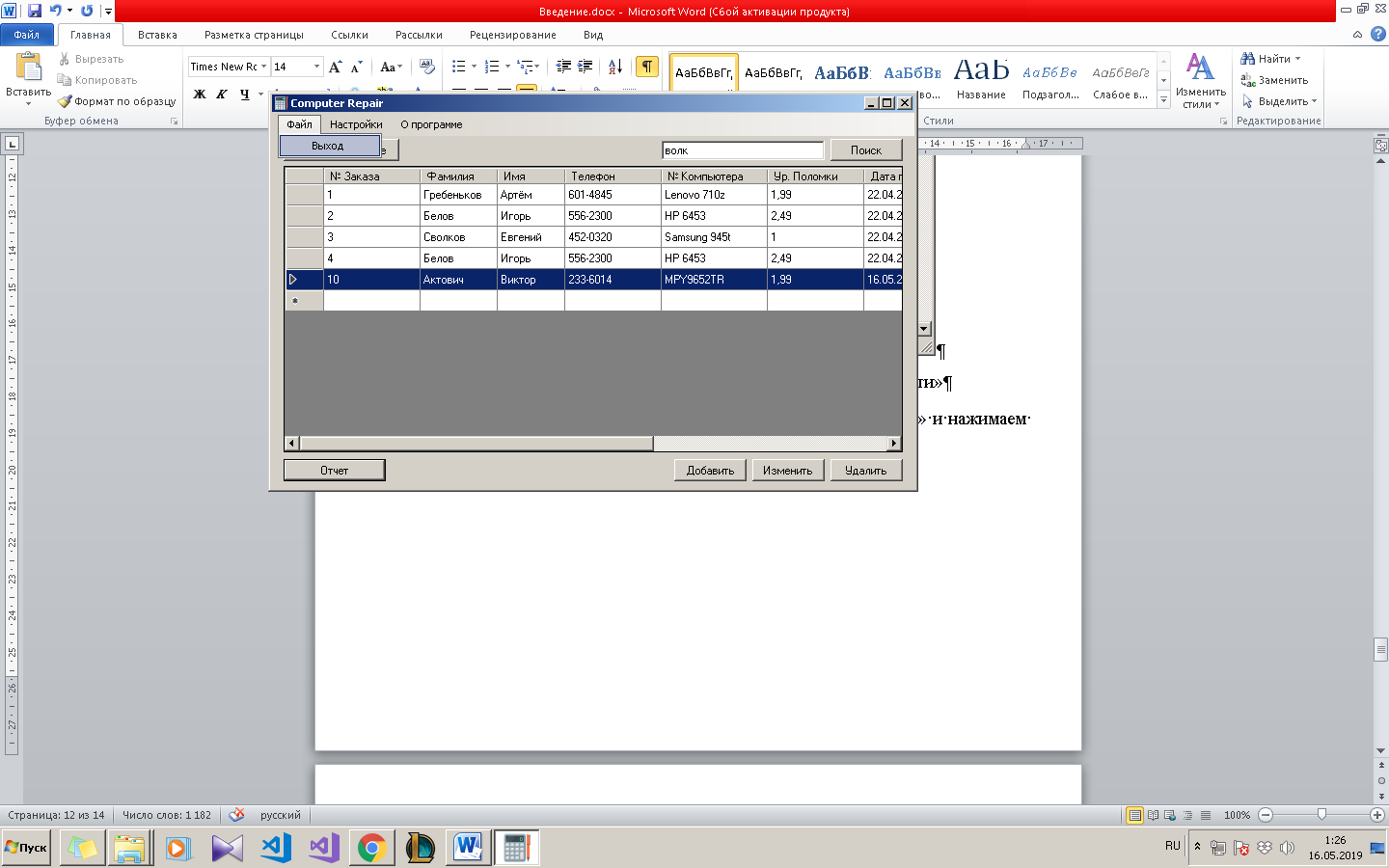


Рисунок 3.10 «Выход из программы»

1. **ТЕСТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА**

Тестирование программного продукта проводилось по всем вариантам использования. Результаты тестирования представлены в таблице 4.1.

Таблица 4.1 – Результаты тестирования программного продукта

|  |  |  |
| --- | --- | --- |
| Название теста | Последовательность действий | Результат |
| Запуск | Запуск файла ComputerRepair.exe | Запуск программы |
| Проверка работоспособности кнопок | Перемещение по формам и нажатие на кнопки | Все кнопки функционируют |
| Проверка работоспособности поиска | Ввод информации для поиска | Отображается необходимая информация |
| Проверка добавления данных | Заполнение полей | Запись успешно добавлена |
| Проверка удаления данных | Выбор данных | Запись успешно удалена |

В данной таблица показывает, что программы работоспособна и готова к использованию в реальных условиях.

# **ЗАКЛЮЧЕНИЕ**

В данном проекте разработано программное средство АРМ «Учета заказов ремонта компьютерной техники», в котором были реализованы следующие задачи:

* организация поиска данных по списку оформленных заказов.
* организовать разработку собственной системы базы данных
* организация добавления и удаления данных;
* организация просмотра всей информации находящейся в базе данных;
* организация печати гарантийного чека на оказанную услугу.

Для разработки программы была выбрана среда программирования Visual Studio 2019, так как она обладает высокопроизводительным компилятором, а также библиотеками визуальных компонентов, которые обеспечивают создание простого и интуитивно понятного интерфейса.

Интерфейс программы довольно прост даже для пользователя, который в обыденной жизни не часто пользуется программными продуктами такого рода.

Программу можно использовать в различных ремонтных сервисных центрах.

Достоинства программы:

* лёгкое освоение;
* легкая модификация;
* удобный интерфейс;
* работа в ОС Windows.
* Недостатки:
* неизменность смены интерфейса программного средства.

Созданное программное средство работоспособно, работает корректно, без сбоев. Работа в программе довольно проста и рассчитана на любого пользователя.

# **СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

[1] Язык С++: курс объектно-ориентированного программирования / У. Савитч. - М.: Вильямс, 2001.

[2] Технология разработки программ / С.А. Орлов. - СПб.: Питер, 2004.

[3] Технология программирования / Г.С. Иванова. - М.: МГТУ им. М.Э. Баумана, 2002.

[4] Язык программирования / П. Стивен - М. Издательство «Диасофт», 2002.

[5] Как программировать на С++ / П. Дейтел, X. Дейтел. - М.: Издательство «Бином» 2001.

[6] Язык СИ++: Уч. Пособие. Финансы и статистика / В.В. Подбельский - М.: 2001.

[7] Как программировать на С++: Пер. с англ. / Х. Дейтел, П. Дейтел - М.: Бином, 2000.

[8] Язык программирования С++. Лекции и упражнения. Учебник: Пер. с англ. / С. Прата - СПб.: ДиаСофтЮП, 2003.

# **ПРИЛОЖЕНИЕ А**

**(Обязательное)**

**Листинг программы**

MainForm.h

private: List<String^>^ tableName;

public: void Set(int IdO, int IdU)

{

IdOrder = IdO;

IdUser = IdU;

IdHelper::SaveAllId(IdUser, IdOrder);

}

public: void Set(int IdO)

{

IdOrder = IdO;

IdHelper::SaveAllId(IdUser, IdOrder);

}

private: System::Void MainForm\_Load(System::Object^ sender, System::EventArgs^ e) {

try

{

IdHelper^ IdHel = gcnew IdHelper();

IdUser = IdHelper::GetUserId();

IdOrder = IdHelper::GetOrderId();

}

catch (System::Exception^ e)

{

}

}

public: static DataGridViewRow^ rowSel;

public: void LoadDataGrid() {

tableName = TxtDataBaseLib::JobTxtDB::ListTablseAndDTTxt();

List<User^>^ UserList = IdHelper::GetUsersList(tableName[1]);

List<Order^>^ OrderList = IdHelper::GetOrdersList(tableName[2]);

User^ user = gcnew User();

int i = 0;

dataGridView1->Rows->Clear();

for each (Order^ order in OrderList)

{

dataGridView1->Rows->Add();

dataGridView1->Rows[i]->Cells[0]->Value = order->Id;

dataGridView1->Rows[i]->Cells[4]->Value = order->NameComputer;

dataGridView1->Rows[i]->Cells[5]->Value = order->Lvl;

dataGridView1->Rows[i]->Cells[6]->Value = order->BeginDate;

dataGridView1->Rows[i]->Cells[7]->Value = order->EndDate;

dataGridView1->Rows[i]->Cells[8]->Value = order->Cost;

for each (User^ us in UserList)

{

if (us->Id == order->UserId) {

dataGridView1->Rows[i]->Cells[1]->Value = us->LastName;

dataGridView1->Rows[i]->Cells[2]->Value = us->FirstName;

dataGridView1->Rows[i]->Cells[3]->Value = us->Phone;

break;

}

}

i++;

}

}

private: System::Void exitToolStripMenuItem\_Click(System::Object^ sender, System::EventArgs^ e) {

Application::Exit();

}

private: System::Void menuItem\_Setting\_Click(System::Object ^ sender, System::EventArgs ^ e);

private: System::Void menuItem\_About\_Click(System::Object^ sender, System::EventArgs^ e);

private: System::Void button5\_Click(System::Object^ sender, System::EventArgs^ e);

private: System::Void button4\_Click(System::Object^ sender, System::EventArgs^ e);

private: System::Void button6\_Click(System::Object^ sender, System::EventArgs^ e) {

LoadDataGrid();

}

private: System::Void button2\_Click(System::Object^ sender, System::EventArgs^ e) {

String^ orderString;

orderString = String::Concat(dataGridView1->CurrentRow->Cells[0]->Value, "|",

dataGridView1->CurrentRow->Cells[4]->Value);

TxtDataBaseLib::CommandsTableTxt::DeleteRow(orderString, tableName[2]);

LoadDataGrid();

}

private: System::Void button3\_Click(System::Object^ sender, System::EventArgs^ e);

private: System::Void button1\_Click(System::Object^ sender, System::EventArgs^ e) {

for (int i = 0; i < dataGridView1->RowCount; i++)

{

dataGridView1->Rows[i]->Selected = false;

for (int j = 0; j < dataGridView1->ColumnCount; j++)

if (dataGridView1->Rows[i]->Cells[j]->Value != nullptr)

if (dataGridView1->Rows[i]->Cells[j]->Value->ToString()->Contains(textBox1->Text))

{

dataGridView1->Rows[i]->Selected = true;

break;

}

}

}

private: System::Void printDocument1\_PrintPage(System::Object^ sender, System::Drawing::Printing::PrintPageEventArgs^ e) {

rowSel = dataGridView1->CurrentRow;

e->Graphics->DrawString(L"Чек на оказание услуг № " + rowSel->Cells[0]->Value, this->Font, Brushes::Black, 50, 50);

e->Graphics->DrawString(L"Фамилия закзчика " + rowSel->Cells[1]->Value, this->Font, Brushes::Black, 50, 90);

e->Graphics->DrawString(L"Имя закзчика " + rowSel->Cells[2]->Value, this->Font, Brushes::Black, 50, 110);

e->Graphics->DrawString(L"Номер телефона " + rowSel->Cells[3]->Value, this->Font, Brushes::Black, 50, 130);

e->Graphics->DrawString(L"Номер компьютера " + rowSel->Cells[4]->Value, this->Font, Brushes::Black, 50, 150);

e->Graphics->DrawString(L"Уровень поломки " + rowSel->Cells[5]->Value, this->Font, Brushes::Black, 50, 180);

e->Graphics->DrawString(L"Дата принятия " + rowSel->Cells[6]->Value, this->Font, Brushes::Black, 50, 210);

e->Graphics->DrawString(L"Дата окончание " + rowSel->Cells[7]->Value, this->Font, Brushes::Black, 50, 240);

e->Graphics->DrawString(L"Итого " + rowSel->Cells[8]->Value, this->Font, Brushes::Black, 50, 270);

e->Graphics->DrawString(L"Подпись продавца \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" , this->Font, Brushes::Black, 50, 315);

int hight = 1027;

Pen^ blackPen = gcnew Pen(Color::FromArgb(255, 0, 0, 255),3);

e->Graphics->DrawRectangle(blackPen, 25, 30, 360, 420);

MainForm.cpp

namespace ComputerRepair {

System::Void MainForm::menuItem\_Setting\_Click(System::Object^ sender, System::EventArgs^ e) {

SettingForm ^Sf = gcnew SettingForm(this);

Sf->ShowDialog();

}

System::Void MainForm::menuItem\_About\_Click(System::Object ^ sender, System::EventArgs ^ e)

{

AboutForm ^Af = gcnew AboutForm();

Af->ShowDialog();

}

System::Void MainForm::button5\_Click(System::Object ^ sender, System::EventArgs ^ e)

{

printPreviewDialog1->Document = printDocument1;

printPreviewDialog1->ShowDialog();

}

System::Void MainForm::button4\_Click(System::Object ^ sender, System::EventArgs ^ e)

{

ProfileForm ^Pf = gcnew ProfileForm(this);

Pf->ShowDialog();

}

System::Void MainForm::button3\_Click(System::Object ^ sender, System::EventArgs ^ e)

{

rowSel = dataGridView1->CurrentRow;

EditForm ^Ef = gcnew EditForm(this);

Ef->ShowDialog();

}

}

IdHelper.h

using namespace System;

using namespace System::IO;

using namespace System::Collections::Generic;

ref struct User

{

int Id;

String^ LastName;

String^ FirstName;

String^ Phone;

};

ref struct Order

{

int Id;

String^ NameComputer;

double Lvl;

double Cost;

DateTime BeginDate;

DateTime EndDate;

int UserId;

};

ref class IdHelper

{

public:

static int SaveAllId(int UserId, int OrderId) {

StreamWriter^ pwriter = gcnew StreamWriter("configID");

try

{

pwriter->WriteLine("UserId|"+UserId);

pwriter->WriteLine("OrderId|"+OrderId);

pwriter->Close();

return 0;

}

catch (System::Exception^ e)

{

pwriter->Close();

return -1;

}

}

static int GetOrderId() {

array<String^>^ str;

StreamReader^ reader = gcnew StreamReader("configID");

try

{

do

{

str = reader->ReadLine()->Split((char)'|');

if (str[0] == System::Convert::ToString("OrderId")) {

reader->Close();

return System::Convert::ToInt32(str[1]);

}

} while (reader->Peek() != -1);

}

catch (System::Exception^ e)

{

reader->Close();

return 0;

}

}

static List<User^>^ GetUsersList(String^ nameTable) {

List<User^>^ userList = gcnew List<User^>();

StreamReader^ reader = gcnew StreamReader(nameTable);

try

{

do

{

array<String^>^ str = reader->ReadLine()->Split((char)'|');

if (str != nullptr) {

User^ user = gcnew User();

user->Id = Convert::ToInt32(str[0]);

user->LastName = str[1];

user->FirstName = str[2];

user->Phone = str[3];

userList->Add(user);

str = nullptr;

}

} while (reader->Peek() != -1);

reader->Close();

return userList;

}

catch (System::Exception^ e)

{

reader->Close();

return userList=nullptr;

}

}

static List<Order^>^ GetOrdersList(String^ nameTable) {

List<Order^>^ orderList = gcnew List<Order^>();

StreamReader^ reader = gcnew StreamReader(nameTable);

try

{

do

{

array<String^>^ str = reader->ReadLine()->Split((char)'|');

if (str != nullptr) {

Order^ order = gcnew Order();

order->Id = Convert::ToInt32(str[0]);

order->NameComputer = str[1];

order->Lvl = Convert::ToDouble(str[2]);

order->Cost = Convert::ToDouble(str[3]);

order->BeginDate = Convert::ToDateTime(str[4]);

order->EndDate = Convert::ToDateTime(str[5]);

order->UserId = Convert::ToInt32(str[6]);

orderList->Add(order);

str = nullptr;

}

} while (reader->Peek() != -1);

reader->Close();

return orderList;

}

catch (System::Exception^ e)

{

reader->Close();

return orderList = nullptr;

}

}

};

CommandsTableTxt.cpp

List<String^>^ TxtDataBaseLib::CommandsTableTxt::ReadRwos(String ^ name)

{

listDTableTxt = gcnew List<String^>();

StreamReader^ reader = gcnew StreamReader(name);

try

{

do

{

listDTableTxt->Add(reader->ReadLine());

} while (reader->Peek() != -1);

return listDTableTxt;

}

catch (System::Exception^ e)

{

reader->Close();

return listDTableTxt = nullptr;

}

}

int TxtDataBaseLib::CommandsTableTxt::WriteRowsList(List<String^>^ listData, String ^ name)

{

StreamWriter^ pwriter = gcnew StreamWriter(name);

try

{

for each (String^ var in listData)

{

pwriter->WriteLine(var);

}

pwriter->Close();

return 0;

}

catch (System::Exception^ e)

{

pwriter->Close();

return -1;

}

}

int TxtDataBaseLib::CommandsTableTxt::AddNewRowsList(List<String^>^ listData, String ^ name)

{

StreamWriter^ pwriter = gcnew StreamWriter(name, true);

try

{

for each (String^ var in listData)

{

pwriter->WriteLine(var);

}

pwriter->Close();

return 0;

}

catch (System::Exception^ e)

{

pwriter->Close();

return -1;

}

}

int TxtDataBaseLib::CommandsTableTxt::WriteRow(String ^ Data, String ^ name)

{

StreamWriter^ pwriter = gcnew StreamWriter(name);

try

{

pwriter->WriteLine(Data);

pwriter->Close();

return 0;

}

catch (System::Exception^ e)

{

pwriter->Close();

return -1;

}

}

int TxtDataBaseLib::CommandsTableTxt::AddNewRow(String ^ Data, String ^ name)

{

StreamWriter^ pwriter = gcnew StreamWriter(name, true);

try

{

pwriter->WriteLine(Data);

pwriter->Close();

return 0;

}

catch (System::Exception^ e)

{

pwriter->Close();

return -1;

}

}

List<String^>^ TxtDataBaseLib::CommandsTableTxt::Search(String ^ nameSearch, String ^ nameTable)

{

listDTableTxt = gcnew List<String^>();

StreamReader^ reader = gcnew StreamReader(nameTable);

try

{

do

{

textFile = String::Concat(reader->ReadLine());

if(textFile->Contains(nameSearch)==true)

listDTableTxt->Add(textFile);

} while (reader->Peek() != -1);

reader->Close();

return listDTableTxt;

}

catch (System::Exception^ e)

{

reader->Close();

return listDTableTxt = nullptr;

}

}

List<String^>^ TxtDataBaseLib::CommandsTableTxt::DeleteRow(String ^ strDelete, String ^ nameTable)

{

listDTableTxt = gcnew List<String^>();

StreamReader^ reader = gcnew StreamReader(nameTable);

try

{

do

{

textFile = String::Concat(reader->ReadLine());

if (textFile->Contains(strDelete)!=true)

listDTableTxt->Add(textFile);

} while (reader->Peek() != -1);

reader->Close();

WriteRowsList(listDTableTxt, nameTable);

return listDTableTxt;

}

catch (System::Exception^ e)

{

reader->Close();

return listDTableTxt = nullptr;

}

}

};

TxtDataBaseLib.cpp

using namespace System;

using namespace System::IO;

void TxtDataBaseLib::JobTxtDB::buldPath()

{

try

{

int index = 0;

listPath = gcnew List<String^>();

StreamReader^ reader = gcnew StreamReader("configDatabase");

do

{

listPath->Add(reader->ReadLine());

} while (index++, reader->Peek() != -1);

reader->Close();

pathDataBase = String::Concat(listPath[0],"\\");

}

catch (FileNotFoundException^ ex)

{

listPath->Add("Файл не найден");

}

catch (System::Exception^ e)

{

listPath->Add("-1");

}

}

int TxtDataBaseLib::JobTxtDB::CreateDTTxt(String ^ name)

{

try

{

StreamWriter^ pwriter = gcnew StreamWriter("configDatabase");

pwriter->WriteLine(name);

pwriter->Close();

Directory::CreateDirectory(name);

return 0;

}

catch (System::Exception^ e)

{

return -1;

}

}

int TxtDataBaseLib::JobTxtDB::CreateDTableTxt(String ^ name)

{

try

{

buldPath();

StreamWriter^ pwriter = gcnew StreamWriter(pathDataBase + name);

pwriter->Close();

pwriter = gcnew StreamWriter("configDatabase", true);

pwriter->WriteLine(pathDataBase + name);

pwriter->Close();

return 0;

}

catch (System::Exception^ e)

{

return -1;

}

}

int TxtDataBaseLib::JobTxtDB::DropDTTxt()

{

try

{

buldPath();

for (int index = listPath->Count-1; index > 0; index--)

{

File::Delete(listPath[index]);

}

Directory::Delete(listPath[0]);

File::Delete("configDatabase");

return 0;

}

catch (System::Exception^ e)

{

return -1;

}

}

int TxtDataBaseLib::JobTxtDB::DropDTableTxt(String ^ name)

{

try

{

File::Delete(name);

listPath->Remove(name);

StreamWriter^ pwriter = gcnew StreamWriter("configDatabase");

for each (String^ var in listPath)

{

pwriter->WriteLine(var);

}

pwriter->Close();

return 0;

}

catch (System::Exception^ e)

{

return -1;

}

}

List<String^>^ TxtDataBaseLib::JobTxtDB::ListTablseAndDTTxt()

{

try

{

buldPath();

return listPath;

}

catch (System::Exception^ e)

{

return listPath;

}

}

# **ПРИЛОЖЕНИЕ Б**

**(Обязательное)**

**Скриншот работы программы**

