Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Петрозаводский государственный университет»

Физико-технический институт

Кафедра информационно-измерительных систем и физической электроники

**Командное задание по разработке программного обеспечения в рамках “Унифицированного процесса разработки ПО”**

курсовой проект по дисциплине «Технология программирования»

Выполнили:

студенты группы 21318

Тихманович А.А

Луйкин В.И

Иванов О.И

«\_\_ » \_\_\_\_\_\_\_\_ 2021 г.

Проверил:

канд. физ.-мат. наук.,

Бульба А.В

«\_\_ » \_\_\_\_\_\_\_\_ 2021 г.

Петрозаводск 2021

**Цель работы:** закрепить полученные знания в контексте использования языков С++ и UML, практически применить концепцию “Унифицированного процесса разработки ПО”, сопровождая  работу с помощью распределенной системы контроля версий Git через веб-сервис GitHub.

**Кратко о программной реализации:** средой разработки мы выбрали Microsoft Visual Studio 2019, язык C++, перечисление всех созданных нами заголовочников и единиц компиляции: CarRepair.h (В этом файле описываются все элементы, используемые в программе(классы, их атрибуты и методы)), CarMenu.cpp (здесь содержится main()), CarProject.cpp (содержится реализации функции, объявленных в заголовочном файле).

**Описание процесса разработки:**

1. **Краткое словесное описание сюжета.**

К нам обратился заказчик, который владеет автомастерской с большой загруженностью, с просьбой создать для его сотрудников консольную программу. Программа должна помочь менеджеру автомастерской в организации расписания заказов. Данная программа должна давать возможность менеджеру добавлять запись услуги с указанием имени и фамилии клиента, информации о машине (марка и модель), названия услуги, ее стоимости и даты исполнения. Новые записи должны автоматически вноситься в общее расписание, и у менеджера будет возможность посмотреть это расписание. Также программа будет предоставлять возможность изменения и удаления записей из расписания. Еще в программа предусматривает функцию просмотра прайс-листа (название услуги и ее стоимости). Программа обязательно должна иметь возможность учета доходов и расходов и также она должна формировать отчет за указанный пользователем период.

1. **Описание имеющихся у заказчика материалов (данных) с которыми ему приходится работать.**

Таблица с расписанием:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Имя  Клиента | Фамилия  Клиента | Марка  Машины | Модель  Машины | Название  Услуги | Времязаписи | СтоимостьУслуги |
| Тихм | Артем | Дэу | Нексия | Замена масла | 2021-12-12 18:00 | 300 |
|  |  |  |  |  |  |  |

Прайс-лист:

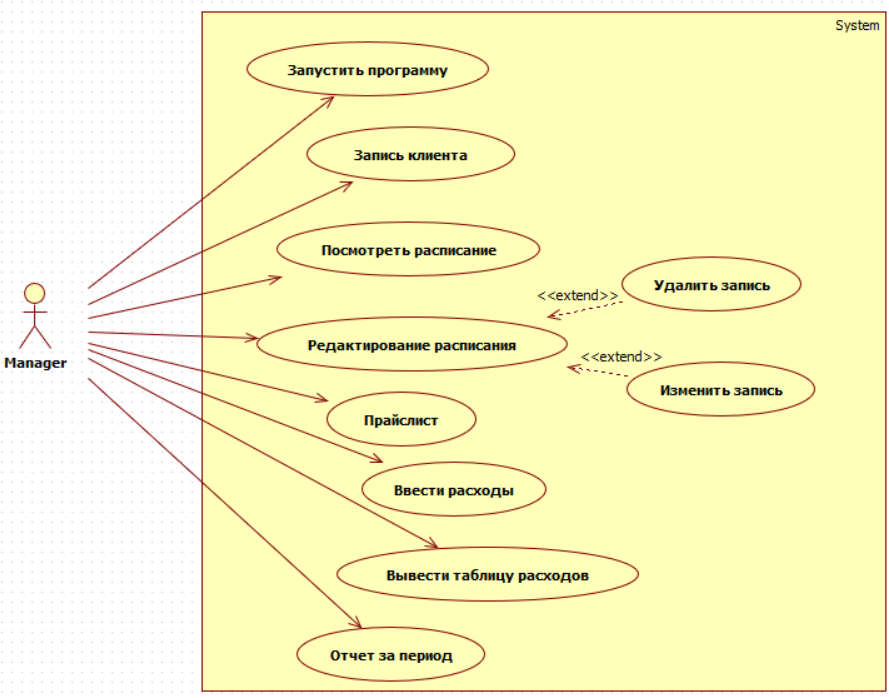
|  |  |
| --- | --- |
| Услуга | Стоимость |
| Замена масла | 300 |
| Полировка кузова | 800 |
| Замена ремня ГРМ | 2000 |
| Замена стекол | 2700 |
| Ремонт двигателя | 500 |
|  |  |

1. **Действующие субъекты (actors) и их функциональные обязанности.**

С программой работает только один человек: менеджер. Таким образом, один и тот же человек вводит информацию и просматривает ее.

1. **Список вариантов использования, которые могут возникнуть при работе программы. Диаграмма вариантов использования.**

* Запустить программу
* Запись клиента
* Посмотреть расписание
* Редактирование расписания
* Удаление записи
* Изменение записи
* Прайс-лист
* Ввод расходов
* Вывести таблицу расходов
* Отчет на период



1. **Текст сценария для каждого варианта использования.**

«Запустить программу»

Во время запуска программы, на экран должно выводиться меню, из которого пользователь может выбрать нужное действие.

«Запись клиента»

На экране должно отобразиться сообщение, в котором программа просит менеджера ввести имя клиента, фамилию клиента, марку машины, модель машины и время записи. Название услуги и стоимость выбирается из уже подготовленного прайс-листа. Эта информация должна заноситься в таблицу с расписанием. Этот пункт меню выбирается путем нажатия цифры 1 на клавиатуре.

«Посмотреть расписание»

На экране должна отобразиться таблица с расписанием, отсортированная по дате и времени. Этот пункт меню выбирается путем нажатия цифры 2 на клавиатуре.

«Редактирование расписания»

На экране выводится вся таблица. Менеджер выбирает необходимую запись и вводит ее дату. Далее предоставляется выбор удаления или редактирование записи. Если менеджер хочет удалить запись, то вводит «2», а если изменить, то вводит «1». Этот пункт меню выбирается путем нажатия цифры 3 на клавиатуре.

«Удаление записи»

На экран выводится сообщение с подтверждением об удалении. Если пользователь отказался удалять, то его перебрасывает в главное меню, а если согласился, то запись удаляется и его также перебрасывает в главное меню.

«Изменение записи»

На экран выводится сообщение с выбором пункта изменения элементов записи. Менеджер выбирает нужный пункт и вводит новую информацию, которая заносится в таблицу в место старой.

«Прайс-лист»

На экране должна отобразиться таблица с прайс-листом, данные для которого находятся в текстовом файле. Этот пункт меню выбирается путем нажатия цифры 4 на клавиатуре.

«Ввести расходы»

Экран ввода расхода должен содержать приглашение пользователю на ввод закупочного товара, стоимости товара и даты закупки. Затем программа вставляет информацию в таблицу расходов. Этот пункт меню выбирается путем нажатия цифры 5 на клавиатуре.

«Вывести таблицу расходов»

На экране выводится таблица расходов. В которой содержится информация о продукте расхода, дате расхода и стоимости закупаемой продукции. Этот пункт меню выбирается путем нажатия цифры 6 на клавиатуре.

«Отчет за период»

На экране вводится период за который фиксируется доходы и расходы и рассчитывается прибыль. Этот пункт меню выбирается путем нажатия цифры 7 на клавиатуре.

«Добавить запись» Сценарий 2.

На экране отображается экран ввода новой записи. Введенная дата уже занята какой-то другой записью. Пользователю выводится сообщение об ошибке.

«Редактирование расписания» Сценарий 2.

На экране отображается экран ввода существующей записи. Пользователь ошибся и ввел несуществующий номер записи. В итоге выводится сообщение об ошибке.

«Изменение записи» Сценарий 2.

На экране отображается экран изменения существующей записи. При изменении записи пользователь изменил дату на существующую. Выводится сообщение об ошибке.

1. **Диаграммы действий.**

Диаграмма действий для варианта использования «Запись клиента»:

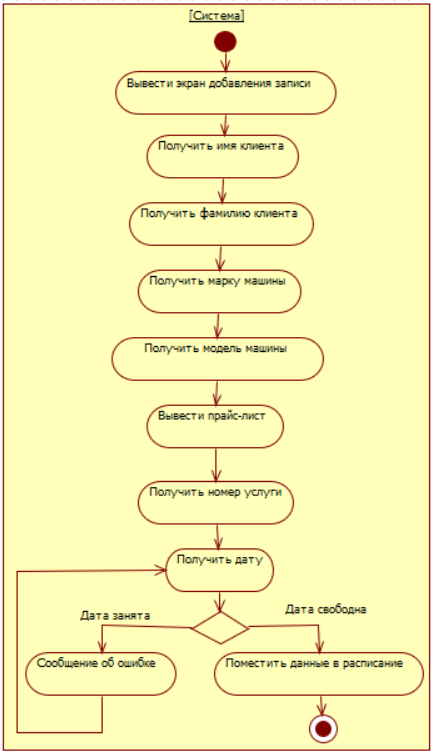


Диаграмма действий для варианта использования «Редактирование расписания»:

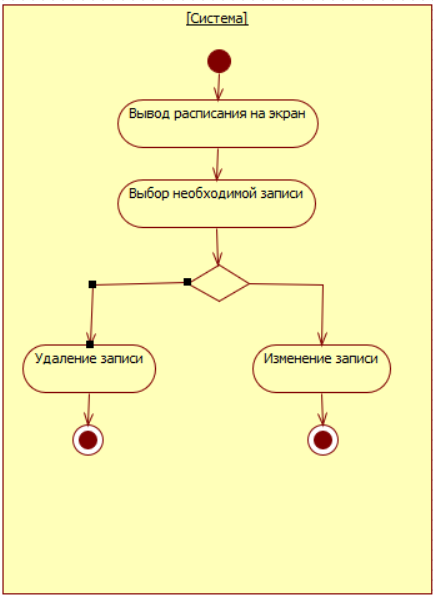


Диаграмма действий для варианта использования «Удаление записи»:

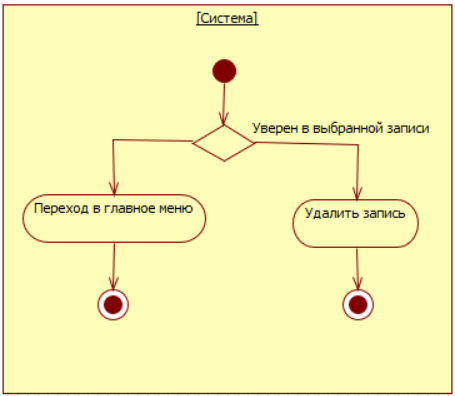


Диаграмма действий для варианта использования «Изменение записи»:

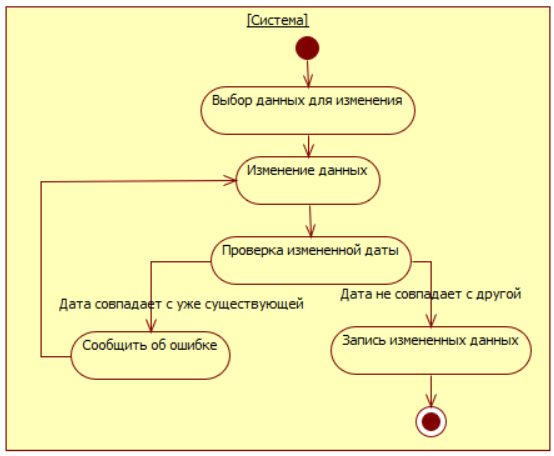


Диаграмма действий для варианта использования «Ввести расходы»:

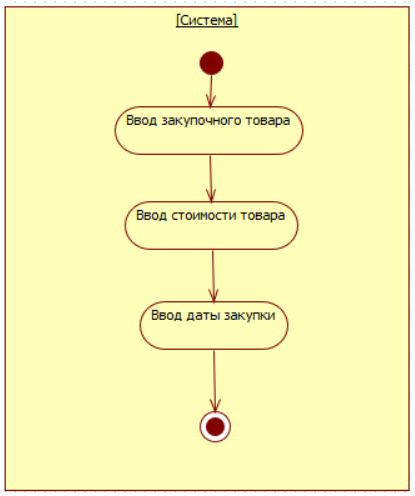
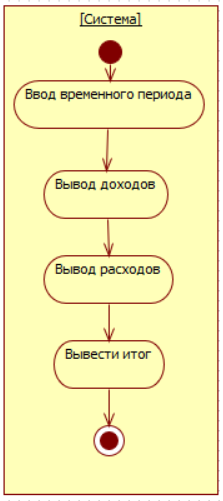


Диаграмма действий для варианта использования «Отчет за период»:



1. **Список существительных.**
2. Пользовательский интерфейс
3. Запись
4. Добавления записи
5. Имя клиента
6. Фамилия клиента
7. Название услуги
8. Стоимость услуги
9. Марка машины
10. Модель машины
11. Дата исполнения услуги
12. Время исполнения услуги
13. Прайс-лист
14. Расписание
15. Выбор изменения/удаления записи
16. Изменения записи
17. Удаления записи
18. Отчет за период
19. Сумма оказанных услуг
20. Номер услуги
21. Ввод расходов
22. Доходы
23. Расходы
24. Таблица расходов
25. **Уточненный список существительных.**
26. Пользовательский интерфейс.
27. Запись
28. Добавления записи
29. Расписание
30. Выбор изменения/удаления записи
31. Изменения записи
32. Удаления записи
33. Расходы
34. Ввод расходов
35. Таблица расходов
36. Прайс-лист
37. Отчет за период
38. **Список атрибутов.**

Класс Запись: Имя клиента, Фамилия клиента, Марка машины, Модель машины, Название услуги, Стоимость услуги, День, Месяц, Год, Час, Минута.

Класс Отчет за период: Год1, Месяц1, День1, Год2, Месяц2, День2, указатель на Расписание , указатель на Таблицу расходов, Доход, Расход, Итог.

Класс Экран пользовательского интерфейса: указатель на Экран добавления записи, указатель на Расписание, указатель на Экран выбора изменения/удаления записи, указатель на Прайс-лист, указатель на Отчет за период, указатель на Таблицу расходов, указатель на Экран добавления расходов.

Класс Добавления записи: указатель на Прайс-лист, Имя клиента, Фамилия клиента, Марка машины, Модель машины, Название услуги, Стоимость услуги, День, Месяц, Год, Час, Минута, указатель на Расписание, Номер услуги.

Класс Расписание: указатель на Экран добавления записи, Имя клиента, Фамилия клиента, Марка машины, Модель машины, Название услуги, Стоимость услуги, День, Месяц, Год, Час, Минута, указатель на Запись.

Класс Экран выбора изменения/удаления записи: указатель на Расписание, указатель на Экран изменения записи, указатель на Экран удаления записи, День, Месяц, Год, Час, Минута.

Класс Экран изменения записи: Имя клиента, Фамилия клиента, Марка машины, Модель машины, Название услуги, Стоимость услуги, День, Месяц, Год, Час, Минута, указатель на Запись, указатель на Расписание.

Класс Экран удаления записи: указатель на Расписание, День, Месяц, Год, Час, Минута.

Класс Прайс-лист: Название услуги, Стоимость услуги, Номер услуги.

Класс Таблица расходов: Расход, День, Месяц, Год, Цена.

Класс Расходы: Расход, День, Месяц, Год, Цена.

Класс Экран добавления расхода: указатель на Таблицу расходов, Расход, День, Месяц, Год, Цена.

1. **Список сообщений.**

«Запустить программу»

**Выводится:** При запуске программы посылается сообщение классу «Пользовательский интерфейс» показывая ему вывести себя. Имя: Menu()

**Выбрать:** Выводит приглашение выбора пункта меню и посылает сообщение классу, соответствующему выбранному пункту меню.

«Запись клиента»

**Отобразится**: Экран пользовательского интерфейса должен послать сообщение классу «добавления записи», приказывая ему вывести себя и получить данные от пользователя. Имя:setClient()

**Просит ввести**: Взаимодействие класса «добавления записи» с пользователем. setClient() выводит приглашение и записывает ответы пользователя.

**Вывести прайс-лист**: Экран пользовательского интерфейса должен послать сообщение классу «прайс-лист» приказывая ему вывести себя. Имя: showPriceList()

**Заносится**: Объект класса “Экран добавления записи” посылает сообщение объекту класса “Расписание”. В качестве аргумента используется новый объект класса “Запись ”. Объект “Расписание” может затем вставить этот новый объект в свой список.

«Посмотреть расписание»

**Отобразится:** Экран пользовательского интерфейса должен послать сообщение классу «Расписание», приказывая ему вывести себя и отсортировать по дате и времени. Имя: showTimeTable()

«Редактирование расписания»

**Выводится:** Экран пользовательского интерфейса должен послать сообщение классу «Расписание», приказывая ему вывести себя и отсортировать по дате и времени. Имя: showTimetable()

**Просит ввести:** Взаимодействие класса «Расписание» с пользователем. Выводит приглашение и записывает ответ пользователя (дата и время записи) Имя: getRecordDate()

**Просит выбрать:** Отображается приглашение, записывается выбор пользователя и обращается к соответствующему классу (классы удаления или изменения записи)

«Удаление записи»

**Выводится:** Выводится сообщение с запросом подтверждения действия (удаления записи)

**Перебрасывает:** Взаимодействие класса «Удаление записи» с «Пользовательским интерфейсом». Происходит перенос пользователя в главное меню.

**Удаляется:** Класс «Удаление записи» посылает сообщение классу «Расписание» об удалении определенной строки из таблицы с расписанием. Имя: DeleteClient()

«Изменение записи»

**Выводится:** Выводится список атрибутов класса «Запись» для последующего выбора определенного атрибута для изменения информации.

**Выбор:** Отображается приглашение, записывается выбор пользователя и обращается к соответствующему атрибуту (имя клиента, фамилия клиента, марка машины, модель машины, услуга, стоимость или дата записи)

**Ввод:** Взаимодействие класса «Изменение записи» с пользователем. Выводит приглашение и записывает ответ пользователя (Новое значение атрибута). Имя: EditInfo()

**Заносится:** «Изменение записи» посылает сообщение классу «Расписание». Старая информация на изменяемом месте удаляется и на ее место записывается новая.

«Прайс-лист»

**Вывод:** Экран пользовательского интерфейса должен послать сообщение классу «прайс-лист» приказывая ему вывести себя. Имя: showPriceList()

«Ввести расходы»

**Ввод:** Взаимодействие класса «Ввести расходы» с пользователем. Выводит приглашение и записывает ответ пользователя (Расходы) Имя: setExpenses()

**Заносится:** Класс «Ввести расходы» посылает сообщение классу «Таблица расходов» о добавлении новой информации.

**«**Вывести таблицу расходов**»**

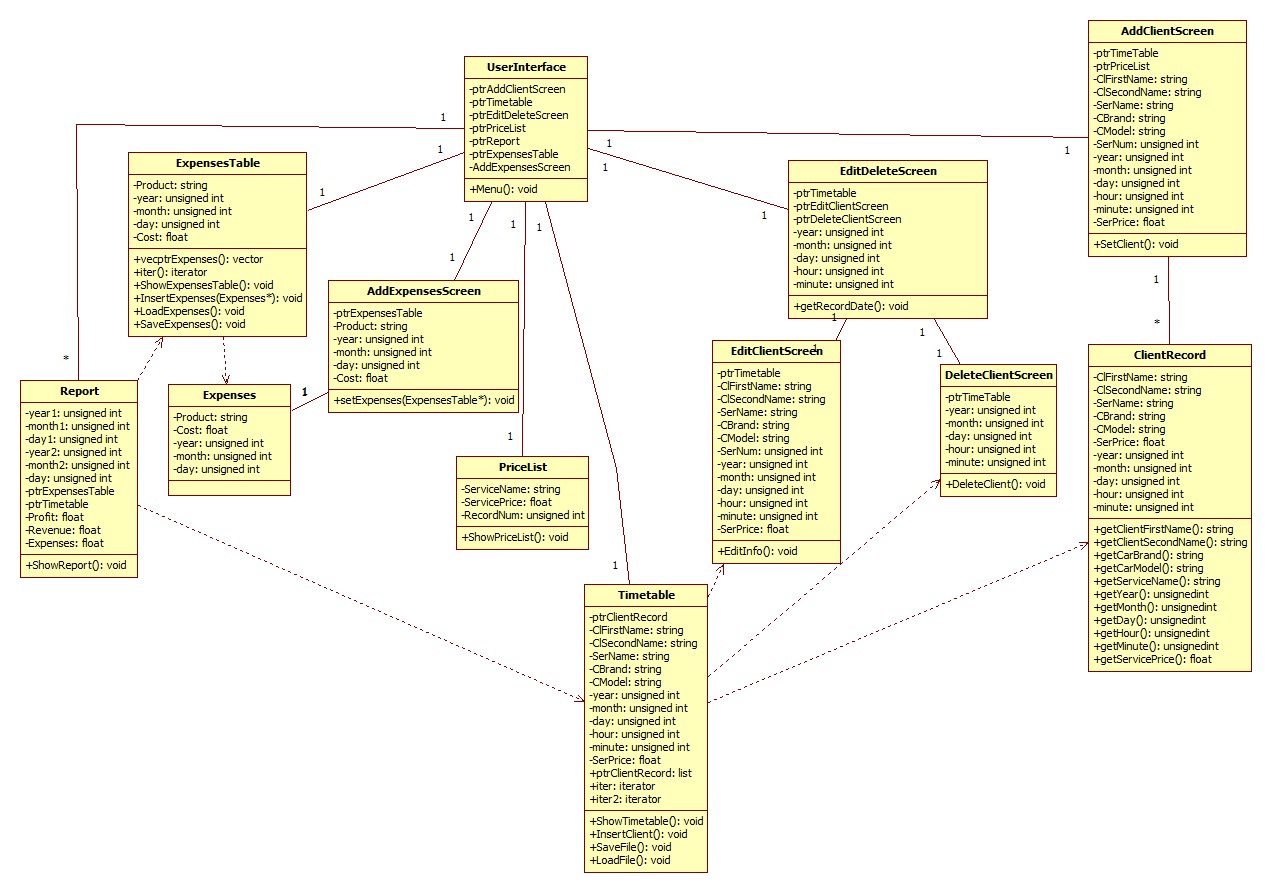
**Отобразится:** Экран пользовательского интерфейса должен послать сообщение классу «Таблица расходов», приказывая ему вывести себя. Имя: showExpensesTable()

«Отчет за период»

**Ввод:** Взаимодействие класса «Отчет за период» с пользователем. Выводит приглашение и записывает ответ пользователя (Период).

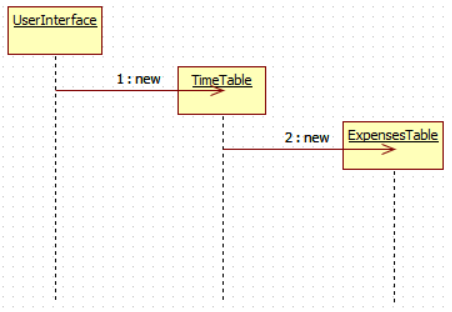
**Вывод:** Выводится период, данные о доходах (обрабатывает информацию из класса «Расписание»), данные о расходах(обрабатывает информацию из класса «Таблица расходов») и итог(формируется из разности доходов и расходов). Имя: showReport()

1. **Диаграмма классов.**

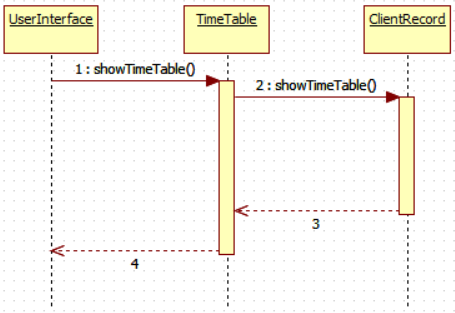


1. **Диаграммы последовательностей.**

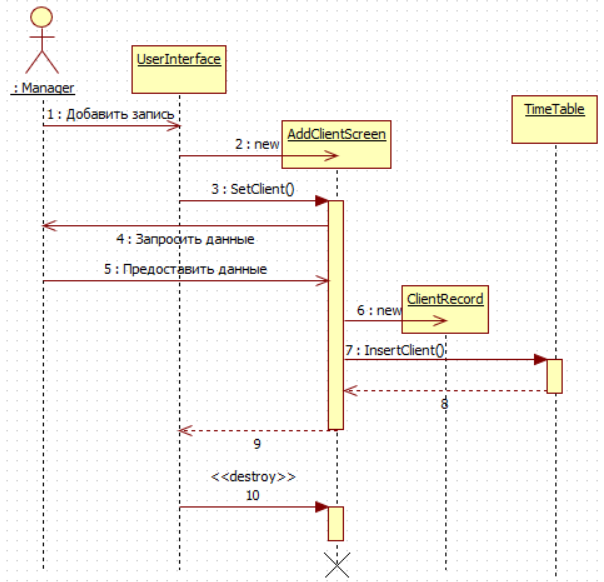
**Диаграмма последовательностей для варианта использования «Запустить программу»:**



**Диаграмма последовательностей для варианта использования «Вывод расписания»:**



**Диаграмма последовательностей для варианта использования «Запись клиента»:**



1. **Заголовочный файл CarRepair.h.**

#include <iostream>

#include <string>

#include <list>

#include <vector>

using namespace std;

class PriceList

{

private:

string ServiceName, line;

float ServicePrice;

unsigned int RecordNum;

public:

void ShowPrices();

};

class ClientRecord

{

private:

string ClFirstName, ClSecondName, SerName, CBrand, CModel;

float SerPrice;

unsigned int year, month, day, hour, minute;

public:

ClientRecord(string ClFN, string ClSN, string CB, string CM, string SN, unsigned int yy, unsigned int mm, unsigned int dd, unsigned int hh, unsigned int mint, float SP);

~ClientRecord();

string getClientFirstName();

string getClientSecondName();

string getCarBrand();

string getCarModel();

string getServiceName();

unsigned int getYear();

unsigned int getMonth();

unsigned int getDay();

unsigned int getHour();

unsigned int getMinute();

float getServicePrice();

};

class TimeTable

{

private:

string ClFirstName, ClSecondName, SerName, CBrand, CModel;

float SerPrice;

unsigned int year, month, day, hour, minute;

public:

list <ClientRecord\*> ptrClientRecord;

list <ClientRecord\*>::iterator iter,iter2;

void ShowTimeTable();

void InsertClient(ClientRecord\*);

void SaveFile();

void LoadFile();

};

class AddClientScreen

{

private:

//ClientRecord\* ptrClientRecord;

TimeTable\* ptrTimeTable;

PriceList\* ptrPriceList;

string ClFirstName, ClSecondName, SerName, CBrand, CModel, line;

unsigned int SerNum, year, month, day, hour, minute;

float SerPrice;

public:

AddClientScreen(TimeTable\* ptrTT) : ptrTimeTable(ptrTT)

{

/\* \*/

}

void setClient();

};

class Expenses

{

public:

string Product;

unsigned int year, month, day;

float Cost;

Expenses(string P, unsigned int y, unsigned int m, unsigned int d, float ct) :

Product(P), year(y), month(m), day(d), Cost(ct)

{

/\* \*/

}

};

class ExpensesTable

{

private:

string Product;

unsigned int year, month, day;

float Cost;

public:

vector<Expenses\*> vecptrExpenses;

vector<Expenses\*>::iterator iter;

void insertExpenses(Expenses\*);

void ShowExpensesTable();

void LoadExpenses();

void SaveExpenses();

};

class AddExpensesScreen

{

private:

ExpensesTable\* ptrExpensesTable;

string Product;

unsigned int year, month, day;

float Cost;

public:

AddExpensesScreen(ExpensesTable\*);

void setExpenses();

};

class EditClientScreen

{

private:

TimeTable\* ptrTimeTable;

string ClFirstName, ClSecondName, SerName, CBrand, CModel, line;

unsigned int SerNum, year, month, day, hour, minute;

float SerPrice;

public:

EditClientScreen(TimeTable\* ptrTT) :ptrTimeTable(ptrTT)

{

}

void EditInfo(unsigned int YY, unsigned int MM, unsigned int DD, unsigned int hh, unsigned int mm);

};

class DeleteClientScreen

{

private:

TimeTable\* ptrTimeTable;

unsigned int year, month, day, hour, minute;

public:

DeleteClientScreen(TimeTable\* ptrTT) :ptrTimeTable(ptrTT)

{

}

void DeleteClient(unsigned int YY, unsigned int MM, unsigned int DD, unsigned int hh, unsigned int mm);

};

class EditDeleteScreen

{

private:

TimeTable\* ptrTimeTable;

EditClientScreen\* ptrEditClientScreen;

DeleteClientScreen\* ptrDeleteClientScreen;

unsigned int year, month, day, hour, minute, choice;

public:

EditDeleteScreen(TimeTable\* ptrTT) :ptrTimeTable(ptrTT)

{

}

~EditDeleteScreen();

void getRecordDate();

};

class Report

{

private:

unsigned int year1, month1, day1, year2, month2, day2;

float Revenue, Expenses, Profit;

TimeTable\* ptrTimeTable;

ExpensesTable\* ptrExpensesTable;

public:

Report(TimeTable\* ptrTT, ExpensesTable\* ptrET): ptrTimeTable(ptrTT), ptrExpensesTable(ptrET)

{

}

void ShowReport();

};

class UserInterface

{

private:

PriceList\* ptrPriceList;

AddClientScreen\* ptrAddClientScreen;

EditDeleteScreen\* ptrEditDeleteScreen;

TimeTable\* ptrTimeTable;

ExpensesTable\* ptrExpensesTable;

AddExpensesScreen\* ptrAddExpensesScreen;

Report\* ptrReport;

char choice;

public:

UserInterface();

~UserInterface();

void Menu();

};

1. **Исходный файл CarMenu.cpp**

#include "CarRepair.h"

int main()

{

setlocale(LC\_ALL, "rus");

UserInterface Interface;

Interface.Menu();

return 0;

}

**Исходный файл CarProject.cpp**

#include <stdio.h>

#include <iostream>

#include <iomanip>

#include <fstream>

#include "CarRepair.h"

using namespace std;

UserInterface::UserInterface()

{

ptrTimeTable = new TimeTable;

ptrExpensesTable = new ExpensesTable;

ptrTimeTable->LoadFile();

ptrExpensesTable->LoadExpenses();

}

UserInterface::~UserInterface()

{

delete ptrTimeTable;

delete ptrExpensesTable;

}

void UserInterface::Menu()

{

cout << "Добро пожаловать" << endl;

while (true)

{

cout << "1. Добавить запись" << endl;

cout << "2. Посмотреть расписание" << endl;

cout << "3. Редактировать расписание" << endl;

cout << "4. Посмотреть прайс-лист" << endl;

cout << "5. Ввести расходы" << endl;

cout << "6. Посмотреть таблицу расходов" << endl;

cout << "7. Посмотреть отчет за период" << endl;

cout << "8. Выход" << endl;

cout << "Введите цифру нужного пункта" << endl;

cin >> choice;

system("cls");

switch (choice)

{

case '1':

{

cout << "Добавление записи" << endl;

ptrAddClientScreen = new AddClientScreen(ptrTimeTable);

ptrAddClientScreen->setClient();

delete ptrAddClientScreen;

system("pause");

break;

}

case '2':

{

char ch;

cout << "Расписание" << endl;

cout << "ИмяКлиента" << "| |"<< "ФамилияКлиента" << "| |"<< "МаркаМашины" << "| |" <<"МодельМашины" << "| |" << "НазваниеУслуги" << "| |" << left << setw(20) << "ВремяЗаписи" << "| |" << "СтоимостьУслуги" << endl;

ptrTimeTable->ShowTimeTable();

cout << "Хотите сохранить данные? (y/n)" << endl;

cin >> ch;

if (ch == 'y')

{

ptrTimeTable->SaveFile();

}

system("pause");

break;

}

case '3':

{

cout << "Редактирование расписания" << endl;

ptrEditDeleteScreen = new EditDeleteScreen(ptrTimeTable);

ptrTimeTable->ShowTimeTable();

ptrEditDeleteScreen->getRecordDate();

delete ptrEditDeleteScreen;

break;

}

case '4':

{

cout << "Прайс-лист" << endl;

cout << "НомерУслуги" << "| |" << left << setw(20) << "НазваниеУслуги" << "| |" << "СтоимостьУслуги" << endl;

PriceList a;

a.ShowPrices();

system("pause");

break;

}

case '5':

{

cout << "Ввод расходов" << endl;

ptrAddExpensesScreen = new AddExpensesScreen(ptrExpensesTable);

ptrAddExpensesScreen->setExpenses();

delete ptrAddExpensesScreen;

system("pause");

break;

}

case '6':

{

char ch;

cout << "Таблица расходов" << endl;

cout << left << setw(10) << "Расход" << "| | " << "ДатаРасхода" << "| | " << "СуммаРасхода" << endl;

ptrExpensesTable->ShowExpensesTable();

cout << "Хотите сохранить данные? (y/n)" << endl;

cin >> ch;

if (ch == 'y')

{

ptrExpensesTable->SaveExpenses();

}

system("pause");

break;

}

case '7':

{

cout << "Отчет за период" << endl;

ptrReport = new Report(ptrTimeTable, ptrExpensesTable);

ptrReport->ShowReport();

delete ptrReport;

system("Pause");

break;

}

case '8':

{

char ch;

cout << "До свидания" << endl;

cout << "Хотите сохранить данные? (y/n)" << endl;

cin >> ch;

if (ch == 'y')

{

ptrTimeTable->SaveFile();

ptrExpensesTable->SaveExpenses();

}

return;

break;

}

default:

{

cout << "Нет такого пункта" << endl;

system("pause");

break;

}

}

system("cls");

}

}

void AddClientScreen::setClient()

{

cout << "Введите Имя и Фамилию клиента:" << endl;

cin >> ClFirstName;

cin >> ClSecondName;

cout << "Введите марку и модель машины:" << endl;

cin >> CBrand;

cin >> CModel;

PriceList PList;

PList.ShowPrices();

cout << "Введите Номер услуги: " << endl;

cin >> SerNum;

ifstream in (".\\PriceList.txt"); //for .exe

//ifstream in ("../PriceList.txt");

if (in.is\_open())

{

while (getline(in, line))

{

if (line.find(to\_string(SerNum)) != string::npos)

{

string s1[3];

int j = 0;

for (int i = 0; i <= 2; i++)

{

while (line[j] != ';')

{

s1[i] = s1[i] + line[j];

j++;

}

j++;

}

SerName = s1[1];

SerPrice = stof(s1[2]);

}

}

}

in.close();

cout << "Введите дату выполнения услуги(YYYY MM DD hh mm):" << endl;

bool search = true;

bool write;

while (search)

{

cin >> year;

cin >> month;

cin >> day;

cin >> hour;

cin >> minute;

write = true;

if (ptrTimeTable->ptrClientRecord.empty())

//запись даты, так как нет записей

search = false;

else

{

ptrTimeTable->iter = ptrTimeTable->ptrClientRecord.begin();

while (ptrTimeTable->iter != ptrTimeTable->ptrClientRecord.end() && write)

{

if (year == (\*ptrTimeTable->iter)->getYear() && month == (\*ptrTimeTable->iter)->getMonth() && day == (\*ptrTimeTable->iter)->getDay() && hour == (\*ptrTimeTable->iter)->getHour() && minute == (\*ptrTimeTable->iter)->getMinute())

{

cout << "Запись на эту дату уже существует." << endl;

write = false;

}

\*ptrTimeTable->iter++;

}

if (write)

{

//запись даты, так как нет совпадений

search = false;

}

}

}

ClientRecord\* ptrClientRecord = new ClientRecord(ClFirstName, ClSecondName, CBrand, CModel, SerName, year, month, day, hour, minute, SerPrice);

ptrTimeTable->InsertClient(ptrClientRecord);

}

void PriceList::ShowPrices()

{

ifstream in (".\\PriceList.txt"); //for .exe

//ifstream in ("../PriceList.txt");

if (in.is\_open())

{

while (getline(in, line))

{

string s1[3];

int j = 0;

for (int i = 0; i <= 2; i++)

{

while (line[j] != ';')

{

s1[i] = s1[i] + line[j];

j++;

}

j++;

}

RecordNum = atoi(s1[0].c\_str());

ServiceName = s1[1];

ServicePrice = stof(s1[2]);

cout << left << setw(10) << RecordNum << " | | " << setw(18) << ServiceName << " | | " << ServicePrice << endl;

}

}

in.close();

}

ClientRecord::ClientRecord(string ClFN, string ClSN, string CB, string CM, string SN,

unsigned int yy, unsigned int mm, unsigned int dd, unsigned int hh, unsigned int mint, float SP) : //конструктор

ClFirstName(ClFN), ClSecondName(ClSN), CBrand(CB), CModel(CM), SerName(SN), year(yy), month(mm),

day(dd), hour(hh), minute(mint), SerPrice(SP)

{

}

ClientRecord::~ClientRecord()

{

}

string ClientRecord::getClientFirstName()

{

return ClFirstName;

}

string ClientRecord::getClientSecondName()

{

return ClSecondName;

}

string ClientRecord::getCarBrand()

{

return CBrand;

}

string ClientRecord::getCarModel()

{

return CModel;

}

string ClientRecord::getServiceName()

{

return SerName;

}

unsigned int ClientRecord::getYear()

{

return year;

}

unsigned int ClientRecord::getMonth()

{

return month;

}

unsigned int ClientRecord::getDay()

{

return day;

}

unsigned int ClientRecord::getHour()

{

return hour;

}

unsigned int ClientRecord::getMinute()

{

return minute;

}

float ClientRecord::getServicePrice()

{

return SerPrice;

}

void TimeTable::ShowTimeTable()

{

if (ptrClientRecord.empty())

cout << "Нет записей\n" << endl;

else

{

iter = ptrClientRecord.begin();

while (iter != prev(ptrClientRecord.end()))

{

iter2 = next(iter);

while (iter2 != ptrClientRecord.end())

{

if ((\*iter)->getYear() >= (\*iter2)->getYear())

{

swap(\*iter, \*iter2);

}

\*iter2++;

}

\*iter++;

}

iter = ptrClientRecord.begin();

while (iter != prev(ptrClientRecord.end()))

{

iter2 = next(iter);

while (iter2 != ptrClientRecord.end())

{

if ((\*iter)->getMonth() >= (\*iter2)->getMonth() && (\*iter)->getYear() == (\*iter2)->getYear())

{

swap(\*iter, \*iter2);

}

\*iter2++;

}

\*iter++;

}

iter = ptrClientRecord.begin();

while (iter != prev(ptrClientRecord.end()))

{

iter2 = next(iter);

while (iter2 != ptrClientRecord.end())

{

if ((\*iter)->getDay() >= (\*iter2)->getDay() && (\*iter)->getMonth() == (\*iter2)->getMonth() && (\*iter)->getYear() == (\*iter2)->getYear())

{

swap(\*iter, \*iter2);

}

\*iter2++;

}

\*iter++;

}

iter = ptrClientRecord.begin();

while (iter != prev(ptrClientRecord.end()))

{

iter2 = next(iter);

while (iter2 != ptrClientRecord.end())

{

if ((\*iter)->getHour() >= (\*iter2)->getHour() && (\*iter)->getDay() == (\*iter2)->getDay() && (\*iter)->getMonth() == (\*iter2)->getMonth() && (\*iter)->getYear() == (\*iter2)->getYear())

{

swap(\*iter, \*iter2);

}

\*iter2++;

}

\*iter++;

}

iter = ptrClientRecord.begin();

while (iter != prev(ptrClientRecord.end()))

{

iter2 = next(iter);

while (iter2 != ptrClientRecord.end())

{

if ((\*iter)->getMinute() >= (\*iter2)->getMinute() && (\*iter)->getHour() == (\*iter2)->getHour() && (\*iter)->getDay() == (\*iter2)->getDay() && (\*iter)->getMonth() == (\*iter2)->getMonth() && (\*iter)->getYear() == (\*iter2)->getYear())

{

swap(\*iter, \*iter2);

}

\*iter2++;

}

\*iter++;

}

iter = ptrClientRecord.begin();

while (iter != ptrClientRecord.end())

{

cout << left << setw(10) << (\*iter)->getClientFirstName() << "| |" << setw(14) << (\*iter)->getClientSecondName() << "| |" << setw(11) << (\*iter)->getCarBrand() << "| |" << setw(12) <<(\*iter)->getCarModel() << "| |" << setw(14) << (\*iter)->getServiceName() << "| |";

cout << left << setw(1) <<(\*iter)->getYear() << "-";

if ((\*iter)->getMonth()<10)

{

cout << left << "0" << setw(1) << (\*iter)->getMonth() << "-";

}

else

{

cout << left << setw(1) << (\*iter)->getMonth() <<"-";

}

if ((\*iter)->getDay() < 10)

{

cout << left << "0" << setw(1) << (\*iter)->getDay() <<" ";

}

else

{

cout << left << setw(1) << (\*iter)->getDay() <<" ";

}

if ((\*iter)->getHour() < 10)

{

cout << left << "0" << setw(1) << (\*iter)->getHour() <<":";

}

else

{

cout << left << setw(1) << (\*iter)->getHour() << ":";

}

if ((\*iter)->getMinute() < 10)

{

cout << left << "0" << setw(5) << (\*iter)->getMinute() << "| |";

}

else

{

cout << left << setw(6) << (\*iter)->getMinute() << "| |";

}

cout << left << (\*iter)->getServicePrice() << endl;

\*iter++;

}

}

}

void TimeTable::SaveFile()

{

ofstream out;

out.open(".\\TimeTable.txt"); //for .exe

//out.open("../TimeTable.txt");

if (out.is\_open())

{

if (ptrClientRecord.empty())

cout << "Нет записей\n" << endl;

else

{

iter = ptrClientRecord.begin();

while (iter != ptrClientRecord.end())

{

out << (\*iter)->getClientFirstName() << ";" << (\*iter)->getClientSecondName() << ";" << (\*iter)->getCarBrand() << ";" << (\*iter)->getCarModel() << ";"

<< (\*iter)->getServiceName() << ";" << (\*iter)->getYear() << ";" << (\*iter)->getMonth() << ";" << (\*iter)->getDay() << ";" << (\*iter)->getHour() << ";"

<< (\*iter)->getMinute() << ";" << (\*iter)->getServicePrice() << ";" << endl;

\*iter++;

}

}

}

}

void TimeTable::LoadFile()

{

string line;

ifstream in (".\\TimeTable.txt"); //for .exe

//ifstream in("../TimeTable.txt");

if (in.is\_open())

{

while (getline(in, line))

{

string s1[11];

int j = 0;

for (int i = 0; i <= 10; i++)

{

while (line[j] != ';')

{

s1[i] = s1[i] + line[j];

j++;

}

j++;

}

ClFirstName = s1[0];

ClSecondName = s1[1];

CBrand = s1[2];

CModel = s1[3];

SerName = s1[4];

year = atoi(s1[5].c\_str());

month = atoi(s1[6].c\_str());

day = atoi(s1[7].c\_str());

hour = atoi(s1[8].c\_str());

minute = atoi(s1[9].c\_str());

SerPrice = stof(s1[10]);

ClientRecord\* ptrClientRecord = new ClientRecord(ClFirstName, ClSecondName, CBrand, CModel, SerName, year, month, day, hour, minute, SerPrice);

InsertClient(ptrClientRecord);

}

}

in.close();

}

void ExpensesTable::LoadExpenses()

{

string line;

ifstream in(".\\ExpensesTable.txt"); //for .exe

//ifstream in("../ExpensesTable.txt");

if (in.is\_open())

{

while (getline(in, line))

{

string s1[5];

int j = 0;

for (int i = 0; i <= 4; i++)

{

while (line[j] != ';')

{

s1[i] = s1[i] + line[j];

j++;

}

j++;

}

Product = s1[0];

year = atoi(s1[1].c\_str());

month = atoi(s1[2].c\_str());

day = atoi(s1[3].c\_str());

Cost = stof(s1[4]);

Expenses\* ptrExpenses = new Expenses(Product, year, month, day, Cost);

insertExpenses(ptrExpenses);

}

}

in.close();

}

void ExpensesTable::SaveExpenses()

{

ofstream out;

out.open(".\\ExpensesTable.txt"); //for .exe

//out.open("../ExpensesTable.txt");

if (out.is\_open())

{

if (vecptrExpenses.empty())

cout << "Нет расходов\n" << endl;

else

{

iter = vecptrExpenses.begin();

while (iter != vecptrExpenses.end())

{

out << (\*iter)->Product << ";" << (\*iter)->year << ";" << (\*iter)->month << ";" << (\*iter)->day << ";"<< (\*iter)->Cost << ";" << endl;

\*iter++;

}

}

}

}

void TimeTable::InsertClient(ClientRecord\* ptrCR)

{

ptrClientRecord.push\_back(ptrCR);

}

void ExpensesTable::insertExpenses(Expenses\* ptrExp)

{

vecptrExpenses.push\_back(ptrExp);

}

void ExpensesTable::ShowExpensesTable()

{

if (vecptrExpenses.size() == 0)

cout << "Расходов нет\n" << endl;

else

{

iter = vecptrExpenses.begin();

while (iter != vecptrExpenses.end())

{

cout << left << setw(9) << (\*iter)->Product << " | | " << (\*iter)->year << "-" << (\*iter)->month << "-" << (\*iter)->day << " | | " << (\*iter)->Cost << endl;

iter++;

}

cout << endl;

}

}

AddExpensesScreen::AddExpensesScreen(ExpensesTable\* ExpTab) : ptrExpensesTable(ExpTab)

{

}

void AddExpensesScreen::setExpenses()

{

cout << "Введите что израсходовали: ";

cin >> Product;

cout << "Введите дату расхода(YYYY MM DD): ";

cin >> year;

cin >> month;

cin >> day;

cout << "Введите стоимость расхода: ";

cin >> Cost;

Expenses\* ptrExpenses = new Expenses(Product, year, month, day, Cost);

ptrExpensesTable->insertExpenses(ptrExpenses);

}

EditDeleteScreen::~EditDeleteScreen()

{

}

void EditClientScreen::EditInfo(unsigned int YY, unsigned int MM, unsigned int DD, unsigned int hh, unsigned int mm)

{

bool search = true;

int choice;

if (ptrTimeTable->ptrClientRecord.empty())

cout << "Нет записей" << endl;

else

{

ptrTimeTable->iter = ptrTimeTable->ptrClientRecord.begin();

while (search == true && ptrTimeTable->iter != ptrTimeTable->ptrClientRecord.end())

{

if (YY == (\*ptrTimeTable->iter)->getYear() && MM == (\*ptrTimeTable->iter)->getMonth() && DD == (\*ptrTimeTable->iter)->getDay() && hh == (\*ptrTimeTable->iter)->getHour() && mm == (\*ptrTimeTable->iter)->getMinute())

{

search = false;

}

else

{

\*ptrTimeTable->iter++;

}

}

ClFirstName = (\*ptrTimeTable->iter)->getClientFirstName();

ClSecondName = (\*ptrTimeTable->iter)->getClientSecondName();

CBrand = (\*ptrTimeTable->iter)->getCarBrand();

CModel = (\*ptrTimeTable->iter)->getCarModel();

SerName = (\*ptrTimeTable->iter)->getServiceName();

year = (\*ptrTimeTable->iter)->getYear();

month = (\*ptrTimeTable->iter)->getMonth();

day = (\*ptrTimeTable->iter)->getDay();

hour = (\*ptrTimeTable->iter)->getHour();

minute = (\*ptrTimeTable->iter)->getMinute();

SerPrice = (\*ptrTimeTable->iter)->getServicePrice();

delete\* ptrTimeTable->iter;

ptrTimeTable->iter = ptrTimeTable->ptrClientRecord.erase(ptrTimeTable->iter);

while (true)

{

cout << "Выберите данные для изменения:" << endl;

cout << "1. Имя:" << ClFirstName << endl;

cout << "2. Фамилия:" << ClSecondName << endl;

cout << "3. Марка машины:" << CBrand << endl;

cout << "4. Модель машины:" << CModel << endl;

cout << "5. Услуга:" << SerName << endl;

cout << "6. Дата исполнения услуги:" << year << "-" << month << "-" << day << " " << hour << ":" << minute << endl;

cout << "7. Сохранить и выйти" << endl;

cout << "Введите цифру нужного пункта:" << endl;

cin >> choice;

system("cls");

switch (choice)

{

case 1:

{

cout << "Введите Имя:" << endl;

cin >> ClFirstName;

break;

}

case 2:

{

cout << "Введите Фамилию:" << endl;

cin >> ClSecondName;

break;

}

case 3:

{

cout << "Введите марку машины:" << endl;

cin >> CBrand;

break;

}

case 4:

{

cout << "Введите модель машины:" << endl;

cin >> CModel;

break;

}

case 5:

{

PriceList PList;

PList.ShowPrices();

cout << "Введите Номер услуги: " << endl;

cin >> SerNum;

ifstream in (".\\PriceList.txt"); //for .exe

//ifstream in("../PriceList.txt");

if (in.is\_open())

{

while (getline(in, line))

{

if (line.find(to\_string(SerNum)) != string::npos) {

SerName = line.substr(3, 13);

SerPrice = stof(line.substr(16, 4));

}

}

}

in.close();

break;

}

case 6:

{

cout << "Введите дату выполнения услуги(YYYY MM DD hh mm):" << endl;

bool search = true;

bool write;

while (search)

{

cin >> YY;

cin >> MM;

cin >> DD;

cin >> hh;

cin >> mm;

write = true;

if (ptrTimeTable->ptrClientRecord.empty())

//запись даты, так как нет записей

search = false;

else

{

ptrTimeTable->iter = ptrTimeTable->ptrClientRecord.begin();

while (ptrTimeTable->iter != ptrTimeTable->ptrClientRecord.end() && write)

{

if (YY == (\*ptrTimeTable->iter)->getYear() && MM == (\*ptrTimeTable->iter)->getMonth() && DD == (\*ptrTimeTable->iter)->getDay() && hh == (\*ptrTimeTable->iter)->getHour() && mm == (\*ptrTimeTable->iter)->getMinute())

{

cout << "Запись на эту дату уже существует." << endl;

write = false;

}

\*ptrTimeTable->iter++;

}

if (write)

{

//запись даты, так как нет совпадений

search = false;

}

}

}

break;

}

case 7:

{

ClientRecord\* ptrClientRecord = new ClientRecord(ClFirstName, ClSecondName, CBrand, CModel, SerName, year, month, day, hour, minute, SerPrice);

ptrTimeTable->InsertClient(ptrClientRecord);

cout << "Данные успешно изменены" << endl;

return;

break;

}

default:

{

cout << "Нет такого пункта" << endl;

system("pause");

break;

}

}

system("cls");

}

}

}

void DeleteClientScreen::DeleteClient(unsigned int YY, unsigned int MM, unsigned int DD, unsigned int hh, unsigned int mm)

{

char conf;

if (ptrTimeTable->ptrClientRecord.empty())

cout << "Нет записей" << endl;

else

{

ptrTimeTable->iter = ptrTimeTable->ptrClientRecord.begin();

while (ptrTimeTable->iter != ptrTimeTable->ptrClientRecord.end())

{

if (YY == (\*ptrTimeTable->iter)->getYear() && MM == (\*ptrTimeTable->iter)->getMonth() && DD == (\*ptrTimeTable->iter)->getDay() && hh == (\*ptrTimeTable->iter)->getHour() && mm == (\*ptrTimeTable->iter)->getMinute())

{

cout << "Уверены ли вы в удалении данной записи?(y/n)" << endl;

cin >> conf;

if (conf=='y')

{

delete\* ptrTimeTable->iter;

ptrTimeTable->iter = ptrTimeTable->ptrClientRecord.erase(ptrTimeTable->iter);

cout << "Запись успешно удалена" << endl;

}

}

\*ptrTimeTable->iter++;

}

}

}

void EditDeleteScreen::getRecordDate()

{

cout << "Введите дату записи, которую хотите редактировать(YYYY MM DD hh mm):" << endl;

cin >> year;

cin >> month;

cin >> day;

cin >> hour;

cin >> minute;

cout << "Изменить - 1\nУдалить - 2\nВыбор:" << endl;

cin >> choice;

system("cls");

switch (choice)

{

case 1:

{

ptrEditClientScreen = new EditClientScreen(ptrTimeTable);

ptrEditClientScreen->EditInfo(year, month, day, hour, minute);

delete ptrEditClientScreen;

system("Pause");

break;

}

case 2:

{

ptrDeleteClientScreen = new DeleteClientScreen(ptrTimeTable);

ptrDeleteClientScreen->DeleteClient(year, month, day, hour, minute);

delete ptrDeleteClientScreen;

system("Pause");

break;

}

default:

cout << "Нет такого пункта" << endl;

system("pause");

break;

}

}

void Report::ShowReport()

{

cout << "Введите начальную дату (YYYY MM DD): " << endl;

cin >> year1;

cin >> month1;

cin >> day1;

cout << "Введите конечную дату (YYYY MM DD): " << endl;

cin >> year2;

cin >> month2;

cin >> day2;

Revenue = 0;

ptrTimeTable->iter = ptrTimeTable->ptrClientRecord.begin();

while (ptrTimeTable->iter != ptrTimeTable->ptrClientRecord.end())

{

if (year1 == (\*ptrTimeTable->iter)->getYear())

{

if (month1 == (\*ptrTimeTable->iter)->getMonth())

{

if (day1 <= (\*ptrTimeTable->iter)->getDay())

{

Revenue += (\*ptrTimeTable->iter)->getServicePrice();

}

}

else if (month1 < (\*ptrTimeTable->iter)->getMonth())

{

Revenue += (\*ptrTimeTable->iter)->getServicePrice();

}

}

else if (year1 < (\*ptrTimeTable->iter)->getYear() && year2 >(\*ptrTimeTable->iter)->getYear())

{

Revenue += (\*ptrTimeTable->iter)->getServicePrice();

}

else if (year2 == (\*ptrTimeTable->iter)->getYear())

{

if (month2 == (\*ptrTimeTable->iter)->getMonth())

{

if (day2 >= (\*ptrTimeTable->iter)->getDay())

{

Revenue += (\*ptrTimeTable->iter)->getServicePrice();

}

}

else if (month2 > (\*ptrTimeTable->iter)->getMonth())

{

Revenue += (\*ptrTimeTable->iter)->getServicePrice();

}

}

\*ptrTimeTable->iter++;

}

Expenses = 0;

ptrExpensesTable->iter = ptrExpensesTable->vecptrExpenses.begin();

while (ptrExpensesTable->iter != ptrExpensesTable->vecptrExpenses.end())

{

if (year1 == (\*ptrExpensesTable->iter)->year)

{

if (month1 == (\*ptrExpensesTable->iter)->month)

{

if (day1 <= (\*ptrExpensesTable->iter)->day)

{

Expenses += (\*ptrExpensesTable->iter)->Cost;

}

}

else if (month1 < (\*ptrExpensesTable->iter)->month)

{

Expenses += (\*ptrExpensesTable->iter)->Cost;

}

}

else if (year1 < (\*ptrExpensesTable->iter)->year && year2 >(\*ptrExpensesTable->iter)->year)

{

Expenses += (\*ptrExpensesTable->iter)->Cost;

}

else if (year2 == (\*ptrExpensesTable->iter)->year)

{

if (month2 == (\*ptrExpensesTable->iter)->month)

{

if (day2 >= (\*ptrExpensesTable->iter)->day)

{

Expenses += (\*ptrExpensesTable->iter)->Cost;

}

}

else if (month2 > (\*ptrExpensesTable->iter)->month)

{

Expenses += (\*ptrExpensesTable->iter)->Cost;

}

}

\*ptrExpensesTable->iter++;

}

Profit = Revenue - Expenses;

cout << "Доходы за указанный период: " << Revenue << endl;

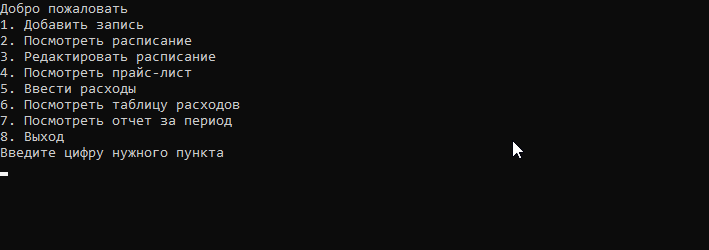
cout << "Расходы за указанный период: " << Expenses << endl;

cout << "Выручка за указанный период: " << Profit << endl;

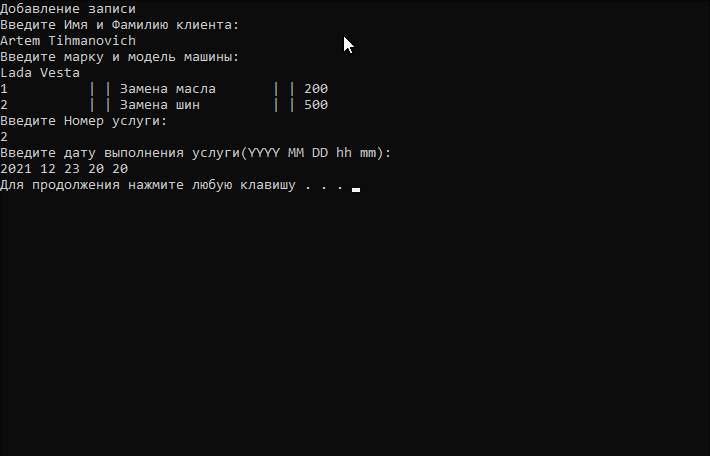
}

1. **Руководство пользователя.**

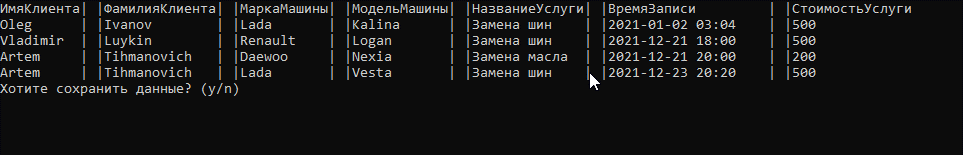
При запуске программы менеджер видит меню:



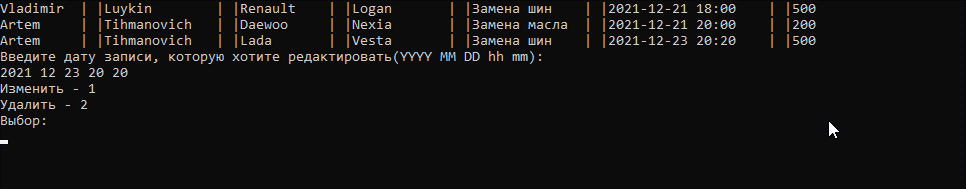
Далее менеджеру нужно ввести цифру нужного для него пункта. При выборе Добавления записи пользователя переносит в Экран добавления записи, где пользователь вводит информацию о клиенте:



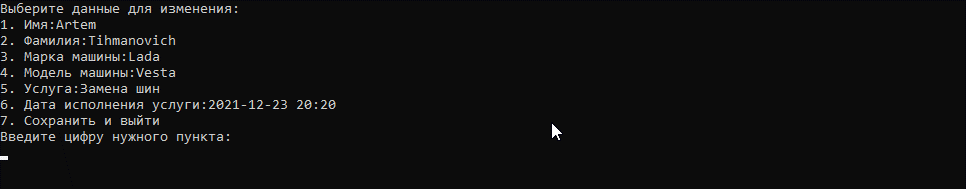
При выборе второго варианта «Посмотреть расписание» менеджеру показывается расписание, в котором записи отсортированы по дате и времени:



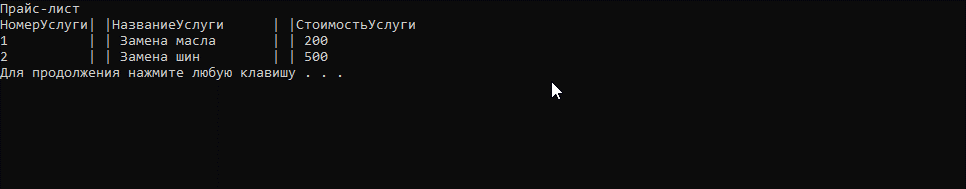
Также пользователю предлагается сохранить расписание в отдельный txt файл. При выборе третьего варианта «Редактировать расписание» менеджеру показывается расписание, после которого менеджер вводит дату и время записи которую хочет изменить или удалить и выбирает нужное ему действие (изменение или удаление):



При выборе удаления пользователю задается вопрос об уверенности удалении и если пользователь уверен, то запись удалится. При выборе изменения пользователя перекидывает в окно изменения записи, где он выбирает что именно нужно изменить:



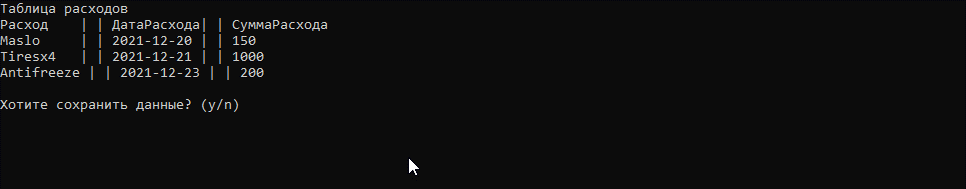
При выборе четвертого варианта «Посмотреть прайс-лист» пользователю выводится прайс-лист с названием услуги и ее стоимостью:



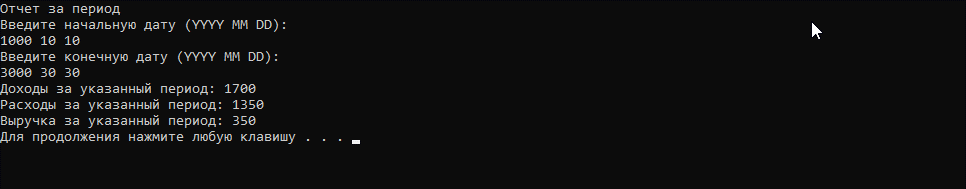
При выборе пятого варианта «Ввести расходы» пользователя перебрасывает в окно Ввода расходов, где он вводит что именно израсходовали, дату, когда закупали нужный товар и стоимость расходованного товара:



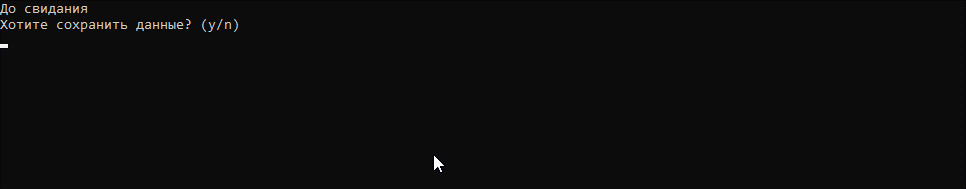
При выборе шестого варианта «Посмотреть таблицу расходов» » менеджеру показывается вся таблица расходов:



При выборе седьмого варианта «Посмотреть отчет за период» » менеджеру предлагается ввести начальную и конечную дату для создания период и за это период показываются доходы, расходы и выручка:

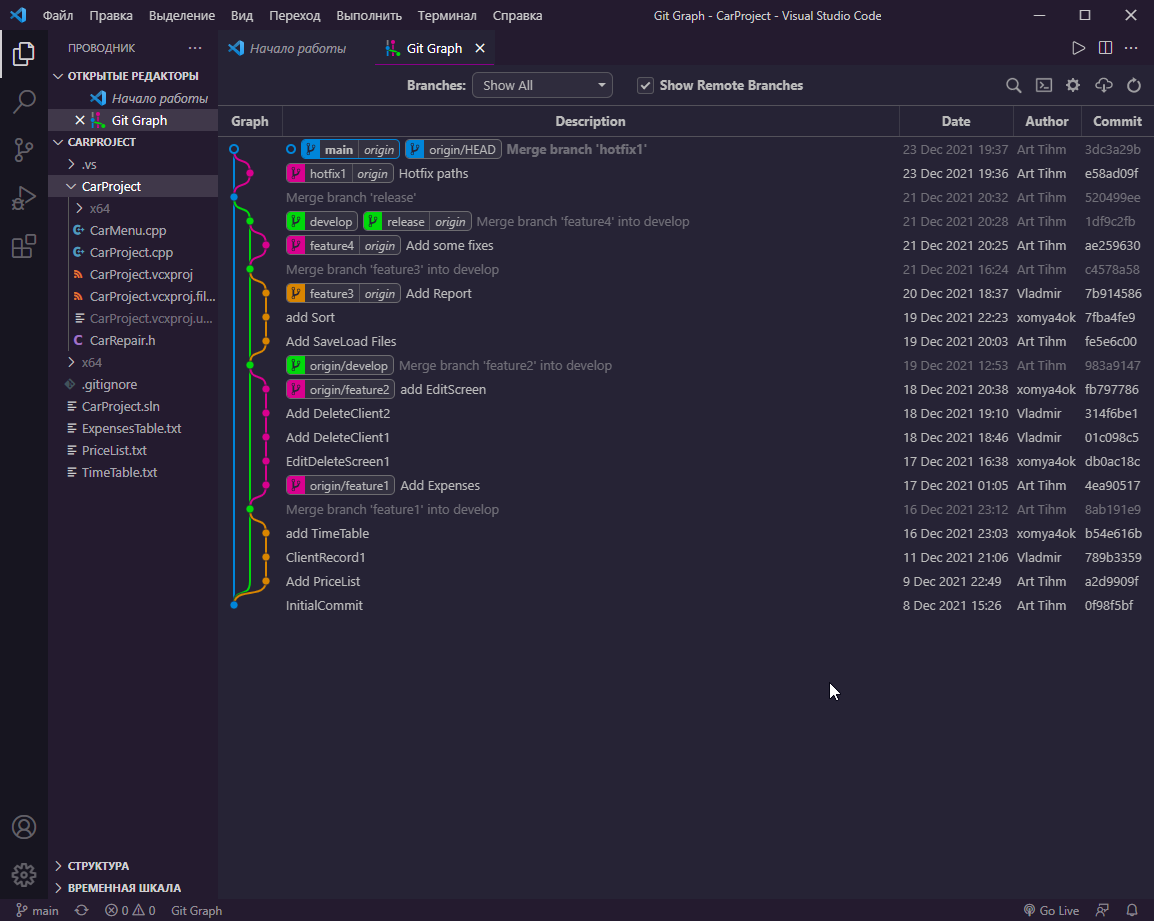


При выходе из программы пользователя еще раз спрашивается хочет ли он сохранить введенные им данные:



**История проекта на GitHub**

Адрес проекта:<https://github.com/Tihman/CarProject>



InitialCommit (Ветка main): Создание скелета программы.

Add PriceList (Ветка feature1): Добавление класса таблицы прайс-листа.

ClientRecord1 (Ветка feature1): Добавление класса записи клиента.

add TimeTable (Ветка feature1): Добавление класса таблицы расписания.

Add Expenses (Ветка feature2): Добавление класса таблицы расходов.

EditDeleteScreen (Ветка feature2): Добавление экрана редактирования записей.

Add DeleteClient1 (Ветка feature2): Добавление класса удаления записи клиента.

Add DeleteClient2 (Ветка feature2): Доработка класса удаления записи клиента и устранение недочетов.

add EditScreen (Ветка feature2): Добавление класса изменения данных записи клиента.

Add SaveLoad Files (Ветка feature3): Добавление классов сохранения и загрузки таблицы расписания.

add Sort (Ветка feature3): Добавление сортировки таблицы расписания.

Add Report (Ветка feature3): Добавление класса отчета по доходам и расходам.

Add some fixes (Ветка feature4): Устранение недочетов.

Hotfix paths (Ветка hotfix1): Исправление путей к рабочим файлам.

**Заключение:**

При реализации нашего проекта мы использовали среду разработки Microsoft Visual Studio 2019 и язык программирования C++. Для того чтобы обезопасить себя от ошибок и утери данных мы использовали систему контроля версий Git и выгружали наши действия на веб-сервис GitHub. Для упрощения написания кода были нарисованы диаграммы вариантов использования, диаграммы деятельности, диаграмма классов и диаграммы последовательностей. Все они приложены выше. В результате разработки были реализованы все прецеденты. Для оптимизации работы программы мы применяли принципы раздельной компиляции, а также реализовали очистку памяти. Программа работает без сбоев и зависаний. В итоге программа обладает всем функционалом, указанным заказчиком, из чего можно сделать вывод, что цель проекта была достигнута.