Министерство высшего образования и науки Российской Федерации

федеральное государственное бюджетное образовательное учреждение

высшего профессионального образования

«ПЕТРОЗАВОДСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Кафедра физики твердого тела

**Барбершоп «Цирюльня»**

Командное задание

по дисциплине

«Технология программирования»

Выполнили: студенты 3 курса

физико-технического института, группа 21318

Дмитриев О.О.  
Афонин А.А.  
Анкудинов Д.П.

Тимурзиев А.М.

Проверил: Бульба Артем Владимирович.

Петрозаводск

1. г.

**Цель работы.**

Разработать программное обеспечение (проект), работая в команде, используя систему контроля версий (GitHub), моделирование унифицированных диаграмм (UML)

**Задачи:**

1. Познакомиться с принципами работы в команде.
2. Научится использовать при работе в команде систему контроля версии Git.
3. Научиться строить UML-диаграммы.

**Требования к проекту:**

1. Программа не должна вызывать системных сбоев или зависать.
2. Программа должна освобождать всю выделенную память.
3. Должны отсутствовать конструкции, приводящие к неопределённому поведению.
4. Переменные должны быть, по возможности, локальны.
5. Не должно быть неиспользуемых переменных.
6. Алгоритмы не должны быть избыточны
7. Свои классы необходимо реализовывать в отдельных файлах.
8. Необходимо отделять интерфейс от реализации.
9. Необходимо использовать STL-контейнеры и итераторы.
10. Позволять заносить клиентов и сотрудников в базу данных.

**Программная реализация**

Среда разработки – Microsoft Visual Studio 2019  
Язык разработки – C++  
Заголовочные файлы (.h):

* User – является базовым классом для Employee, MockDatabase, Client, Admin.
* status – единица компиляции, содержит перечисление для типа пользователя и статус клиента.
* speciality - единица компиляции, содержит перечисление должностей и значения по умолчанию.
* ServiceOrder – класс, предоставляющий объект «заказ» и методы по работе с ним.
* Service – класс, предоставляющий объект «услуга» и методы по работе с ним.
* MockDatabase – класс, эмулирующий базу данных и запросы к ней.
* mainForm – класс, предоставляющий winform. Отвечает за графический интерфейс.
* menuForm – класс, предоставляющий winform. Отвечает за графический интерфейс.
* Employee – класс наследник от User. Предоставляет объект «рабочий» и методы по работе с ним.
* Client– класс наследник от User. Предоставляет объект «клиент» и методы по работе с ним.
* checkFunctions – заголовочный файл, содержит объявления ряда необходимых для работы функций.
* Admin – класс наследник от User. Предоставляет объект «Администратор» и методы по работе с ним.
* ServiceOrder – класс предоставляет объект «заказ» и методы по работе с ним.

**Краткое словесное описание сюжета.**

Имеется условный барбершоп. Барбершоп находится рядом с центром города, имеет узнаваемость и статус. Его клиентское поле самое большое в городе. Поток посетителей высок и стандартные программные решения их не устраивают, из-за своей широкой направленности. Универсальность программы делает её неудобной и сложной для использования. Поэтому барбершоп «ЦИРЮЛЬНЯ ВИТЯЗЪ» обратился к нам с заказом на разработку специализированного программного обеспечения.

***Заказчик установил следующее техническое задание:***

Программа должна:

* Позволять заносить клиентов и сотрудников в базу данных,
* Предоставлять доступ к просмотру текущих сотрудников и клиентов,
* Формировать расписание исходя из записей на услуги,
* Формировать запись на услугу / отменять запись,
* Формировать финансовую отчётность (Затраты, выручка, прибыль) за год, за сутки, за месяц.

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

***Предоставляемые услуги:***Ниже перечислены все возможные услуги, предоставляемые барбершопом.

|  |  |
| --- | --- |
| **Услуга** | **Цена** |
| **Быстрая стрижка** | **500** |
| **Мужская стрижка** | **900** |
| **Удлиненная стрижка** | **1200** |
| **Детская стрижка** | **600** |
| **Удлиненная детская стрижка** | **800** |
| **Камуфляж седины** | **1000** |
| **Бритье головы** | **1000** |
| **Моделирование бороды** | **500** |
| **Моделирование с опасной бритвой** | **700** |
| **Бритье с распариванием** | **800** |
| **Укладка** | **300** |
| **Стрижка усов** | **300** |
| **Патчи для глаз** | **100** |
| **Мытье головы** | **100** |
| **Моделирование с опасной бритвой** | **700** |
| **Бритье с распариванием** | **800** |

**Статусы**Предопределенные статусы клиента.

|  |  |
| --- | --- |
| **Статус** | **Скидка** |
| **Постоянный клиент** | **10%** |
| **Разовый клиент** | **0%** |
| **Новый клиент** | **15%** |
| **VIP клиент** | **25%** |

Правила получения статуса:

* **Новый клиент – первое посещение**
* **Разовый клиент – 2, 3 посещение**
* **Постоянный клиент – с 4 посещений**
* **VIP клиент – после 10 посещений**

***Финансовая отчётность***

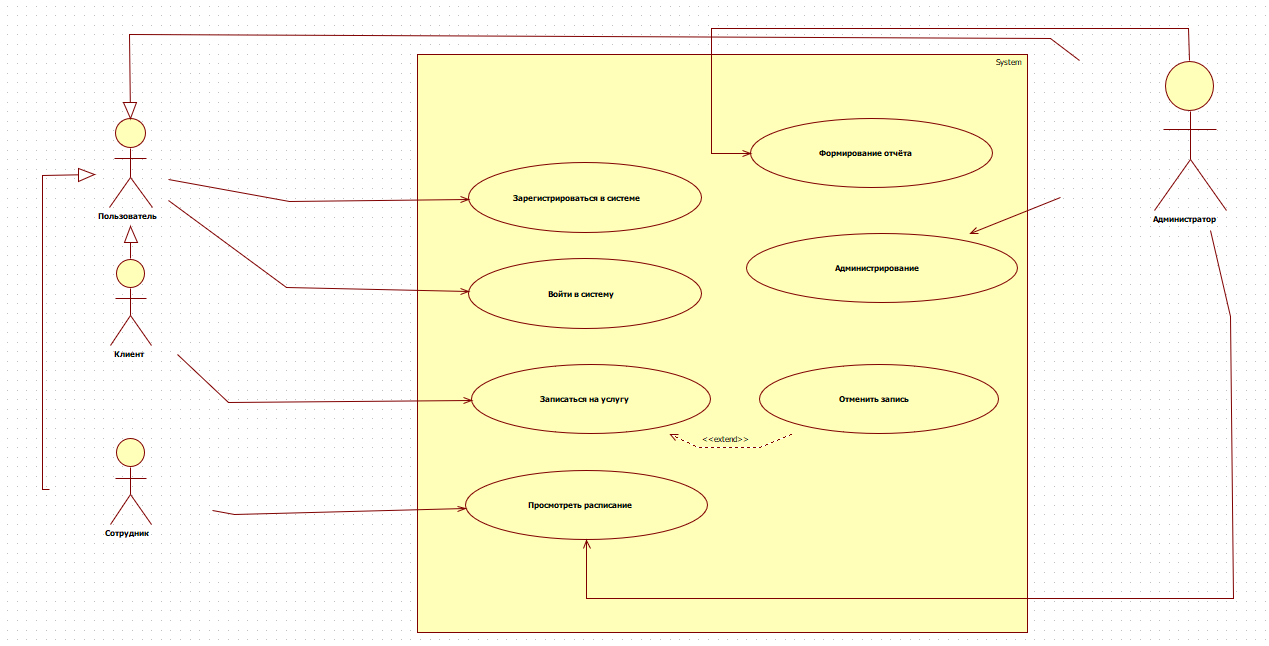
Финансовая информация об организации  
  
Возможные отчёты:

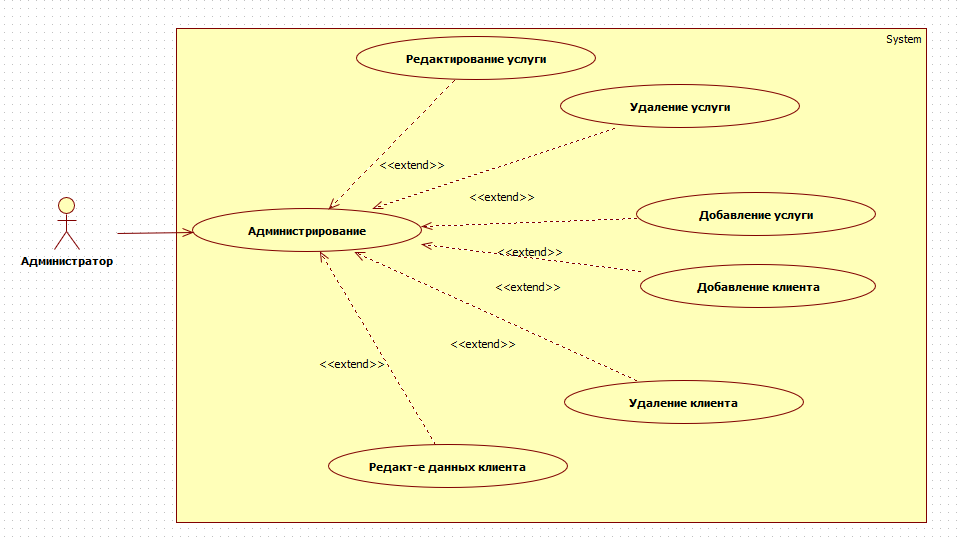
* Отчёт за год, за месяц, за день:
  1. Затраты
  2. Выручка
  3. Прибыль = выручка – затраты
  4. Количество клиентов
  5. Средний чек за выбранный период времени (за год, за месяц, за день).

**Необходимо перечислить действующих субъектов (actors) и их функциональные обязанности.**

* Пользователь
* Регистрация в системе
* Вход в систему
* Клиент
* Регистрация в системе
* Войти в систему
* Запись на услугу
* Отменить запись
* Сотрудник
* Просмотр расписания
* Регистрация в системе
* Вход в систему
* Администратор
* Формирование отчёта
* Администрирование услуг
* CRUD данных о клиентах/сотрудниках
* Просмотр расписания
* Регистрация в системе
* Вход в систему

**Диаграммы вариантов использования.**

****

****

**Сценарии вариантов использования.**

**Зарегистрироваться в системе -** данный вариант использования описывает регистрацию пользователя в системе.

Здесь можно выбрать под какой ролью будет зарегистрирован пользователь. Так же водится имя, пол, дата рождения, номер телефона, e-mail, пароль, должность и тип пользователя.

**Войти в систему** - данный вариант использования описывает вход пользователя в систему барбершоп. Для входа нужно ввести e-mail и пароль. При входе определяется какой тип пользователя вошел и в зависимости от типа открывается нужная панель и набор функций.

**Записаться на услугу** - данный вариант использования описывает запись на услугу в барбершоп. Для записи нужно выбрать дату и время, название услуги, барбера, который будет делать стрижку.

**Просмотреть расписание** - данный вариант использования описывает показ расписания. Просматривать расписание может только сотрудник(барбер), либо администратор. В расписание показывается номер заказа, тип услуги, клиент, стоимость, статус заказа и дата.

**Формирование отчётов** - данный вариант использования

описывает средство для организации

загрузки итоговой информации в текстовый документ, хранящийся

в системе пользователя. Только администратор имеет возможность создавать

отчёт. Отчёт формируется в периоде за год, месяц и день. Отчёт создаётся

по параметрам, которые выбирает пользователь.

**Администрирование услуг** - данный вариант использования

описывает методы, которые используются для

административного управления. Администрирование включает в себя шесть

основных операций: добавление, изменение пользователя, удаление,

добавление, изменение услуги и для составления отчёта.

**Отменить запись** - данный вариант использования описывает

отмену бронирования клиентом на записанную услугу. Отмена записи

производится в одностороннем порядке со стороны клиента.

**Диаграммы действий для вариантов использования.**

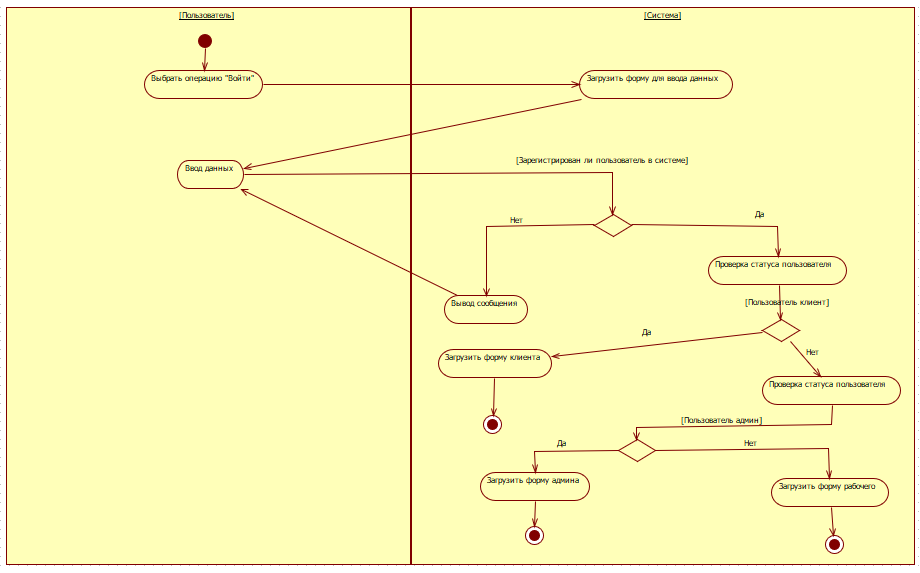
Диаграмма действий «вход в систему» 

Диаграмма действий «регистрация в систему»

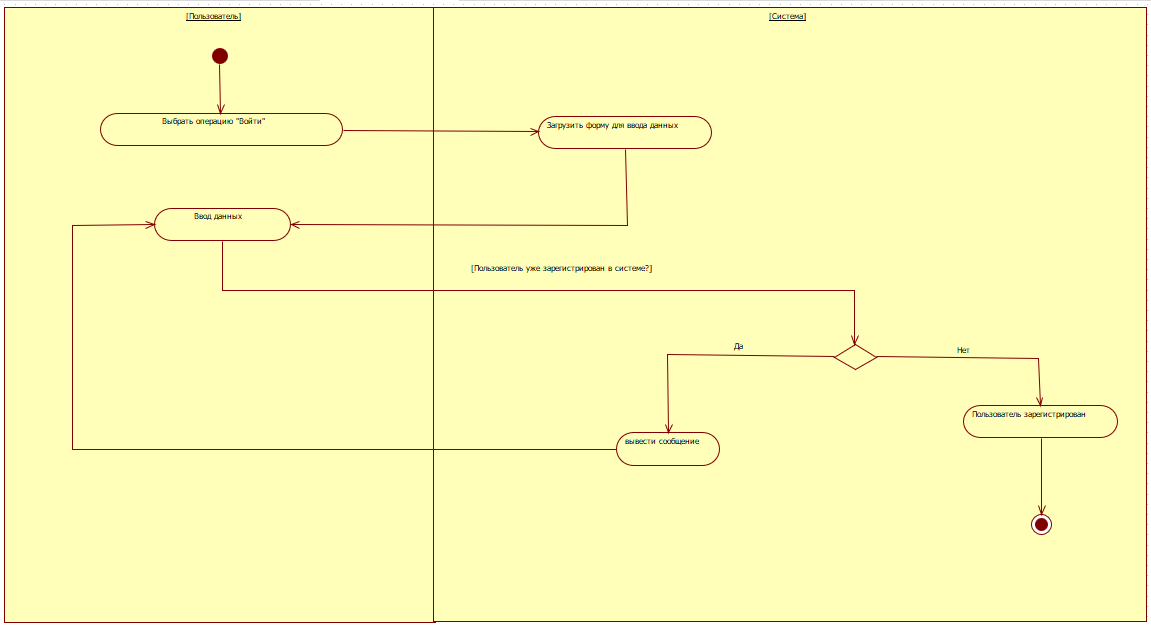
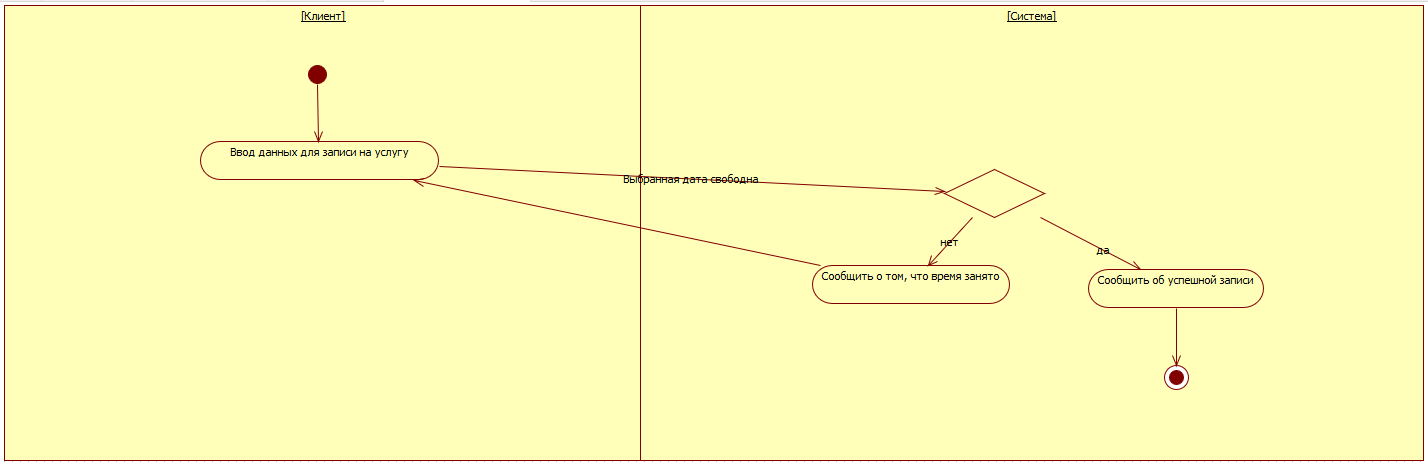
****

Диаграмма действий «запись на услугу»

****

**Предварительный список существительных.**

* Клиент (Client.h)
* Сотрудник (Employee.h)
* Пользователь (User.h)
* Администратор (Admin.h)

**Уточненный список существительных**

* Клиент (Client.h)
* Сотрудник (Employee.h)
* Пользователь (User.h)
* Администратор (Admin.h)
* Услуги (Service.h)
* База данных (MockDatabase.h)
* Обслуживание заказа (ServiceOrder.h)

**Список атрибутов к классам.**

**Класс User.h**

**Поля:**

* int id – идентификатор пользователя
* bool sex – логическая переменная для выбора пола
* userType status – тип пользователя
* string name – ФИО пользователя
* string date – Дата рождения пользователя
* string phonenumber – номер телефона пользователя
* string email – электронная почта пользователя
* size\_t password – пароль, установленный пользователем

**Методы:**

* int getID () – возвращает идентификатор
* string getName () – возвращает имя пользователя
* string getSex () – возвращает пол пользователя
* string getPhonenumber () – возвращает номера телефона
* string getEmail () – возвращает электронную почту
* string getStatusString () – возвращает тип пользователя
* userType getStatus () – возвращает тип пользователя
* string getDate () – возвращает дату рождения
* size\_t getPassword () – возвращает пароль

**Класс Client.h**

**Поля:**

* int numberOfVisits – количество посещений
* clientStatusType clientStatus – статус клиента

**Методы:**

* string getClientStatus () – получение статуса клиента
* int getNumberOfVisits () – получение кол-ва посещений
* void setClientStatus (clientStatusType clientStatus) – установление статуса клиента
* virtual string getInfo () – получение информации о клиенте

**Класс Employee.h**

**Поля:**

* float experience – стаж
* userSpeciality speciality – род деятельности
* string personalAchievements - личностные достижения
* float salary - базовый оклад
* float workingHours - количество рабочих часов

**Методы:**

* void setExperience (float experience) – установить стаж
* void setSpeciality (string speciality) – установить специальность
* string getSpeciality () – получить специальность
* void setPersonalAchievements (string personalAchievements) – установить личностные достижения
* float getExperience () – получение стажа работы
* string getPersonalAchievements () – получение инф-и о личностных достижениях
* float getSalary () – получение размера заработной платы
* float getWorkingHours () – получить количество рабочих часов
* virtual string getInfo () – получение информации о работнике

**Класс Admin.h**

**Методы:**

* void deleteSpecificUser (int id) – удалить определенного пользователя
* void deleteService (int id) – удаление определенной услуги
* void deleteOrder (int id) – удаление определенного заказа
* void changeService (int id) – изменить определенный заказ
* void changeSpecificUser (int id) – изменить определенного пользователя

**Класс Service.h**

**Поля:**

* int ID – идентификатор услуги
* string name – наименование услуги
* int price – стоимость услуги

**Методы:**

* int getID () – получить идентификатор услуги
* void setPrice (int price) – установить стоимость услуги
* void setName (string name) – установить наименование услуги
* int getPrice () – получить стоимость услуги
* string getInfo () – получить информацию об услуге
* string getName () – получить наименование услуги

**Класс ServiceOrder.h**

**Поля:**

* string date – назначеннная дата записи
* int ID – идентификатор заказа
* int serviceID – идентификатор услуги
* int employeeID – идентификатор сотрудника
* int clientID – идентификатор клиента
* float serviceCostWithDiscount – стоимость услуги со скидкой
* bool status – логическая переменная, показывающая статус выполнения заказа

**Методы:**

* int getID() – получить идентификатор заказа
* void setStatus(bool status) – установить статус выполнения заказа
* bool getStatus() – получить статус выполнения заказа
* float getServiceCostWithDiscount() – получить стоимость услуги со скидкой
* int getClientID() – получить идентификатор клиента
* int getEmployeeID() – получить идентификатор сотрудника
* string getDate() – получить назначенную дату заказа
* int getServiceID() – получить идентификатор услуги
* string getTime() – получить установленное время заказа
* double getCost() – получить стоимость заказа
* string getInfo() – получить информацию о заказе
* string getName() – получить имя клиента

**Класс MockDatabase.h**

**Методы:**

* void addUser (string name, bool sex, string date, string phonenumber, string email, string password, userType status, int numberOfVisits = DEF\_VISIT\_NUM) – добавить клиента или администратора
* void addUser (string name, bool sex, string date, string phonenumber, string email, string password, userType status, float experience, userSpeciality speciality, string personalAchievements = DEF\_PERSONAL\_ACHIEVEMENTS) – добавить сотрудника
* void loadUser (int ID, string name, bool sex, string date, string phonenumber, string email, string password, userType status, int numberOfVisits = DEF\_VISIT\_NUM) – загрузить клиента или администратора
* void loadUser (int ID, string name, bool sex, string date, string phonenumber, string email, string password, userType status, float experience, userSpeciality speciality, string personalAchievements) – загрузить сотрудника
* void loadService (int ID, string name, int price) – загрузить услугу
* void addService (string name, int price) – добавить услугу

**Список сообщений**

«Зарегистрироваться в системе»

**Записывает**: добавляет в конец списка listOfUsers новую запись

**Проверяет:** на уникальность e-mail. Вызывается метод emailIsUnique ()

После нажатия на кнопку зарегистрироваться проверяет есть ли такой e-mail в базе данных.

**Записываются данные**: текущая база данных записывается в файл. Вызывается метод writeDatabaseToFile (), writeServicesToFile (), writeServicesOrdersToFile ()

«Вход в систему»

**Просит ввести:** почту и пароль, находит пользователя по e-mail и паролю. findUser ()

**Получить статус:** обращение к методу класса MockDataBasegetUser (), чтобы найти пользователя по ID и узнать его статус getStatus ().

«Записаться на услугу»

**Записаться:** клиентдолжен выбрать услугу, сотрудника и дату/время**.** Вызов метода addServiceOrder ()

**Заполнить:** происходит обращение к базе данных, если количество заказов больше или равно восьми, тогда день занят getOrdersCount ()

**Добавить:** если день свободен, то добавляем в базу данных заказ.

**Обновить:** обновляем доступное время для записи updateAvailableTimes ()

**Документировать**: обращаемся к классу базы данных и с неё записываем в текстовый файл текущие заказы. writeServicesOrdersToFile ()

«Посмотреть расписание»

**Получить:** обращаемся к классу MockDataBase, и получаем список записей

«Формирование отчёта»

**Получить:** обращение к классу MockDataBase базы данных и получение списка заказов.

«Администрирование»

**Добавить:** получаем данные которые вводит пользователь и записываем их в экземпляр класса MockDataBase

**Удалить:** пользователь выбирает клиента или услугу, происходит обращение к экземпляру MockDataBase и удаляется элемент.

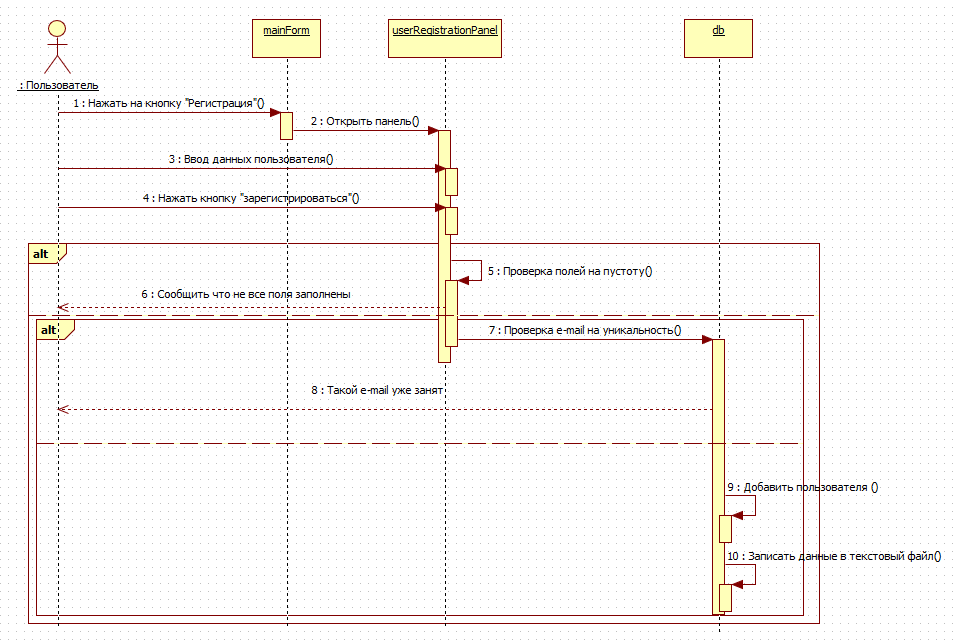
**Редактировать:** пользователь выбирает клиента или услугу, происходит обращение к экземпляру MockDataBase и пользователь редактирует данные.

**Диаграмма классов.**

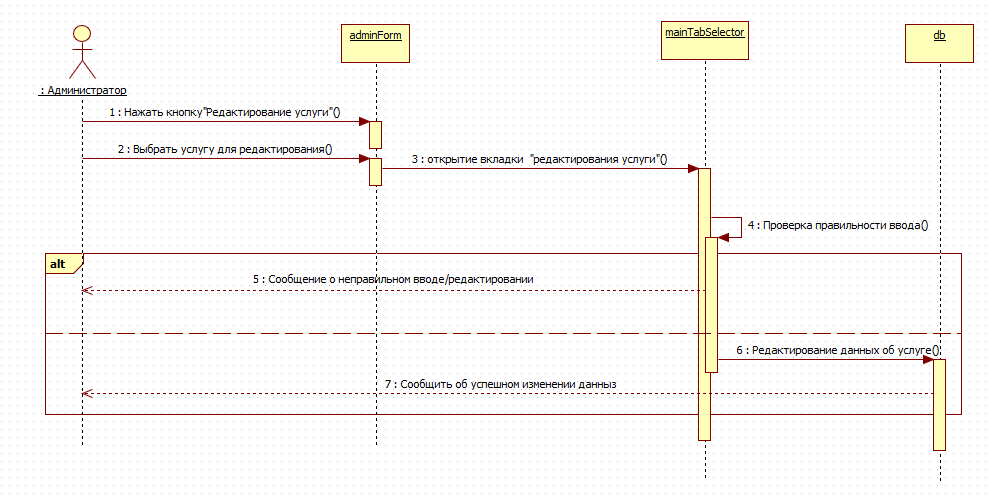
Приложение 1

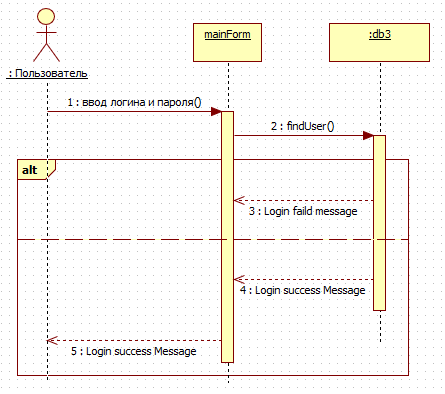
**Диаграммы последовательностей.**

Зарегистрироваться в системе.

****

Редактирование услуги.

****

Войти в систему**.  
**

**Код заголовочных файлов.**

**Service.h**

---------

#pragma once

#include "User.h"

class Service {

private:

int ID;

std::string name;

int price;

public:

Service() {};

Service(int ID, std::string name, int price);

int getID();

void setPrice(int price);

void setName(std::string name);

int getPrice();

std::string getInfo();

std::string getName();

};

---------

**ServiceOrder.h**

---------

#pragma once

#include "User.h"

class ServiceOrder{

private:

std::string date;

std::string time;

int ID;

int serviceID;

int employeeID;

int clientID;

double serviceCostWithDiscount;

bool status;

public:

ServiceOrder() {};

ServiceOrder(int ID, int serviceID, int employeeID, int clientID, std::string date, std::string time,

double serviceCostWithDiscount, bool status);

std::string getTime();

int getID();

std::string getDate();

double getCost();

void setStatus(bool status);

bool getStatus();

double getServiceCostWithDiscount();

int getClientID();

int getEmployeeID();

int getServiceID();

std::string getInfo();

std::string getName();

};

---------

**speciality.h**

---------

#pragma once

//#define barber 0

//#define administrator 1

//#define cleaner 2

//#define director 3

enum userSpeciality { BARBER = 0, ADMINISTRATOR, CLEANER, DIRECTOR };

CONST userSpeciality DEF\_USER\_SPECIALITY = BARBER;

CONST std::string DEF\_PERSONAL\_ACHIEVEMENTS = "ПОБЕДИТЕЛЬ ПО ЖИЗНИ!";

CONST double DEF\_WORK\_EXPERIENCE = 0;

---------

**status.h**

---------

#pragma once

//#define admin 0

//#define client 1

//#define employee 2

#define NOT\_FOUND -1

CONST int DEF\_VISIT\_NUM = 0;

enum userType { ADMIN = 0, EMPLOYEE, CLIENT};

enum clientStatusType { REGULAR = 10, ONCE = 0, NEW = 15, VIP = 25 };

---------

**User.h**

---------

#pragma once

#include<iostream>

#include<cstdio>

#include<cstdlib>

#include <string>

#include <map>

#include <fstream>

#include "msclr\marshal\_cppstd.h"

#include "status.h"

#include "speciality.h"

#include <unordered\_map>

//using namespace std;

using namespace System;

using namespace System::ComponentModel;

using namespace System::Collections;

using namespace System::Windows::Forms;

using namespace System::Data;

using namespace System::Drawing;

class User

{

public:

int id;

bool sex;

userType status;

std::string name;

std::string date;

std::string phonenumber;

std::string email;

size\_t password;

User() {};

User(int id, std::string name, bool sex, std::string date, std::string phonenumber,

std::string email, size\_t password, userType status);

int getID();

std::string getName();

std::string getSex();

std::string getPhonenumber();

std::string getEmail();

std::string getStatusString();

void setName(std::string name);

void User::setSex(bool sex);

void setPhonenumber(std::string phonenumber);

void setEmail(std::string email);

void setStatus(userType status);

void setDate(std::string date);

void setPassword(size\_t password);

userType getStatus();

std::string getDate();

size\_t getPassword();

virtual std::string getInfo();

/\*friend bool operator==(User& left,const std::string& right);

friend bool operator!=(User& left, const std::string& right);

friend bool operator==(const User& left, std::string right);

friend bool operator!=(const User& left, std::string right);\*/

};

---------

**Admin.h**

---------

#pragma once

#include "User.h"

#include "MockDatabase.h"

class Admin : public User {

public:

Admin() {};

Admin(int id, std::string name, bool sex, std::string date, std::string phonenumber,

std::string email, size\_t password, userType status) : User(id, name, sex, date, phonenumber, email, password, status) {};

void deleteSpecificUser(int id);

void deleteService(int id);

void deleteOrder(int id);

void changeService(int id);

void changeSpecificUser(int id);

};

---------

**checkFunctions.h**

---------

#ifndef CHECKFUNCTIONS\_H

#define CHECKFUNCTIONS\_H

#include "User.h"

#include "speciality.h"

bool checkSex(std::string sex);

userType checkSelectedStatus(std::string statusString);

userSpeciality checkSelectedSpeciality(std::string selectedSpeciality);

clientStatusType checkClientStatus(int numberOfVisits);

std::string statusToString(userType statusString);

std::string specialityToString(userType statusString);

static std::string availableTime[8]{ "9:00", "10:00", "11:00", "12:00",

"13:00", "14:00", "15:00", "16:00" };

#endif CHECKFUNCTIONS\_H

---------

**Client.h**

---------

#pragma once

#include "User.h"

class Client : public User {

private:

int numberOfVisits;

clientStatusType clientStatus;

public:

Client() {};

Client(int id, std::string name, bool sex, std::string date, std::string phonenumber,

std::string email, size\_t password, userType status, int numberOfVisits,

clientStatusType clientStatus) : User(id, name, sex, date, phonenumber, email, password, status)

{

this->clientStatus = clientStatus;

this->numberOfVisits = numberOfVisits;

};

std::string getClientStatus();

int getClientDiscount();

int getNumberOfVisits();

void setClientStatus(clientStatusType clientStatus);

void addVisit();

void getService();

virtual std::string getInfo();

};

---------

**Employee.h**

---------

#pragma once

#include "User.h"

class Employee : public User {

private:

double experience;

userSpeciality speciality;

std::string personalAchievements;

double salary;

double workingHours;

public:

Employee() {};

Employee(int id, std::string name, bool sex, std::string date, std::string phonenumber,

std::string email, size\_t password, userType status, double experience, userSpeciality speciality,

std::string personalAchievements): User(id, name, sex, date, phonenumber, email, password, status)

{

this->experience = experience;

this->personalAchievements = personalAchievements;

this->speciality = speciality;

switch (speciality) {

case BARBER:

this->salary = 20250 + experience \* 500;

this->workingHours = 8;

break;

case ADMINISTRATOR:

this->salary = 15000 + experience \* 600;

this->workingHours = 8;

break;

case CLEANER:

this->salary = 13200 + experience \* 200;

this->workingHours = 8;

break;

case DIRECTOR:

this->salary = 66666 + experience \* 666;

this->workingHours = 4;

break;

default:

System::Windows::Forms::MessageBox::Show("There is no such status");

break;

}

};

void setExperience(double experience);

void setSpeciality(userSpeciality speciality);

std::string getSpeciality();

void setPersonalAchievements(std::string personalAchievements);

double getExperience();

std::string getPersonalAchievements();

double getSalary();

double getWorkingHours();

virtual std::string getInfo();

};

---------

**MockDatabase.h**

---------

#ifndef MOCKDATABASE\_H

#define MOCKDATABASE\_H

#include "User.h"

#include "Admin.h"

#include "Employee.h"

#include "Client.h"

#include "checkFunctions.h"

#include "checkFunctions.h"

#include "ServiceOrder.h"

#include "Service.h"

//using namespace std;

using namespace System;

using namespace System::ComponentModel;

using namespace System::Collections;

using namespace System::Windows::Forms;

using namespace System::Data;

using namespace System::Drawing;

class MockDatabase : public User

{

private:

std::list<User\*> listOfUsers;

std::list<ServiceOrder\*> listOfServiceOrders;

std::list<Service\*> listOfService;

static MockDatabase\* db;

public:

//void addUser(std::string name, bool sex, std::string date, std::string phonenumber, std::string email,

// std::string password, userType status, int numberOfVisits = DEF\_VISIT\_NUM, double experience = DEF\_WORK\_EXPERIENCE,

// userSpeciality speciality = DEF\_USER\_SPECIALITY, std::string personalAchievements = DEF\_PERSONAL\_ACHIEVEMENTS);

MockDatabase()

{

}

#pragma region Aphonin

void addUser(std::string name, bool sex, std::string date, std::string phonenumber, std::string email,

std::string password, userType status, int numberOfVisits = DEF\_VISIT\_NUM);

void addUser(std::string name, bool sex, std::string date, std::string phonenumber, std::string email,

std::string password, userType status, double experience,

userSpeciality speciality, std::string personalAchievements = DEF\_PERSONAL\_ACHIEVEMENTS);

void loadUser(int ID, std::string name, bool sex, std::string date, std::string phonenumber, std::string email,

std::string password, userType status, int numberOfVisits = DEF\_VISIT\_NUM);

void loadUser(int ID, std::string name, bool sex, std::string date, std::string phonenumber, std::string email,

std::string password, userType status, double experience,

userSpeciality speciality, std::string personalAchievements);

int numberOfUsers();

int findUser(std::string email, size\_t password);

int getUserStatus(int ID);

bool deleteSpecificUser(int ID);

bool emailIsUnique(std::string email);

User\* getUser(int ID);

std::string getUsers();

std::list<User\*> getUsersList();

void clearListOfServices();

void clearListOfServicesOrder();

void clearListOfUsers();

void addServiceOrder(std::string date, std::string time, int serviceID, int employeeID, int clientID, double cost, bool status);

void loadServiceOrder(int id, std::string date, std::string time, int serviceID, int employeeID, int clientID, double cost, bool status);

bool timeIsAvailable(std::string date, std::string time);

int getMaxUserID();

std::list<int> getOrdersID(Employee\* employee);

int getOrdersCount(std::string date);

bool deleteSpecificServiceOrder(int ID);

std::list<std::string> MockDatabase::getOrdersTimes(std::string date);

static MockDatabase\* getInstance();

int numberOfServicesOrder();

#pragma endregion

#pragma region Dmitriev

void loadService(int ID, std::string name, int price);

void addService(std::string name, int price);

Service\* getService(int ID);

ServiceOrder\* getServiceOrder(int ID);

std::list<ServiceOrder\*> getServiceOrdersList();

std::list<Service\*> getServiceList();

int numberOfServices();

bool deleteSpecificService(int ID);

int getMaxServiceOrderID();

std::string getServices();

int getMaxServiceID();

#pragma endregion

#pragma region Ankudinov

void writeDatabaseToFile();

void readDatabaseFromFile();

void writeServicesOrdersToFile();

void readServicesOrdersFromFile();

void writeServicesToFile();

void readServicesFromFile();

bool readDataAboutUser(userType status, std::vector <std::string> elems);

bool readDataAboutServiceOrder(std::vector <std::string> elems);

bool readDataAboutService(std::vector <std::string> elems);

#pragma endregion

~MockDatabase() // деструктор

{

listOfService.clear();

listOfServiceOrders.clear();

listOfUsers.clear();

//delete db;

}

};

//static MockDatabase db;

#endif MOCKDATABASE\_H

---------

employeeform.h

----------------

//Timurziev

#ifndef EMPLOYEEFORM\_H

#define EMPLOYEEFORM\_H

#include "MockDatabase.h"

#include "checkFunctions.h"

namespace bshop {

using namespace System;

using namespace System::ComponentModel;

using namespace System::Collections;

using namespace System::Windows::Forms;

using namespace System::Data;

using namespace System::Drawing;

/// <summary>

/// Сводка для employeeForm

/// </summary>

public ref class employeeForm : public System::Windows::Forms::Form

{

public:

employeeForm(int ID, int userStatus)

{

InitializeComponent();

this->ID = ID;

this->userStatus = userStatus;

}

protected:

/// <summary>

/// Освободить все используемые ресурсы.

/// </summary>

~employeeForm()

{

if (components)

{

delete components;

}

}

private:

int ID;

int userStatus;

private: System::Windows::Forms::Panel^ employeePanel;

private: System::Windows::Forms::Button^ updateOrdersGridBtn;

private: System::Windows::Forms::Label^ label6;

private: System::Windows::Forms::Label^ label5;

private: System::Windows::Forms::Label^ selectDataLbl;

private: System::Windows::Forms::Label^ selectedDateEmployeeLbl;

private: System::Windows::Forms::TextBox^ selectedDateEmployeeTB;

private: System::Windows::Forms::MonthCalendar^ employeeCalendar;

private: System::Windows::Forms::Label^ label13;

private: System::Windows::Forms::TextBox^ achievementsEmployeeTB;

private: System::Windows::Forms::TextBox^ specialityEmployeeTB;

private: System::Windows::Forms::TextBox^ nameEmployeeTB;

private: System::Windows::Forms::Label^ orderEmployeeLbl;

private: System::Windows::Forms::ComboBox^ orderEmployeeIdCB;

private: System::Windows::Forms::CheckBox^ orderEmployeeCheckCB;

private: System::Windows::Forms::DataGridView^ employeeDataGrid;

private: System::Windows::Forms::DataGridViewTextBoxColumn^ employeeIdDG;

private: System::Windows::Forms::DataGridViewTextBoxColumn^ dateEmployeeDG;

private: System::Windows::Forms::DataGridViewTextBoxColumn^ serviceEmployeeDG;

private: System::Windows::Forms::DataGridViewTextBoxColumn^ clientNameEmployeeDG;

private: System::Windows::Forms::DataGridViewTextBoxColumn^ costOfServiceEmployeeDG;

private: System::Windows::Forms::DataGridViewTextBoxColumn^ ordersTime;

private: System::Windows::Forms::DataGridViewTextBoxColumn^ orderCurStatus;

private: System::Windows::Forms::Label^ achievementsEmployeeLbl;

private: System::Windows::Forms::Label^ specialityEmployeeLbl;

private: System::Windows::Forms::Label^ nameEmployeeLbl;

private: System::Windows::Forms::Button^ exitFromEmployeePanelBtn;

/// <summary>

/// Обязательная переменная конструктора.

/// </summary>

System::ComponentModel::Container ^components;

#pragma region Windows Form Designer generated code

/// <summary>

/// Требуемый метод для поддержки конструктора — не изменяйте

/// содержимое этого метода с помощью редактора кода.

/// </summary>

void InitializeComponent(void)

{

System::ComponentModel::ComponentResourceManager^ resources = (gcnew System::ComponentModel::ComponentResourceManager(employeeForm::typeid));

this->employeePanel = (gcnew System::Windows::Forms::Panel());

this->updateOrdersGridBtn = (gcnew System::Windows::Forms::Button());

this->label6 = (gcnew System::Windows::Forms::Label());

this->label5 = (gcnew System::Windows::Forms::Label());

this->selectDataLbl = (gcnew System::Windows::Forms::Label());

this->selectedDateEmployeeLbl = (gcnew System::Windows::Forms::Label());

this->selectedDateEmployeeTB = (gcnew System::Windows::Forms::TextBox());

this->employeeCalendar = (gcnew System::Windows::Forms::MonthCalendar());

this->label13 = (gcnew System::Windows::Forms::Label());

this->achievementsEmployeeTB = (gcnew System::Windows::Forms::TextBox());

this->specialityEmployeeTB = (gcnew System::Windows::Forms::TextBox());

this->nameEmployeeTB = (gcnew System::Windows::Forms::TextBox());

this->orderEmployeeLbl = (gcnew System::Windows::Forms::Label());

this->orderEmployeeIdCB = (gcnew System::Windows::Forms::ComboBox());

this->orderEmployeeCheckCB = (gcnew System::Windows::Forms::CheckBox());

this->employeeDataGrid = (gcnew System::Windows::Forms::DataGridView());

this->employeeIdDG = (gcnew System::Windows::Forms::DataGridViewTextBoxColumn());

this->dateEmployeeDG = (gcnew System::Windows::Forms::DataGridViewTextBoxColumn());

this->serviceEmployeeDG = (gcnew System::Windows::Forms::DataGridViewTextBoxColumn());

this->clientNameEmployeeDG = (gcnew System::Windows::Forms::DataGridViewTextBoxColumn());

this->costOfServiceEmployeeDG = (gcnew System::Windows::Forms::DataGridViewTextBoxColumn());

this->ordersTime = (gcnew System::Windows::Forms::DataGridViewTextBoxColumn());

this->orderCurStatus = (gcnew System::Windows::Forms::DataGridViewTextBoxColumn());

this->achievementsEmployeeLbl = (gcnew System::Windows::Forms::Label());

this->specialityEmployeeLbl = (gcnew System::Windows::Forms::Label());

this->nameEmployeeLbl = (gcnew System::Windows::Forms::Label());

this->exitFromEmployeePanelBtn = (gcnew System::Windows::Forms::Button());

this->employeePanel->SuspendLayout();

(cli::safe\_cast<System::ComponentModel::ISupportInitialize^>(this->employeeDataGrid))->BeginInit();

this->SuspendLayout();

//

// employeePanel

//

this->employeePanel->Controls->Add(this->updateOrdersGridBtn);

this->employeePanel->Controls->Add(this->label6);

this->employeePanel->Controls->Add(this->label5);

this->employeePanel->Controls->Add(this->selectDataLbl);

this->employeePanel->Controls->Add(this->selectedDateEmployeeLbl);

this->employeePanel->Controls->Add(this->selectedDateEmployeeTB);

this->employeePanel->Controls->Add(this->employeeCalendar);

this->employeePanel->Controls->Add(this->label13);

this->employeePanel->Controls->Add(this->achievementsEmployeeTB);

this->employeePanel->Controls->Add(this->specialityEmployeeTB);

this->employeePanel->Controls->Add(this->nameEmployeeTB);

this->employeePanel->Controls->Add(this->orderEmployeeLbl);

this->employeePanel->Controls->Add(this->orderEmployeeIdCB);

this->employeePanel->Controls->Add(this->orderEmployeeCheckCB);

this->employeePanel->Controls->Add(this->employeeDataGrid);

this->employeePanel->Controls->Add(this->achievementsEmployeeLbl);

this->employeePanel->Controls->Add(this->specialityEmployeeLbl);

this->employeePanel->Controls->Add(this->nameEmployeeLbl);

this->employeePanel->Controls->Add(this->exitFromEmployeePanelBtn);

this->employeePanel->Font = (gcnew System::Drawing::Font(L"Yu Gothic UI Semibold", 12.25F, System::Drawing::FontStyle::Bold));

this->employeePanel->Location = System::Drawing::Point(1, 0);

this->employeePanel->Name = L"employeePanel";

this->employeePanel->Size = System::Drawing::Size(913, 712);

this->employeePanel->TabIndex = 19;

//

// updateOrdersGridBtn

//

this->updateOrdersGridBtn->BackgroundImage = (cli::safe\_cast<System::Drawing::Image^>(resources->GetObject(L"updateOrdersGridBtn.BackgroundImage")));

this->updateOrdersGridBtn->BackgroundImageLayout = System::Windows::Forms::ImageLayout::Zoom;

this->updateOrdersGridBtn->Location = System::Drawing::Point(791, 296);

this->updateOrdersGridBtn->Name = L"updateOrdersGridBtn";

this->updateOrdersGridBtn->Size = System::Drawing::Size(40, 40);

this->updateOrdersGridBtn->TabIndex = 28;

this->updateOrdersGridBtn->UseVisualStyleBackColor = true;

this->updateOrdersGridBtn->Click += gcnew System::EventHandler(this, &employeeForm::updateOrdersGridBtn\_Click);

//

// label6

//

this->label6->Anchor = System::Windows::Forms::AnchorStyles::Left;

this->label6->AutoSize = true;

this->label6->Font = (gcnew System::Drawing::Font(L"Yu Gothic UI Semibold", 8, System::Drawing::FontStyle::Bold));

this->label6->Location = System::Drawing::Point(68, 538);

this->label6->Name = L"label6";

this->label6->Size = System::Drawing::Size(235, 13);

this->label6->TabIndex = 20;

this->label6->Text = L"---------------------------------------------------------";

//

// label5

//

this->label5->Anchor = System::Windows::Forms::AnchorStyles::Left;

this->label5->AutoSize = true;

this->label5->Font = (gcnew System::Drawing::Font(L"Yu Gothic UI Semibold", 8, System::Drawing::FontStyle::Bold));

this->label5->Location = System::Drawing::Point(68, 393);

this->label5->Name = L"label5";

this->label5->Size = System::Drawing::Size(235, 13);

this->label5->TabIndex = 19;

this->label5->Text = L"---------------------------------------------------------";

//

// selectDataLbl

//

this->selectDataLbl->Anchor = System::Windows::Forms::AnchorStyles::Left;

this->selectDataLbl->AutoSize = true;

this->selectDataLbl->Font = (gcnew System::Drawing::Font(L"Yu Gothic UI Semibold", 9.25F, System::Drawing::FontStyle::Bold));

this->selectDataLbl->Location = System::Drawing::Point(373, 303);

this->selectDataLbl->Name = L"selectDataLbl";

this->selectDataLbl->Size = System::Drawing::Size(102, 17);

this->selectDataLbl->TabIndex = 17;

this->selectDataLbl->Text = L"Выберите дату";

//

// selectedDateEmployeeLbl

//

this->selectedDateEmployeeLbl->Anchor = System::Windows::Forms::AnchorStyles::Left;

this->selectedDateEmployeeLbl->AutoSize = true;

this->selectedDateEmployeeLbl->Font = (gcnew System::Drawing::Font(L"Yu Gothic UI Semibold", 9.25F, System::Drawing::FontStyle::Bold));

this->selectedDateEmployeeLbl->Location = System::Drawing::Point(373, 511);

this->selectedDateEmployeeLbl->Name = L"selectedDateEmployeeLbl";

this->selectedDateEmployeeLbl->Size = System::Drawing::Size(114, 17);

this->selectedDateEmployeeLbl->TabIndex = 16;

this->selectedDateEmployeeLbl->Text = L"Выбранная дата:";

//

// selectedDateEmployeeTB

//

this->selectedDateEmployeeTB->Anchor = System::Windows::Forms::AnchorStyles::Left;

this->selectedDateEmployeeTB->Font = (gcnew System::Drawing::Font(L"Yu Gothic UI Semibold", 9.25F, System::Drawing::FontStyle::Bold));

this->selectedDateEmployeeTB->Location = System::Drawing::Point(376, 538);

this->selectedDateEmployeeTB->Name = L"selectedDateEmployeeTB";

this->selectedDateEmployeeTB->Size = System::Drawing::Size(164, 24);

this->selectedDateEmployeeTB->TabIndex = 15;

//

// employeeCalendar

//

this->employeeCalendar->Anchor = System::Windows::Forms::AnchorStyles::Left;

this->employeeCalendar->Location = System::Drawing::Point(376, 334);

this->employeeCalendar->MaxSelectionCount = 1;

this->employeeCalendar->Name = L"employeeCalendar";

this->employeeCalendar->TabIndex = 2;

this->employeeCalendar->DateChanged += gcnew System::Windows::Forms::DateRangeEventHandler(this, &employeeForm::employeeCalendar\_DateChanged);

this->employeeCalendar->DateSelected += gcnew System::Windows::Forms::DateRangeEventHandler(this, &employeeForm::employeeCalendar\_DateSelected);

//

// label13

//

this->label13->Anchor = System::Windows::Forms::AnchorStyles::Left;

this->label13->AutoSize = true;

this->label13->Font = (gcnew System::Drawing::Font(L"Yu Gothic UI Semibold", 8, System::Drawing::FontStyle::Bold));

this->label13->Location = System::Drawing::Point(70, 334);

this->label13->Name = L"label13";

this->label13->Size = System::Drawing::Size(235, 13);

this->label13->TabIndex = 12;

this->label13->Text = L"---------------------------------------------------------";

//

// achievementsEmployeeTB

//

this->achievementsEmployeeTB->Anchor = System::Windows::Forms::AnchorStyles::Left;

this->achievementsEmployeeTB->Font = (gcnew System::Drawing::Font(L"Yu Gothic UI Semibold", 9.25F, System::Drawing::FontStyle::Bold));

this->achievementsEmployeeTB->Location = System::Drawing::Point(72, 435);

this->achievementsEmployeeTB->Multiline = true;

this->achievementsEmployeeTB->Name = L"achievementsEmployeeTB";

this->achievementsEmployeeTB->ReadOnly = true;

this->achievementsEmployeeTB->Size = System::Drawing::Size(231, 93);

this->achievementsEmployeeTB->TabIndex = 10;

//

// specialityEmployeeTB

//

this->specialityEmployeeTB->Anchor = System::Windows::Forms::AnchorStyles::Left;

this->specialityEmployeeTB->Font = (gcnew System::Drawing::Font(L"Yu Gothic UI Semibold", 9.25F, System::Drawing::FontStyle::Bold));

this->specialityEmployeeTB->Location = System::Drawing::Point(203, 361);

this->specialityEmployeeTB->Name = L"specialityEmployeeTB";

this->specialityEmployeeTB->ReadOnly = true;

this->specialityEmployeeTB->Size = System::Drawing::Size(100, 24);

this->specialityEmployeeTB->TabIndex = 9;

//

// nameEmployeeTB

//

this->nameEmployeeTB->Anchor = System::Windows::Forms::AnchorStyles::Left;

this->nameEmployeeTB->Font = (gcnew System::Drawing::Font(L"Yu Gothic UI Semibold", 9.25F, System::Drawing::FontStyle::Bold));

this->nameEmployeeTB->Location = System::Drawing::Point(203, 300);

this->nameEmployeeTB->Name = L"nameEmployeeTB";

this->nameEmployeeTB->ReadOnly = true;

this->nameEmployeeTB->Size = System::Drawing::Size(100, 24);

this->nameEmployeeTB->TabIndex = 8;

//

// orderEmployeeLbl

//

this->orderEmployeeLbl->Anchor = System::Windows::Forms::AnchorStyles::Left;

this->orderEmployeeLbl->AutoSize = true;

this->orderEmployeeLbl->Font = (gcnew System::Drawing::Font(L"Yu Gothic UI Semibold", 9.25F, System::Drawing::FontStyle::Bold));

this->orderEmployeeLbl->Location = System::Drawing::Point(72, 565);

this->orderEmployeeLbl->Name = L"orderEmployeeLbl";

this->orderEmployeeLbl->Size = System::Drawing::Size(42, 17);

this->orderEmployeeLbl->TabIndex = 7;

this->orderEmployeeLbl->Text = L"Заказ";

//

// orderEmployeeIdCB

//

this->orderEmployeeIdCB->Anchor = System::Windows::Forms::AnchorStyles::Left;

this->orderEmployeeIdCB->Font = (gcnew System::Drawing::Font(L"Yu Gothic UI Semibold", 9.25F, System::Drawing::FontStyle::Bold));

this->orderEmployeeIdCB->FormattingEnabled = true;

this->orderEmployeeIdCB->Location = System::Drawing::Point(203, 559);

this->orderEmployeeIdCB->Name = L"orderEmployeeIdCB";

this->orderEmployeeIdCB->Size = System::Drawing::Size(100, 23);

this->orderEmployeeIdCB->TabIndex = 6;

this->orderEmployeeIdCB->SelectionChangeCommitted += gcnew System::EventHandler(this, &employeeForm::orderEmployeeIdCB\_SelectionChangeCommitted);

//

// orderEmployeeCheckCB

//

this->orderEmployeeCheckCB->Anchor = System::Windows::Forms::AnchorStyles::Left;

this->orderEmployeeCheckCB->AutoSize = true;

this->orderEmployeeCheckCB->Font = (gcnew System::Drawing::Font(L"Yu Gothic UI Semibold", 9.25F, System::Drawing::FontStyle::Bold));

this->orderEmployeeCheckCB->Location = System::Drawing::Point(203, 588);

this->orderEmployeeCheckCB->Name = L"orderEmployeeCheckCB";

this->orderEmployeeCheckCB->Size = System::Drawing::Size(91, 21);

this->orderEmployeeCheckCB->TabIndex = 5;

this->orderEmployeeCheckCB->Text = L"Выполнен";

this->orderEmployeeCheckCB->UseVisualStyleBackColor = true;

this->orderEmployeeCheckCB->CheckedChanged += gcnew System::EventHandler(this, &employeeForm::orderEmployeeCheckCB\_CheckedChanged);

//

// employeeDataGrid

//

this->employeeDataGrid->AllowUserToAddRows = false;

this->employeeDataGrid->AllowUserToDeleteRows = false;

this->employeeDataGrid->Anchor = System::Windows::Forms::AnchorStyles::Left;

this->employeeDataGrid->AutoSizeColumnsMode = System::Windows::Forms::DataGridViewAutoSizeColumnsMode::Fill;

this->employeeDataGrid->BackgroundColor = System::Drawing::SystemColors::ButtonHighlight;

this->employeeDataGrid->ColumnHeadersHeightSizeMode = System::Windows::Forms::DataGridViewColumnHeadersHeightSizeMode::AutoSize;

this->employeeDataGrid->Columns->AddRange(gcnew cli::array< System::Windows::Forms::DataGridViewColumn^ >(7) {

this->employeeIdDG,

this->dateEmployeeDG, this->serviceEmployeeDG, this->clientNameEmployeeDG, this->costOfServiceEmployeeDG, this->ordersTime, this->orderCurStatus

});

this->employeeDataGrid->Location = System::Drawing::Point(71, 48);

this->employeeDataGrid->Name = L"employeeDataGrid";

this->employeeDataGrid->Size = System::Drawing::Size(760, 238);

this->employeeDataGrid->TabIndex = 4;

//

// employeeIdDG

//

this->employeeIdDG->HeaderText = L"id";

this->employeeIdDG->Name = L"employeeIdDG";

//

// dateEmployeeDG

//

this->dateEmployeeDG->HeaderText = L"Дата";

this->dateEmployeeDG->Name = L"dateEmployeeDG";

//

// serviceEmployeeDG

//

this->serviceEmployeeDG->HeaderText = L"Услуга";

this->serviceEmployeeDG->Name = L"serviceEmployeeDG";

//

// clientNameEmployeeDG

//

this->clientNameEmployeeDG->HeaderText = L"Клиент";

this->clientNameEmployeeDG->Name = L"clientNameEmployeeDG";

//

// costOfServiceEmployeeDG

//

this->costOfServiceEmployeeDG->HeaderText = L"Стоимость";

this->costOfServiceEmployeeDG->Name = L"costOfServiceEmployeeDG";

//

// ordersTime

//

this->ordersTime->HeaderText = L"Время";

this->ordersTime->Name = L"ordersTime";

//

// orderCurStatus

//

this->orderCurStatus->HeaderText = L"Статус";

this->orderCurStatus->Name = L"orderCurStatus";

//

// achievementsEmployeeLbl

//

this->achievementsEmployeeLbl->Anchor = System::Windows::Forms::AnchorStyles::Left;

this->achievementsEmployeeLbl->AutoSize = true;

this->achievementsEmployeeLbl->Font = (gcnew System::Drawing::Font(L"Yu Gothic UI Semibold", 9.25F, System::Drawing::FontStyle::Bold));

this->achievementsEmployeeLbl->Location = System::Drawing::Point(68, 411);

this->achievementsEmployeeLbl->Name = L"achievementsEmployeeLbl";

this->achievementsEmployeeLbl->Size = System::Drawing::Size(138, 17);

this->achievementsEmployeeLbl->TabIndex = 3;

this->achievementsEmployeeLbl->Text = L"Личные достижения";

//

// specialityEmployeeLbl

//

this->specialityEmployeeLbl->Anchor = System::Windows::Forms::AnchorStyles::Left;

this->specialityEmployeeLbl->AutoSize = true;

this->specialityEmployeeLbl->Font = (gcnew System::Drawing::Font(L"Yu Gothic UI Semibold", 9.25F, System::Drawing::FontStyle::Bold));

this->specialityEmployeeLbl->Location = System::Drawing::Point(72, 364);

this->specialityEmployeeLbl->Name = L"specialityEmployeeLbl";

this->specialityEmployeeLbl->Size = System::Drawing::Size(77, 17);

this->specialityEmployeeLbl->TabIndex = 2;

this->specialityEmployeeLbl->Text = L"Должность";

//

// nameEmployeeLbl

//

this->nameEmployeeLbl->Anchor = System::Windows::Forms::AnchorStyles::Left;

this->nameEmployeeLbl->AutoSize = true;

this->nameEmployeeLbl->Font = (gcnew System::Drawing::Font(L"Yu Gothic UI Semibold", 9.25F, System::Drawing::FontStyle::Bold));

this->nameEmployeeLbl->Location = System::Drawing::Point(72, 307);

this->nameEmployeeLbl->Name = L"nameEmployeeLbl";

this->nameEmployeeLbl->Size = System::Drawing::Size(35, 17);

this->nameEmployeeLbl->TabIndex = 1;

this->nameEmployeeLbl->Text = L"Имя";

//

// exitFromEmployeePanelBtn

//

this->exitFromEmployeePanelBtn->Anchor = System::Windows::Forms::AnchorStyles::Left;

this->exitFromEmployeePanelBtn->Font = (gcnew System::Drawing::Font(L"Yu Gothic UI Semibold", 9.25F, System::Drawing::FontStyle::Bold));

this->exitFromEmployeePanelBtn->Location = System::Drawing::Point(737, 607);

this->exitFromEmployeePanelBtn->Name = L"exitFromEmployeePanelBtn";

this->exitFromEmployeePanelBtn->Size = System::Drawing::Size(94, 35);

this->exitFromEmployeePanelBtn->TabIndex = 0;

this->exitFromEmployeePanelBtn->Text = L"Выход";

this->exitFromEmployeePanelBtn->UseVisualStyleBackColor = true;

this->exitFromEmployeePanelBtn->Click += gcnew System::EventHandler(this, &employeeForm::exitFromEmployeePanelBtn\_Click);

//

// employeeForm

//

this->AutoScaleDimensions = System::Drawing::SizeF(6, 13);

this->AutoScaleMode = System::Windows::Forms::AutoScaleMode::Font;

this->ClientSize = System::Drawing::Size(914, 712);

this->Controls->Add(this->employeePanel);

this->Name = L"employeeForm";

this->Text = L"Работник";

this->FormClosing += gcnew System::Windows::Forms::FormClosingEventHandler(this, &employeeForm::employeeForm\_FormClosing);

this->Load += gcnew System::EventHandler(this, &employeeForm::employeeForm\_Load);

this->employeePanel->ResumeLayout(false);

this->employeePanel->PerformLayout();

(cli::safe\_cast<System::ComponentModel::ISupportInitialize^>(this->employeeDataGrid))->EndInit();

this->ResumeLayout(false);

}

#pragma endregion

private: System::Void employeeForm\_Load(System::Object^ sender, System::EventArgs^ e);

private: System::Void orderEmployeeIdCB\_SelectionChangeCommitted(System::Object^ sender, System::EventArgs^ e);

private: System::Void employeeCalendar\_DateChanged(System::Object^ sender, System::Windows::Forms::DateRangeEventArgs^ e);

private: System::Void employeeCalendar\_DateSelected(System::Object^ sender, System::Windows::Forms::DateRangeEventArgs^ e);

private: System::Void orderEmployeeCheckCB\_CheckedChanged(System::Object^ sender, System::EventArgs^ e);

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

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

private: System::Void updateTimetable(MonthCalendar^ employeeCalendar, TextBox^ selectedDateEmployeeTB);

private: System::Void updateOrdersGrid(std::list<ServiceOrder\*> orders);

private: System::Void employeeForm\_FormClosing(System::Object^ sender, System::Windows::Forms::FormClosingEventArgs^ e);

};

}

#endif EMPLOYEEFORM\_H

----------------

adminForm.h

----------------

#ifndef ADMINFORM\_H

#define ADMINFORM\_H

#include "MockDatabase.h"

#include "checkFunctions.h"

namespace bshop {

using namespace System;

using namespace System::ComponentModel;

using namespace System::Collections;

using namespace System::Windows::Forms;

using namespace System::Data;

using namespace System::Drawing;

/// <summary>

/// Сводка для adminForm

/// </summary>

public ref class adminForm : public System::Windows::Forms::Form

{

public:

adminForm(int ID, int userStatus)

{

InitializeComponent();

this->ID = ID;

this->userStatus = userStatus;

}

protected:

/// <summary>

/// Освободить все используемые ресурсы.

/// </summary>

~adminForm()

{

if (components)

{

delete components;

}

}

private: System::Windows::Forms::Panel^ adminPanel;

protected:

private: System::Windows::Forms::TabControl^ mainTabSekector;

private: System::Windows::Forms::TabPage^ mainTab;

private: System::Windows::Forms::Label^ label1;

private: System::Windows::Forms::Button^ updateGridsBtn;

private: System::Windows::Forms::Button^ writeDataBtn;

private: System::Windows::Forms::Button^ loadBtn;

private: System::Windows::Forms::Label^ label2;

private: System::Windows::Forms::Label^ label3;

private: System::Windows::Forms::DataGridView^ usersDataGrid;

private: System::Windows::Forms::DataGridViewTextBoxColumn^ userID;

private: System::Windows::Forms::DataGridViewTextBoxColumn^ userName;

private: System::Windows::Forms::DataGridViewTextBoxColumn^ userCurrentStatus;

private: System::Windows::Forms::DataGridView^ servicesDataGrid;

private: System::Windows::Forms::DataGridViewTextBoxColumn^ servicesID;

private: System::Windows::Forms::DataGridViewTextBoxColumn^ serviceName;

private: System::Windows::Forms::DataGridView^ ordersDataGrid;

private: System::Windows::Forms::TabPage^ addUserTab;

private: System::Windows::Forms::Panel^ userRegistrationPanel;

private: System::Windows::Forms::MaskedTextBox^ addUserPhonenumberTB;

private: System::Windows::Forms::MaskedTextBox^ addUserDateTB;

private: System::Windows::Forms::GroupBox^ addUserSexGP;

private: System::Windows::Forms::RadioButton^ addUserFemaleRB;

private: System::Windows::Forms::RadioButton^ addUserMaleRB;

private: System::Windows::Forms::Panel^ specialityPanel;

private: System::Windows::Forms::Label^ addStatusLbl;

private: System::Windows::Forms::ComboBox^ addUserSpecialityCB;

private: System::Windows::Forms::ComboBox^ addUserTypeCB;

private: System::Windows::Forms::Label^ addUserTypeLbl;

private: System::Windows::Forms::Button^ addUserFormBtn;

private: System::Windows::Forms::Label^ addPassLbl;

private: System::Windows::Forms::Label^ addEmailLbl;

private: System::Windows::Forms::Label^ addPhonenumberLbl;

private: System::Windows::Forms::Label^ addDateLbl;

private: System::Windows::Forms::TextBox^ addUserPasswordTB;

private: System::Windows::Forms::TextBox^ addUserMailTB;

private: System::Windows::Forms::TextBox^ addUserNameTB;

private: System::Windows::Forms::Label^ addUser;

private: System::Windows::Forms::TabPage^ changeUserTab;

private: System::Windows::Forms::Panel^ changeUserPanel;

private: System::Windows::Forms::MaskedTextBox^ changeUserPhonenumberTB;

private: System::Windows::Forms::MaskedTextBox^ changeUserDateTB;

private: System::Windows::Forms::Panel^ changeUserSpecialityPanel;

private: System::Windows::Forms::Label^ label10;

private: System::Windows::Forms::TextBox^ changePersonalAchievementsTB;

private: System::Windows::Forms::Label^ label7;

private: System::Windows::Forms::TextBox^ changeUserExperienceTB;

private: System::Windows::Forms::Label^ label4;

private: System::Windows::Forms::ComboBox^ changeUserSpecialityCB;

private: System::Windows::Forms::GroupBox^ changeUserSexGP;

private: System::Windows::Forms::RadioButton^ changeUserFemaleRB;

private: System::Windows::Forms::RadioButton^ changeUserMaleRB;

private: System::Windows::Forms::ComboBox^ changeUserIdCB;

private: System::Windows::Forms::Label^ label9;

private: System::Windows::Forms::Label^ label8;

private: System::Windows::Forms::Label^ UserIdLbl;

private: System::Windows::Forms::ComboBox^ changeUserTypeCB;

private: System::Windows::Forms::Label^ changeUserTypeLbl;

private: System::Windows::Forms::Button^ changeUserFormBtn;

private: System::Windows::Forms::Label^ changePassLbl;

private: System::Windows::Forms::Label^ changeEmailLbl;

private: System::Windows::Forms::Label^ changePhonenumberLbl;

private: System::Windows::Forms::Label^ changeDateLbl;

private: System::Windows::Forms::TextBox^ changeUserPasswordTB;

private: System::Windows::Forms::TextBox^ changeUserMailTB;

private: System::Windows::Forms::TextBox^ changeUserNameTB;

private: System::Windows::Forms::TabPage^ addServiceTab;

private: System::Windows::Forms::Panel^ addServicePanel;

private: System::Windows::Forms::Button^ loadServicesBtn;

private: System::Windows::Forms::Button^ updateServicesBtn;

private: System::Windows::Forms::Button^ saveServicesBtn;

private: System::Windows::Forms::RichTextBox^ servicesRTB;

private: System::Windows::Forms::Label^ addServicePriceLbl;

private: System::Windows::Forms::Label^ addServices;

private: System::Windows::Forms::Label^ addServiceNameLbl;

private: System::Windows::Forms::TextBox^ serviceNameTB;

private: System::Windows::Forms::TextBox^ servicePriceTB;

private: System::Windows::Forms::Button^ addServiceBtn;

private: System::Windows::Forms::TabPage^ changeServiceTab;

private: System::Windows::Forms::Panel^ changeServicePanel;

private: System::Windows::Forms::Label^ deleteServices;

private: System::Windows::Forms::Label^ changeServicePriceLbl;

private: System::Windows::Forms::Label^ changeServiceNameLbl;

private: System::Windows::Forms::TextBox^ changeServiceNameTB;

private: System::Windows::Forms::TextBox^ changeServicePriceTB;

private: System::Windows::Forms::Button^ changeServiceFormBtn;

private: System::Windows::Forms::TabPage^ showReportTab;

private: System::Windows::Forms::Panel^ reportPanel;

private: System::Windows::Forms::CheckBox^ averageСheckCB;

private: System::Windows::Forms::CheckBox^ clientsNumberCB;

private: System::Windows::Forms::CheckBox^ profitCB;

private: System::Windows::Forms::CheckBox^ proceedsCB;

private: System::Windows::Forms::CheckBox^ ExpensesCB;

private: System::Windows::Forms::RichTextBox^ makeReportRTB;

private: System::Windows::Forms::Button^ makeReportBtn;

private: System::Windows::Forms::ComboBox^ reportPeriodCB;

private: System::Windows::Forms::Label^ reportPeriodLbl;

private: System::Windows::Forms::Label^ financialStatementsLbl;

private: System::Windows::Forms::Button^ deleteSpecidicServiceOrderBtn;

private: System::Windows::Forms::Button^ deleteSpecificUser;

private: System::Windows::Forms::Button^ showReportBtn;

private: System::Windows::Forms::Button^ changeServiceBtn;

private: System::Windows::Forms::Button^ addServiceBtnFromAdminPanel;

private: System::Windows::Forms::Button^ delOrderBtn;

private: System::Windows::Forms::Button^ changeUserBtn;

private: System::Windows::Forms::Button^ addUserBtn;

private: System::Windows::Forms::Label^ AdminLbl;

private:

int ID;

int userStatus;

private: System::Windows::Forms::DataGridViewTextBoxColumn^ Price;

private: System::Windows::Forms::DataGridViewTextBoxColumn^ orderID;

private: System::Windows::Forms::DataGridViewTextBoxColumn^ employeeID;

private: System::Windows::Forms::DataGridViewTextBoxColumn^ clientID;

private: System::Windows::Forms::DataGridViewTextBoxColumn^ orderStatus;

private: System::Windows::Forms::DataGridViewTextBoxColumn^ serviceInOrder;

/// <summary>

/// Обязательная переменная конструктора.

/// </summary>

System::ComponentModel::Container ^components;

#pragma region Windows Form Designer generated code

/// <summary>

/// Требуемый метод для поддержки конструктора — не изменяйте

/// содержимое этого метода с помощью редактора кода.

/// </summary>

void InitializeComponent(void)

{

System::Windows::Forms::Label^ changeNameLbl;

System::Windows::Forms::Label^ addNameLbl;

System::ComponentModel::ComponentResourceManager^ resources = (gcnew System::ComponentModel::ComponentResourceManager(adminForm::typeid));

this->adminPanel = (gcnew System::Windows::Forms::Panel());

this->mainTabSekector = (gcnew System::Windows::Forms::TabControl());

this->mainTab = (gcnew System::Windows::Forms::TabPage());

this->label1 = (gcnew System::Windows::Forms::Label());

this->updateGridsBtn = (gcnew System::Windows::Forms::Button());

this->writeDataBtn = (gcnew System::Windows::Forms::Button());

this->loadBtn = (gcnew System::Windows::Forms::Button());

this->label2 = (gcnew System::Windows::Forms::Label());

this->label3 = (gcnew System::Windows::Forms::Label());

this->usersDataGrid = (gcnew System::Windows::Forms::DataGridView());

this->userID = (gcnew System::Windows::Forms::DataGridViewTextBoxColumn());

this->userName = (gcnew System::Windows::Forms::DataGridViewTextBoxColumn());

this->userCurrentStatus = (gcnew System::Windows::Forms::DataGridViewTextBoxColumn());

this->servicesDataGrid = (gcnew System::Windows::Forms::DataGridView());

this->servicesID = (gcnew System::Windows::Forms::DataGridViewTextBoxColumn());

this->serviceName = (gcnew System::Windows::Forms::DataGridViewTextBoxColumn());

this->Price = (gcnew System::Windows::Forms::DataGridViewTextBoxColumn());

this->ordersDataGrid = (gcnew System::Windows::Forms::DataGridView());

this->orderID = (gcnew System::Windows::Forms::DataGridViewTextBoxColumn());

this->employeeID = (gcnew System::Windows::Forms::DataGridViewTextBoxColumn());

this->clientID = (gcnew System::Windows::Forms::DataGridViewTextBoxColumn());

this->orderStatus = (gcnew System::Windows::Forms::DataGridViewTextBoxColumn());

this->serviceInOrder = (gcnew System::Windows::Forms::DataGridViewTextBoxColumn());

this->addUserTab = (gcnew System::Windows::Forms::TabPage());

this->userRegistrationPanel = (gcnew System::Windows::Forms::Panel());

this->addUserPhonenumberTB = (gcnew System::Windows::Forms::MaskedTextBox());

this->addUserDateTB = (gcnew System::Windows::Forms::MaskedTextBox());

this->addUserSexGP = (gcnew System::Windows::Forms::GroupBox());

this->addUserFemaleRB = (gcnew System::Windows::Forms::RadioButton());

this->addUserMaleRB = (gcnew System::Windows::Forms::RadioButton());

this->specialityPanel = (gcnew System::Windows::Forms::Panel());

this->addStatusLbl = (gcnew System::Windows::Forms::Label());

this->addUserSpecialityCB = (gcnew System::Windows::Forms::ComboBox());

this->addUserTypeCB = (gcnew System::Windows::Forms::ComboBox());

this->addUserTypeLbl = (gcnew System::Windows::Forms::Label());

this->addUserFormBtn = (gcnew System::Windows::Forms::Button());

this->addPassLbl = (gcnew System::Windows::Forms::Label());

this->addEmailLbl = (gcnew System::Windows::Forms::Label());

this->addPhonenumberLbl = (gcnew System::Windows::Forms::Label());

this->addDateLbl = (gcnew System::Windows::Forms::Label());

this->addUserPasswordTB = (gcnew System::Windows::Forms::TextBox());

this->addUserMailTB = (gcnew System::Windows::Forms::TextBox());

this->addUserNameTB = (gcnew System::Windows::Forms::TextBox());

this->addUser = (gcnew System::Windows::Forms::Label());

this->changeUserTab = (gcnew System::Windows::Forms::TabPage());

this->changeUserPanel = (gcnew System::Windows::Forms::Panel());

this->changeUserPhonenumberTB = (gcnew System::Windows::Forms::MaskedTextBox());

this->changeUserDateTB = (gcnew System::Windows::Forms::MaskedTextBox());

this->changeUserSpecialityPanel = (gcnew System::Windows::Forms::Panel());

this->label10 = (gcnew System::Windows::Forms::Label());

this->changePersonalAchievementsTB = (gcnew System::Windows::Forms::TextBox());

this->label7 = (gcnew System::Windows::Forms::Label());

this->changeUserExperienceTB = (gcnew System::Windows::Forms::TextBox());

this->label4 = (gcnew System::Windows::Forms::Label());

this->changeUserSpecialityCB = (gcnew System::Windows::Forms::ComboBox());

this->changeUserSexGP = (gcnew System::Windows::Forms::GroupBox());

this->changeUserFemaleRB = (gcnew System::Windows::Forms::RadioButton());

this->changeUserMaleRB = (gcnew System::Windows::Forms::RadioButton());

this->changeUserIdCB = (gcnew System::Windows::Forms::ComboBox());

this->label9 = (gcnew System::Windows::Forms::Label());

this->label8 = (gcnew System::Windows::Forms::Label());

this->UserIdLbl = (gcnew System::Windows::Forms::Label());

this->changeUserTypeCB = (gcnew System::Windows::Forms::ComboBox());

this->changeUserTypeLbl = (gcnew System::Windows::Forms::Label());

this->changeUserFormBtn = (gcnew System::Windows::Forms::Button());

this->changePassLbl = (gcnew System::Windows::Forms::Label());

this->changeEmailLbl = (gcnew System::Windows::Forms::Label());

this->changePhonenumberLbl = (gcnew System::Windows::Forms::Label());

this->changeDateLbl = (gcnew System::Windows::Forms::Label());

this->changeUserPasswordTB = (gcnew System::Windows::Forms::TextBox());

this->changeUserMailTB = (gcnew System::Windows::Forms::TextBox());

this->changeUserNameTB = (gcnew System::Windows::Forms::TextBox());

this->addServiceTab = (gcnew System::Windows::Forms::TabPage());

this->addServicePanel = (gcnew System::Windows::Forms::Panel());

this->loadServicesBtn = (gcnew System::Windows::Forms::Button());

this->updateServicesBtn = (gcnew System::Windows::Forms::Button());

this->saveServicesBtn = (gcnew System::Windows::Forms::Button());

this->servicesRTB = (gcnew System::Windows::Forms::RichTextBox());

this->addServicePriceLbl = (gcnew System::Windows::Forms::Label());

this->addServices = (gcnew System::Windows::Forms::Label());

this->addServiceNameLbl = (gcnew System::Windows::Forms::Label());

this->serviceNameTB = (gcnew System::Windows::Forms::TextBox());

this->servicePriceTB = (gcnew System::Windows::Forms::TextBox());

this->addServiceBtn = (gcnew System::Windows::Forms::Button());

this->changeServiceTab = (gcnew System::Windows::Forms::TabPage());

this->changeServicePanel = (gcnew System::Windows::Forms::Panel());

this->deleteServices = (gcnew System::Windows::Forms::Label());

this->changeServicePriceLbl = (gcnew System::Windows::Forms::Label());

this->changeServiceNameLbl = (gcnew System::Windows::Forms::Label());

this->changeServiceNameTB = (gcnew System::Windows::Forms::TextBox());

this->changeServicePriceTB = (gcnew System::Windows::Forms::TextBox());

this->changeServiceFormBtn = (gcnew System::Windows::Forms::Button());

this->showReportTab = (gcnew System::Windows::Forms::TabPage());

this->reportPanel = (gcnew System::Windows::Forms::Panel());

this->averageСheckCB = (gcnew System::Windows::Forms::CheckBox());

this->clientsNumberCB = (gcnew System::Windows::Forms::CheckBox());

this->profitCB = (gcnew System::Windows::Forms::CheckBox());

this->proceedsCB = (gcnew System::Windows::Forms::CheckBox());

this->ExpensesCB = (gcnew System::Windows::Forms::CheckBox());

this->makeReportRTB = (gcnew System::Windows::Forms::RichTextBox());

this->makeReportBtn = (gcnew System::Windows::Forms::Button());

this->reportPeriodCB = (gcnew System::Windows::Forms::ComboBox());

this->reportPeriodLbl = (gcnew System::Windows::Forms::Label());

this->financialStatementsLbl = (gcnew System::Windows::Forms::Label());

this->deleteSpecidicServiceOrderBtn = (gcnew System::Windows::Forms::Button());

this->deleteSpecificUser = (gcnew System::Windows::Forms::Button());

this->showReportBtn = (gcnew System::Windows::Forms::Button());

this->changeServiceBtn = (gcnew System::Windows::Forms::Button());

this->addServiceBtnFromAdminPanel = (gcnew System::Windows::Forms::Button());

this->delOrderBtn = (gcnew System::Windows::Forms::Button());

this->changeUserBtn = (gcnew System::Windows::Forms::Button());

this->addUserBtn = (gcnew System::Windows::Forms::Button());

this->AdminLbl = (gcnew System::Windows::Forms::Label());

changeNameLbl = (gcnew System::Windows::Forms::Label());

addNameLbl = (gcnew System::Windows::Forms::Label());

this->adminPanel->SuspendLayout();

this->mainTabSekector->SuspendLayout();

this->mainTab->SuspendLayout();

(cli::safe\_cast<System::ComponentModel::ISupportInitialize^>(this->usersDataGrid))->BeginInit();

(cli::safe\_cast<System::ComponentModel::ISupportInitialize^>(this->servicesDataGrid))->BeginInit();

(cli::safe\_cast<System::ComponentModel::ISupportInitialize^>(this->ordersDataGrid))->BeginInit();

this->addUserTab->SuspendLayout();

this->userRegistrationPanel->SuspendLayout();

this->addUserSexGP->SuspendLayout();

this->specialityPanel->SuspendLayout();

this->changeUserTab->SuspendLayout();

this->changeUserPanel->SuspendLayout();

this->changeUserSpecialityPanel->SuspendLayout();

this->changeUserSexGP->SuspendLayout();

this->addServiceTab->SuspendLayout();

this->addServicePanel->SuspendLayout();

this->changeServiceTab->SuspendLayout();

this->changeServicePanel->SuspendLayout();

this->showReportTab->SuspendLayout();

this->reportPanel->SuspendLayout();

this->SuspendLayout();

//

// changeNameLbl

//

changeNameLbl->AutoSize = true;

changeNameLbl->Location = System::Drawing::Point(224, 83);

changeNameLbl->Name = L"changeNameLbl";

changeNameLbl->Size = System::Drawing::Size(35, 17);

changeNameLbl->TabIndex = 45;

changeNameLbl->Text = L"Имя";

//

// addNameLbl

//

addNameLbl->AutoSize = true;

addNameLbl->Location = System::Drawing::Point(98, 51);

addNameLbl->Name = L"addNameLbl";

addNameLbl->Size = System::Drawing::Size(35, 17);

addNameLbl->TabIndex = 27;

addNameLbl->Text = L"Имя";

//

// adminPanel

//

this->adminPanel->Controls->Add(this->mainTabSekector);

this->adminPanel->Controls->Add(this->deleteSpecidicServiceOrderBtn);

this->adminPanel->Controls->Add(this->deleteSpecificUser);

this->adminPanel->Controls->Add(this->showReportBtn);

this->adminPanel->Controls->Add(this->changeServiceBtn);

this->adminPanel->Controls->Add(this->addServiceBtnFromAdminPanel);

this->adminPanel->Controls->Add(this->delOrderBtn);

this->adminPanel->Controls->Add(this->changeUserBtn);

this->adminPanel->Controls->Add(this->addUserBtn);

this->adminPanel->Controls->Add(this->AdminLbl);

this->adminPanel->Location = System::Drawing::Point(12, 12);

this->adminPanel->Name = L"adminPanel";

this->adminPanel->Size = System::Drawing::Size(904, 725);

this->adminPanel->TabIndex = 2;

//

// mainTabSekector

//

this->mainTabSekector->Controls->Add(this->mainTab);

this->mainTabSekector->Controls->Add(this->addUserTab);

this->mainTabSekector->Controls->Add(this->changeUserTab);

this->mainTabSekector->Controls->Add(this->addServiceTab);

this->mainTabSekector->Controls->Add(this->changeServiceTab);

this->mainTabSekector->Controls->Add(this->showReportTab);

this->mainTabSekector->Location = System::Drawing::Point(302, 3);

this->mainTabSekector->Name = L"mainTabSekector";

this->mainTabSekector->SelectedIndex = 0;

this->mainTabSekector->Size = System::Drawing::Size(583, 712);

this->mainTabSekector->TabIndex = 33;

this->mainTabSekector->Selected += gcnew System::Windows::Forms::TabControlEventHandler(this, &adminForm::mainTabSekector\_Selected);

//

// mainTab

//

this->mainTab->BackColor = System::Drawing::Color::Gainsboro;

this->mainTab->Controls->Add(this->label1);

this->mainTab->Controls->Add(this->updateGridsBtn);

this->mainTab->Controls->Add(this->writeDataBtn);

this->mainTab->Controls->Add(this->loadBtn);

this->mainTab->Controls->Add(this->label2);

this->mainTab->Controls->Add(this->label3);

this->mainTab->Controls->Add(this->usersDataGrid);

this->mainTab->Controls->Add(this->servicesDataGrid);

this->mainTab->Controls->Add(this->ordersDataGrid);

this->mainTab->Cursor = System::Windows::Forms::Cursors::Arrow;

this->mainTab->Location = System::Drawing::Point(4, 22);

this->mainTab->Name = L"mainTab";

this->mainTab->Padding = System::Windows::Forms::Padding(3);

this->mainTab->Size = System::Drawing::Size(575, 686);

this->mainTab->TabIndex = 0;

this->mainTab->Text = L"Главная";

//

// label1

//

this->label1->AutoSize = true;

this->label1->Font = (gcnew System::Drawing::Font(L"Yu Gothic UI Semibold", 15.25F, System::Drawing::FontStyle::Bold));

this->label1->Location = System::Drawing::Point(6, 22);

this->label1->Name = L"label1";

this->label1->Size = System::Drawing::Size(155, 30);

this->label1->TabIndex = 13;

this->label1->Text = L"Пользователи";

//

// updateGridsBtn

//

this->updateGridsBtn->BackgroundImage = (cli::safe\_cast<System::Drawing::Image^>(resources->GetObject(L"updateGridsBtn.BackgroundImage")));

this->updateGridsBtn->BackgroundImageLayout = System::Windows::Forms::ImageLayout::Zoom;

this->updateGridsBtn->Location = System::Drawing::Point(511, 12);

this->updateGridsBtn->Name = L"updateGridsBtn";

this->updateGridsBtn->Size = System::Drawing::Size(40, 40);

this->updateGridsBtn->TabIndex = 12;

this->updateGridsBtn->UseVisualStyleBackColor = true;

this->updateGridsBtn->Click += gcnew System::EventHandler(this, &adminForm::updateGridsBtn\_Click);

//

// writeDataBtn

//

this->writeDataBtn->BackgroundImage = (cli::safe\_cast<System::Drawing::Image^>(resources->GetObject(L"writeDataBtn.BackgroundImage")));

this->writeDataBtn->BackgroundImageLayout = System::Windows::Forms::ImageLayout::Zoom;

this->writeDataBtn->Location = System::Drawing::Point(465, 12);

this->writeDataBtn->Name = L"writeDataBtn";

this->writeDataBtn->Size = System::Drawing::Size(40, 40);

this->writeDataBtn->TabIndex = 31;

this->writeDataBtn->UseVisualStyleBackColor = true;

this->writeDataBtn->Click += gcnew System::EventHandler(this, &adminForm::writeDataBtn\_Click);

//

// loadBtn

//

this->loadBtn->BackgroundImage = (cli::safe\_cast<System::Drawing::Image^>(resources->GetObject(L"loadBtn.BackgroundImage")));

this->loadBtn->BackgroundImageLayout = System::Windows::Forms::ImageLayout::Zoom;

this->loadBtn->Location = System::Drawing::Point(419, 12);

this->loadBtn->Name = L"loadBtn";

this->loadBtn->Size = System::Drawing::Size(40, 40);

this->loadBtn->TabIndex = 32;

this->loadBtn->UseVisualStyleBackColor = true;

this->loadBtn->Click += gcnew System::EventHandler(this, &adminForm::loadBtn\_Click);

//

// label2

//

this->label2->AutoSize = true;

this->label2->Font = (gcnew System::Drawing::Font(L"Yu Gothic UI Semibold", 15.25F, System::Drawing::FontStyle::Bold));

this->label2->Location = System::Drawing::Point(6, 235);

this->label2->Name = L"label2";

this->label2->Size = System::Drawing::Size(79, 30);

this->label2->TabIndex = 14;

this->label2->Text = L"Услуги";

//

// label3

//

this->label3->AutoSize = true;

this->label3->Font = (gcnew System::Drawing::Font(L"Yu Gothic UI Semibold", 15.25F, System::Drawing::FontStyle::Bold));

this->label3->Location = System::Drawing::Point(3, 460);

this->label3->Name = L"label3";

this->label3->Size = System::Drawing::Size(84, 30);

this->label3->TabIndex = 15;

this->label3->Text = L"Заказы";

//

// usersDataGrid

//

this->usersDataGrid->AllowUserToAddRows = false;

this->usersDataGrid->AllowUserToDeleteRows = false;

this->usersDataGrid->BackgroundColor = System::Drawing::SystemColors::ButtonHighlight;

this->usersDataGrid->ColumnHeadersHeightSizeMode = System::Windows::Forms::DataGridViewColumnHeadersHeightSizeMode::AutoSize;

this->usersDataGrid->Columns->AddRange(gcnew cli::array< System::Windows::Forms::DataGridViewColumn^ >(3) {

this->userID,

this->userName, this->userCurrentStatus

});

this->usersDataGrid->EditMode = System::Windows::Forms::DataGridViewEditMode::EditProgrammatically;

this->usersDataGrid->Location = System::Drawing::Point(8, 55);

this->usersDataGrid->MultiSelect = false;

this->usersDataGrid->Name = L"usersDataGrid";

this->usersDataGrid->Size = System::Drawing::Size(543, 168);

this->usersDataGrid->TabIndex = 7;

//

// userID

//

this->userID->HeaderText = L"ID";

this->userID->Name = L"userID";

//

// userName

//

this->userName->HeaderText = L"Name";

this->userName->Name = L"userName";

//

// userCurrentStatus

//

this->userCurrentStatus->HeaderText = L"Status";

this->userCurrentStatus->Name = L"userCurrentStatus";

//

// servicesDataGrid

//

this->servicesDataGrid->AllowUserToAddRows = false;

this->servicesDataGrid->AllowUserToDeleteRows = false;

this->servicesDataGrid->BackgroundColor = System::Drawing::SystemColors::ButtonHighlight;

this->servicesDataGrid->ColumnHeadersHeightSizeMode = System::Windows::Forms::DataGridViewColumnHeadersHeightSizeMode::AutoSize;

this->servicesDataGrid->Columns->AddRange(gcnew cli::array< System::Windows::Forms::DataGridViewColumn^ >(3) {

this->servicesID,

this->serviceName, this->Price

});

this->servicesDataGrid->Location = System::Drawing::Point(8, 268);

this->servicesDataGrid->MultiSelect = false;

this->servicesDataGrid->Name = L"servicesDataGrid";

this->servicesDataGrid->Size = System::Drawing::Size(543, 178);

this->servicesDataGrid->TabIndex = 8;

//

// servicesID

//

this->servicesID->HeaderText = L"ID";

this->servicesID->Name = L"servicesID";

//

// serviceName

//

this->serviceName->HeaderText = L"Name";

this->serviceName->Name = L"serviceName";

//

// Price

//

this->Price->HeaderText = L"Price";

this->Price->Name = L"Price";

//

// ordersDataGrid

//

this->ordersDataGrid->AllowUserToAddRows = false;

this->ordersDataGrid->AllowUserToDeleteRows = false;

this->ordersDataGrid->BackgroundColor = System::Drawing::SystemColors::ButtonHighlight;

this->ordersDataGrid->ColumnHeadersHeightSizeMode = System::Windows::Forms::DataGridViewColumnHeadersHeightSizeMode::AutoSize;

this->ordersDataGrid->Columns->AddRange(gcnew cli::array< System::Windows::Forms::DataGridViewColumn^ >(5) {

this->orderID,

this->employeeID, this->clientID, this->orderStatus, this->serviceInOrder

});

this->ordersDataGrid->Location = System::Drawing::Point(8, 493);

this->ordersDataGrid->MultiSelect = false;

this->ordersDataGrid->Name = L"ordersDataGrid";

this->ordersDataGrid->Size = System::Drawing::Size(543, 176);

this->ordersDataGrid->TabIndex = 9;

//

// orderID

//

this->orderID->HeaderText = L"ID";

this->orderID->Name = L"orderID";

//

// employeeID

//

this->employeeID->HeaderText = L"employee";

this->employeeID->Name = L"employeeID";

//

// clientID

//

this->clientID->HeaderText = L"client";

this->clientID->Name = L"clientID";

//

// orderStatus

//

this->orderStatus->HeaderText = L"status";

this->orderStatus->Name = L"orderStatus";

//

// serviceInOrder

//

this->serviceInOrder->HeaderText = L"service";

this->serviceInOrder->Name = L"serviceInOrder";

//

// addUserTab

//

this->addUserTab->BackColor = System::Drawing::Color::Gainsboro;

this->addUserTab->Controls->Add(this->userRegistrationPanel);

this->addUserTab->Location = System::Drawing::Point(4, 22);

this->addUserTab->Name = L"addUserTab";

this->addUserTab->Padding = System::Windows::Forms::Padding(3);

this->addUserTab->Size = System::Drawing::Size(575, 686);

this->addUserTab->TabIndex = 1;

this->addUserTab->Text = L"Добавить пользователя";

//

// userRegistrationPanel

//

this->userRegistrationPanel->Controls->Add(this->addUserPhonenumberTB);

this->userRegistrationPanel->Controls->Add(this->addUserDateTB);

this->userRegistrationPanel->Controls->Add(this->addUserSexGP);

this->userRegistrationPanel->Controls->Add(this->specialityPanel);

this->userRegistrationPanel->Controls->Add(this->addUserTypeCB);

this->userRegistrationPanel->Controls->Add(this->addUserTypeLbl);

this->userRegistrationPanel->Controls->Add(this->addUserFormBtn);

this->userRegistrationPanel->Controls->Add(this->addPassLbl);

this->userRegistrationPanel->Controls->Add(this->addEmailLbl);

this->userRegistrationPanel->Controls->Add(this->addPhonenumberLbl);

this->userRegistrationPanel->Controls->Add(this->addDateLbl);

this->userRegistrationPanel->Controls->Add(addNameLbl);

this->userRegistrationPanel->Controls->Add(this->addUserPasswordTB);

this->userRegistrationPanel->Controls->Add(this->addUserMailTB);

this->userRegistrationPanel->Controls->Add(this->addUserNameTB);

this->userRegistrationPanel->Controls->Add(this->addUser);

this->userRegistrationPanel->Font = (gcnew System::Drawing::Font(L"Yu Gothic UI Semibold", 9.25F, System::Drawing::FontStyle::Bold));

this->userRegistrationPanel->Location = System::Drawing::Point(6, 5);

this->userRegistrationPanel->Name = L"userRegistrationPanel";

this->userRegistrationPanel->Size = System::Drawing::Size(563, 675);

this->userRegistrationPanel->TabIndex = 26;

//

// addUserPhonenumberTB

//

this->addUserPhonenumberTB->Location = System::Drawing::Point(141, 201);

this->addUserPhonenumberTB->Mask = L"(999) 000-0000";

this->addUserPhonenumberTB->Name = L"addUserPhonenumberTB";

this->addUserPhonenumberTB->Size = System::Drawing::Size(124, 24);

this->addUserPhonenumberTB->TabIndex = 28;

//

// addUserDateTB

//

this->addUserDateTB->Location = System::Drawing::Point(141, 171);

this->addUserDateTB->Mask = L"00/00/0000";

this->addUserDateTB->Name = L"addUserDateTB";

this->addUserDateTB->Size = System::Drawing::Size(124, 24);

this->addUserDateTB->TabIndex = 28;

this->addUserDateTB->ValidatingType = System::DateTime::typeid;

//

// addUserSexGP

//

this->addUserSexGP->Controls->Add(this->addUserFemaleRB);

this->addUserSexGP->Controls->Add(this->addUserMaleRB);

this->addUserSexGP->Location = System::Drawing::Point(88, 78);

this->addUserSexGP->Name = L"addUserSexGP";

this->addUserSexGP->Size = System::Drawing::Size(175, 58);

this->addUserSexGP->TabIndex = 41;

this->addUserSexGP->TabStop = false;

this->addUserSexGP->Text = L"Пол";

//

// addUserFemaleRB

//

this->addUserFemaleRB->AutoSize = true;

this->addUserFemaleRB->Location = System::Drawing::Point(69, 33);

this->addUserFemaleRB->Name = L"addUserFemaleRB";

this->addUserFemaleRB->Size = System::Drawing::Size(82, 21);

this->addUserFemaleRB->TabIndex = 40;

this->addUserFemaleRB->TabStop = true;

this->addUserFemaleRB->Text = L"Женский";

this->addUserFemaleRB->UseVisualStyleBackColor = true;

//

// addUserMaleRB

//

this->addUserMaleRB->AutoSize = true;

this->addUserMaleRB->Location = System::Drawing::Point(69, 13);

this->addUserMaleRB->Name = L"addUserMaleRB";

this->addUserMaleRB->Size = System::Drawing::Size(84, 21);

this->addUserMaleRB->TabIndex = 39;

this->addUserMaleRB->TabStop = true;

this->addUserMaleRB->Text = L"Мужской";

this->addUserMaleRB->UseVisualStyleBackColor = true;

//

// specialityPanel

//

this->specialityPanel->Controls->Add(this->addStatusLbl);

this->specialityPanel->Controls->Add(this->addUserSpecialityCB);

this->specialityPanel->Location = System::Drawing::Point(42, 287);

this->specialityPanel->Name = L"specialityPanel";

this->specialityPanel->Size = System::Drawing::Size(242, 38);

this->specialityPanel->TabIndex = 38;

this->specialityPanel->Visible = false;

//

// addStatusLbl

//

this->addStatusLbl->Anchor = System::Windows::Forms::AnchorStyles::None;

this->addStatusLbl->AutoSize = true;

this->addStatusLbl->Location = System::Drawing::Point(16, 11);

this->addStatusLbl->Name = L"addStatusLbl";

this->addStatusLbl->Size = System::Drawing::Size(77, 17);

this->addStatusLbl->TabIndex = 33;

this->addStatusLbl->Text = L"Должность";

//

// addUserSpecialityCB

//

this->addUserSpecialityCB->Anchor = System::Windows::Forms::AnchorStyles::None;

this->addUserSpecialityCB->FormattingEnabled = true;

this->addUserSpecialityCB->Items->AddRange(gcnew cli::array< System::Object^ >(4) {

L"BARBER", L"ADMINISTRATOR", L"CLEANER",

L"DIRECTOR"

});

this->addUserSpecialityCB->Location = System::Drawing::Point(99, 10);

this->addUserSpecialityCB->Name = L"addUserSpecialityCB";

this->addUserSpecialityCB->Size = System::Drawing::Size(124, 23);

this->addUserSpecialityCB->TabIndex = 26;

//

// addUserTypeCB

//

this->addUserTypeCB->FormattingEnabled = true;

this->addUserTypeCB->Items->AddRange(gcnew cli::array< System::Object^ >(3) { L"ADMIN", L"EMPLOYEE", L"CLIENT" });

this->addUserTypeCB->Location = System::Drawing::Point(141, 142);

this->addUserTypeCB->Name = L"addUserTypeCB";

this->addUserTypeCB->Size = System::Drawing::Size(124, 23);

this->addUserTypeCB->TabIndex = 36;

this->addUserTypeCB->SelectedIndexChanged += gcnew System::EventHandler(this, &adminForm::addUserTypeCB\_SelectedIndexChanged);

this->addUserTypeCB->SelectionChangeCommitted += gcnew System::EventHandler(this, &adminForm::addUserTypeCB\_SelectionChangeCommitted);

//

// addUserTypeLbl

//

this->addUserTypeLbl->AutoSize = true;

this->addUserTypeLbl->Location = System::Drawing::Point(14, 145);

this->addUserTypeLbl->Name = L"addUserTypeLbl";

this->addUserTypeLbl->Size = System::Drawing::Size(120, 17);

this->addUserTypeLbl->TabIndex = 35;

this->addUserTypeLbl->Text = L"Тип пользователя";

//

// addUserFormBtn

//

this->addUserFormBtn->Cursor = System::Windows::Forms::Cursors::Hand;

this->addUserFormBtn->Location = System::Drawing::Point(141, 331);

this->addUserFormBtn->Name = L"addUserFormBtn";

this->addUserFormBtn->Size = System::Drawing::Size(124, 34);

this->addUserFormBtn->TabIndex = 34;

this->addUserFormBtn->Text = L"Добавить";

this->addUserFormBtn->UseVisualStyleBackColor = true;

this->addUserFormBtn->Click += gcnew System::EventHandler(this, &adminForm::addUserFormBtn\_Click);

//

// addPassLbl

//

this->addPassLbl->AutoSize = true;

this->addPassLbl->Location = System::Drawing::Point(80, 264);

this->addPassLbl->Name = L"addPassLbl";

this->addPassLbl->Size = System::Drawing::Size(55, 17);

this->addPassLbl->TabIndex = 32;

this->addPassLbl->Text = L"Пароль";

//

// addEmailLbl

//

this->addEmailLbl->AutoSize = true;

this->addEmailLbl->Location = System::Drawing::Point(88, 234);

this->addEmailLbl->Name = L"addEmailLbl";

this->addEmailLbl->Size = System::Drawing::Size(45, 17);

this->addEmailLbl->TabIndex = 31;

this->addEmailLbl->Text = L"e-mail";

//

// addPhonenumberLbl

//

this->addPhonenumberLbl->AutoSize = true;

this->addPhonenumberLbl->Location = System::Drawing::Point(21, 204);

this->addPhonenumberLbl->Name = L"addPhonenumberLbl";

this->addPhonenumberLbl->Size = System::Drawing::Size(114, 17);

this->addPhonenumberLbl->TabIndex = 30;

this->addPhonenumberLbl->Text = L"Номер телефона";

//

// addDateLbl

//

this->addDateLbl->AutoSize = true;

this->addDateLbl->Location = System::Drawing::Point(30, 174);

this->addDateLbl->Name = L"addDateLbl";

this->addDateLbl->Size = System::Drawing::Size(105, 17);

this->addDateLbl->TabIndex = 29;

this->addDateLbl->Text = L"Дата рождения";

//

// addUserPasswordTB

//

this->addUserPasswordTB->Location = System::Drawing::Point(141, 261);

this->addUserPasswordTB->Name = L"addUserPasswordTB";

this->addUserPasswordTB->Size = System::Drawing::Size(124, 24);

this->addUserPasswordTB->TabIndex = 25;

//

// addUserMailTB

//

this->addUserMailTB->Location = System::Drawing::Point(141, 231);

this->addUserMailTB->Name = L"addUserMailTB";

this->addUserMailTB->Size = System::Drawing::Size(124, 24);

this->addUserMailTB->TabIndex = 24;

//

// addUserNameTB

//

this->addUserNameTB->Location = System::Drawing::Point(141, 48);

this->addUserNameTB->Name = L"addUserNameTB";

this->addUserNameTB->Size = System::Drawing::Size(124, 24);

this->addUserNameTB->TabIndex = 19;

//

// addUser

//

this->addUser->AutoSize = true;

this->addUser->Font = (gcnew System::Drawing::Font(L"Yu Gothic UI Semibold", 15.25F, System::Drawing::FontStyle::Bold));

this->addUser->Location = System::Drawing::Point(30, 11);

this->addUser->Name = L"addUser";

this->addUser->Size = System::Drawing::Size(329, 30);

this->addUser->TabIndex = 37;

this->addUser->Text = L"Добавить нового пользователя";

//

// changeUserTab

//

this->changeUserTab->BackColor = System::Drawing::Color::Gainsboro;

this->changeUserTab->Controls->Add(this->changeUserPanel);

this->changeUserTab->Location = System::Drawing::Point(4, 22);

this->changeUserTab->Name = L"changeUserTab";

this->changeUserTab->Padding = System::Windows::Forms::Padding(3);

this->changeUserTab->Size = System::Drawing::Size(575, 686);

this->changeUserTab->TabIndex = 2;

this->changeUserTab->Text = L"Изменить пользователя";

//

// changeUserPanel

//

this->changeUserPanel->Controls->Add(this->changeUserPhonenumberTB);

this->changeUserPanel->Controls->Add(this->changeUserDateTB);

this->changeUserPanel->Controls->Add(this->changeUserSpecialityPanel);

this->changeUserPanel->Controls->Add(this->changeUserSexGP);

this->changeUserPanel->Controls->Add(this->changeUserIdCB);

this->changeUserPanel->Controls->Add(this->label9);

this->changeUserPanel->Controls->Add(this->label8);

this->changeUserPanel->Controls->Add(this->UserIdLbl);

this->changeUserPanel->Controls->Add(this->changeUserTypeCB);

this->changeUserPanel->Controls->Add(this->changeUserTypeLbl);

this->changeUserPanel->Controls->Add(this->changeUserFormBtn);

this->changeUserPanel->Controls->Add(this->changePassLbl);

this->changeUserPanel->Controls->Add(this->changeEmailLbl);

this->changeUserPanel->Controls->Add(this->changePhonenumberLbl);

this->changeUserPanel->Controls->Add(this->changeDateLbl);

this->changeUserPanel->Controls->Add(changeNameLbl);

this->changeUserPanel->Controls->Add(this->changeUserPasswordTB);

this->changeUserPanel->Controls->Add(this->changeUserMailTB);

this->changeUserPanel->Controls->Add(this->changeUserNameTB);

this->changeUserPanel->Font = (gcnew System::Drawing::Font(L"Yu Gothic UI Semibold", 9.25F, System::Drawing::FontStyle::Bold));

this->changeUserPanel->Location = System::Drawing::Point(3, 6);

this->changeUserPanel->Name = L"changeUserPanel";

this->changeUserPanel->Size = System::Drawing::Size(642, 694);

this->changeUserPanel->TabIndex = 22;

//

// changeUserPhonenumberTB

//

this->changeUserPhonenumberTB->Location = System::Drawing::Point(271, 202);

this->changeUserPhonenumberTB->Mask = L"(999) 000-0000";

this->changeUserPhonenumberTB->Name = L"changeUserPhonenumberTB";

this->changeUserPhonenumberTB->Size = System::Drawing::Size(121, 24);

this->changeUserPhonenumberTB->TabIndex = 62;

//

// changeUserDateTB

//

this->changeUserDateTB->Location = System::Drawing::Point(271, 176);

this->changeUserDateTB->Mask = L"00/00/0000";

this->changeUserDateTB->Name = L"changeUserDateTB";

this->changeUserDateTB->Size = System::Drawing::Size(121, 24);

this->changeUserDateTB->TabIndex = 63;

this->changeUserDateTB->ValidatingType = System::DateTime::typeid;

//

// changeUserSpecialityPanel

//

this->changeUserSpecialityPanel->Controls->Add(this->label10);

this->changeUserSpecialityPanel->Controls->Add(this->changePersonalAchievementsTB);

this->changeUserSpecialityPanel->Controls->Add(this->label7);

this->changeUserSpecialityPanel->Controls->Add(this->changeUserExperienceTB);

this->changeUserSpecialityPanel->Controls->Add(this->label4);

this->changeUserSpecialityPanel->Controls->Add(this->changeUserSpecialityCB);

this->changeUserSpecialityPanel->Location = System::Drawing::Point(73, 310);

this->changeUserSpecialityPanel->Name = L"changeUserSpecialityPanel";

this->changeUserSpecialityPanel->Size = System::Drawing::Size(344, 192);

this->changeUserSpecialityPanel->TabIndex = 61;

this->changeUserSpecialityPanel->Visible = false;

//

// label10

//

this->label10->AutoSize = true;

this->label10->Location = System::Drawing::Point(45, 63);

this->label10->Name = L"label10";

this->label10->Size = System::Drawing::Size(138, 17);

this->label10->TabIndex = 43;

this->label10->Text = L"Личные достижения";

//

// changePersonalAchievementsTB

//

this->changePersonalAchievementsTB->Location = System::Drawing::Point(197, 63);

this->changePersonalAchievementsTB->Multiline = true;

this->changePersonalAchievementsTB->Name = L"changePersonalAchievementsTB";

this->changePersonalAchievementsTB->Size = System::Drawing::Size(122, 75);

this->changePersonalAchievementsTB->TabIndex = 42;

//

// label7

//

this->label7->AutoSize = true;

this->label7->Location = System::Drawing::Point(144, 36);

this->label7->Name = L"label7";

this->label7->Size = System::Drawing::Size(39, 17);

this->label7->TabIndex = 41;

this->label7->Text = L"Стаж";

//

// changeUserExperienceTB

//

this->changeUserExperienceTB->Location = System::Drawing::Point(197, 33);

this->changeUserExperienceTB->Name = L"changeUserExperienceTB";

this->changeUserExperienceTB->Size = System::Drawing::Size(122, 24);

this->changeUserExperienceTB->TabIndex = 40;

//

// label4

//

this->label4->Anchor = System::Windows::Forms::AnchorStyles::None;

this->label4->AutoSize = true;

this->label4->Location = System::Drawing::Point(108, 9);

this->label4->Name = L"label4";

this->label4->Size = System::Drawing::Size(77, 17);

this->label4->TabIndex = 33;

this->label4->Text = L"Должность";

//

// changeUserSpecialityCB

//

this->changeUserSpecialityCB->Anchor = System::Windows::Forms::AnchorStyles::None;

this->changeUserSpecialityCB->FormattingEnabled = true;

this->changeUserSpecialityCB->Items->AddRange(gcnew cli::array< System::Object^ >(4) {

L"BARBER", L"ADMINISTRATOR", L"CLEANER",

L"DIRECTOR"

});

this->changeUserSpecialityCB->Location = System::Drawing::Point(197, 3);

this->changeUserSpecialityCB->Name = L"changeUserSpecialityCB";

this->changeUserSpecialityCB->Size = System::Drawing::Size(122, 23);

this->changeUserSpecialityCB->TabIndex = 26;

//

// changeUserSexGP

//

this->changeUserSexGP->Controls->Add(this->changeUserFemaleRB);

this->changeUserSexGP->Controls->Add(this->changeUserMaleRB);

this->changeUserSexGP->Location = System::Drawing::Point(217, 107);

this->changeUserSexGP->Name = L"changeUserSexGP";

this->changeUserSexGP->Size = System::Drawing::Size(175, 58);

this->changeUserSexGP->TabIndex = 60;

this->changeUserSexGP->TabStop = false;

this->changeUserSexGP->Text = L"Пол";

//

// changeUserFemaleRB

//

this->changeUserFemaleRB->AutoSize = true;

this->changeUserFemaleRB->Location = System::Drawing::Point(69, 33);

this->changeUserFemaleRB->Name = L"changeUserFemaleRB";

this->changeUserFemaleRB->Size = System::Drawing::Size(82, 21);

this->changeUserFemaleRB->TabIndex = 40;

this->changeUserFemaleRB->TabStop = true;

this->changeUserFemaleRB->Text = L"Женский";

this->changeUserFemaleRB->UseVisualStyleBackColor = true;

//

// changeUserMaleRB

//

this->changeUserMaleRB->AutoSize = true;

this->changeUserMaleRB->Location = System::Drawing::Point(69, 13);

this->changeUserMaleRB->Name = L"changeUserMaleRB";

this->changeUserMaleRB->Size = System::Drawing::Size(84, 21);

this->changeUserMaleRB->TabIndex = 39;

this->changeUserMaleRB->TabStop = true;

this->changeUserMaleRB->Text = L"Мужской";

this->changeUserMaleRB->UseVisualStyleBackColor = true;

//

// changeUserIdCB

//

this->changeUserIdCB->FormattingEnabled = true;

this->changeUserIdCB->Location = System::Drawing::Point(57, 112);

this->changeUserIdCB->Name = L"changeUserIdCB";

this->changeUserIdCB->Size = System::Drawing::Size(76, 23);

this->changeUserIdCB->TabIndex = 59;

this->changeUserIdCB->SelectionChangeCommitted += gcnew System::EventHandler(this, &adminForm::changeUserIdCB\_SelectionChangeCommitted);

//

// label9

//

this->label9->AutoSize = true;

this->label9->BorderStyle = System::Windows::Forms::BorderStyle::Fixed3D;

this->label9->Location = System::Drawing::Point(34, 71);

this->label9->Name = L"label9";

this->label9->Size = System::Drawing::Size(165, 36);

this->label9->TabIndex = 58;

this->label9->Text = L" Введите id\r\n для изменения данных";

//

// label8

//

this->label8->AutoSize = true;

this->label8->Font = (gcnew System::Drawing::Font(L"Yu Gothic UI Semibold", 15.25F, System::Drawing::FontStyle::Bold));

this->label8->Location = System::Drawing::Point(30, 12);

this->label8->Name = L"label8";

this->label8->Size = System::Drawing::Size(333, 30);

this->label8->TabIndex = 57;

this->label8->Text = L"Изменить данные пользователя";

//

// UserIdLbl

//

this->UserIdLbl->AutoSize = true;

this->UserIdLbl->Font = (gcnew System::Drawing::Font(L"Microsoft Sans Serif", 8.25F, System::Drawing::FontStyle::Bold, System::Drawing::GraphicsUnit::Point,

static\_cast<System::Byte>(204)));

this->UserIdLbl->Location = System::Drawing::Point(31, 117);

this->UserIdLbl->Name = L"UserIdLbl";

this->UserIdLbl->Size = System::Drawing::Size(20, 13);

this->UserIdLbl->TabIndex = 56;

this->UserIdLbl->Text = L"ID";

//

// changeUserTypeCB

//

this->changeUserTypeCB->FormattingEnabled = true;

this->changeUserTypeCB->Items->AddRange(gcnew cli::array< System::Object^ >(3) { L"ADMIN", L"EMPLOYEE", L"CLIENT" });

this->changeUserTypeCB->Location = System::Drawing::Point(271, 284);

this->changeUserTypeCB->Name = L"changeUserTypeCB";

this->changeUserTypeCB->Size = System::Drawing::Size(121, 23);

this->changeUserTypeCB->TabIndex = 54;

this->changeUserTypeCB->SelectedIndexChanged += gcnew System::EventHandler(this, &adminForm::changeUserTypeCB\_SelectedIndexChanged);

//

// changeUserTypeLbl

//

this->changeUserTypeLbl->AutoSize = true;

this->changeUserTypeLbl->Location = System::Drawing::Point(141, 286);

this->changeUserTypeLbl->Name = L"changeUserTypeLbl";

this->changeUserTypeLbl->Size = System::Drawing::Size(120, 17);

this->changeUserTypeLbl->TabIndex = 53;

this->changeUserTypeLbl->Text = L"Тип пользователя";

//

// changeUserFormBtn

//

this->changeUserFormBtn->Cursor = System::Windows::Forms::Cursors::Hand;

this->changeUserFormBtn->Location = System::Drawing::Point(436, 512);

this->changeUserFormBtn->Name = L"changeUserFormBtn";

this->changeUserFormBtn->Size = System::Drawing::Size(121, 37);

this->changeUserFormBtn->TabIndex = 52;

this->changeUserFormBtn->Text = L"Изменить";

this->changeUserFormBtn->UseVisualStyleBackColor = true;

this->changeUserFormBtn->Click += gcnew System::EventHandler(this, &adminForm::changeUserFormBtn\_Click);

//

// changePassLbl

//

this->changePassLbl->AutoSize = true;

this->changePassLbl->Location = System::Drawing::Point(201, 261);

this->changePassLbl->Name = L"changePassLbl";

this->changePassLbl->Size = System::Drawing::Size(55, 17);

this->changePassLbl->TabIndex = 50;

this->changePassLbl->Text = L"Пароль";

//

// changeEmailLbl

//

this->changeEmailLbl->AutoSize = true;

this->changeEmailLbl->Location = System::Drawing::Point(211, 233);

this->changeEmailLbl->Name = L"changeEmailLbl";

this->changeEmailLbl->Size = System::Drawing::Size(45, 17);

this->changeEmailLbl->TabIndex = 49;

this->changeEmailLbl->Text = L"e-mail";

//

// changePhonenumberLbl

//

this->changePhonenumberLbl->AutoSize = true;

this->changePhonenumberLbl->Location = System::Drawing::Point(144, 207);

this->changePhonenumberLbl->Name = L"changePhonenumberLbl";

this->changePhonenumberLbl->Size = System::Drawing::Size(114, 17);

this->changePhonenumberLbl->TabIndex = 48;

this->changePhonenumberLbl->Text = L"Номер телефона";

//

// changeDateLbl

//

this->changeDateLbl->AutoSize = true;

this->changeDateLbl->Location = System::Drawing::Point(151, 179);

this->changeDateLbl->Name = L"changeDateLbl";

this->changeDateLbl->Size = System::Drawing::Size(105, 17);

this->changeDateLbl->TabIndex = 47;

this->changeDateLbl->Text = L"Дата рождения";

//

// changeUserPasswordTB

//

this->changeUserPasswordTB->Location = System::Drawing::Point(271, 257);

this->changeUserPasswordTB->Name = L"changeUserPasswordTB";

this->changeUserPasswordTB->Size = System::Drawing::Size(120, 24);

this->changeUserPasswordTB->TabIndex = 43;

//

// changeUserMailTB

//

this->changeUserMailTB->Location = System::Drawing::Point(271, 230);

this->changeUserMailTB->Name = L"changeUserMailTB";

this->changeUserMailTB->Size = System::Drawing::Size(121, 24);

this->changeUserMailTB->TabIndex = 42;

//

// changeUserNameTB

//

this->changeUserNameTB->Location = System::Drawing::Point(271, 76);

this->changeUserNameTB->Name = L"changeUserNameTB";

this->changeUserNameTB->Size = System::Drawing::Size(121, 24);

this->changeUserNameTB->TabIndex = 37;

//

// addServiceTab

//

this->addServiceTab->BackColor = System::Drawing::Color::Gainsboro;

this->addServiceTab->Controls->Add(this->addServicePanel);

this->addServiceTab->Location = System::Drawing::Point(4, 22);

this->addServiceTab->Name = L"addServiceTab";

this->addServiceTab->Padding = System::Windows::Forms::Padding(3);

this->addServiceTab->Size = System::Drawing::Size(575, 686);

this->addServiceTab->TabIndex = 3;

this->addServiceTab->Text = L"Добавить услугу";

//

// addServicePanel

//

this->addServicePanel->Controls->Add(this->loadServicesBtn);

this->addServicePanel->Controls->Add(this->updateServicesBtn);

this->addServicePanel->Controls->Add(this->saveServicesBtn);

this->addServicePanel->Controls->Add(this->servicesRTB);

this->addServicePanel->Controls->Add(this->addServicePriceLbl);

this->addServicePanel->Controls->Add(this->addServices);

this->addServicePanel->Controls->Add(this->addServiceNameLbl);

this->addServicePanel->Controls->Add(this->serviceNameTB);

this->addServicePanel->Controls->Add(this->servicePriceTB);

this->addServicePanel->Controls->Add(this->addServiceBtn);

this->addServicePanel->Font = (gcnew System::Drawing::Font(L"Yu Gothic UI Semibold", 9.25F, System::Drawing::FontStyle::Bold));

this->addServicePanel->Location = System::Drawing::Point(3, 6);

this->addServicePanel->Name = L"addServicePanel";

this->addServicePanel->Size = System::Drawing::Size(566, 677);

this->addServicePanel->TabIndex = 27;

//

// loadServicesBtn

//

this->loadServicesBtn->BackgroundImage = (cli::safe\_cast<System::Drawing::Image^>(resources->GetObject(L"loadServicesBtn.BackgroundImage")));

this->loadServicesBtn->BackgroundImageLayout = System::Windows::Forms::ImageLayout::Zoom;

this->loadServicesBtn->Location = System::Drawing::Point(500, 134);

this->loadServicesBtn->Name = L"loadServicesBtn";

this->loadServicesBtn->Size = System::Drawing::Size(40, 39);

this->loadServicesBtn->TabIndex = 17;

this->loadServicesBtn->UseVisualStyleBackColor = true;

this->loadServicesBtn->Click += gcnew System::EventHandler(this, &adminForm::loadServicesBtn\_Click);

//

// updateServicesBtn

//

this->updateServicesBtn->BackgroundImage = (cli::safe\_cast<System::Drawing::Image^>(resources->GetObject(L"updateServicesBtn.BackgroundImage")));

this->updateServicesBtn->BackgroundImageLayout = System::Windows::Forms::ImageLayout::Zoom;

this->updateServicesBtn->Location = System::Drawing::Point(408, 134);

this->updateServicesBtn->Name = L"updateServicesBtn";

this->updateServicesBtn->Size = System::Drawing::Size(40, 40);

this->updateServicesBtn->TabIndex = 16;

this->updateServicesBtn->UseVisualStyleBackColor = true;

this->updateServicesBtn->Click += gcnew System::EventHandler(this, &adminForm::updateServicesBtn\_Click);

//

// saveServicesBtn

//

this->saveServicesBtn->BackgroundImage = (cli::safe\_cast<System::Drawing::Image^>(resources->GetObject(L"saveServicesBtn.BackgroundImage")));

this->saveServicesBtn->BackgroundImageLayout = System::Windows::Forms::ImageLayout::Zoom;

this->saveServicesBtn->Location = System::Drawing::Point(454, 134);

this->saveServicesBtn->Name = L"saveServicesBtn";

this->saveServicesBtn->Size = System::Drawing::Size(40, 40);

this->saveServicesBtn->TabIndex = 16;

this->saveServicesBtn->UseVisualStyleBackColor = true;

this->saveServicesBtn->Click += gcnew System::EventHandler(this, &adminForm::saveServicesBtn\_Click);

//

// servicesRTB

//

this->servicesRTB->Location = System::Drawing::Point(34, 180);

this->servicesRTB->Name = L"servicesRTB";

this->servicesRTB->Size = System::Drawing::Size(506, 143);

this->servicesRTB->TabIndex = 12;

this->servicesRTB->Text = L"";

//

// addServicePriceLbl

//

this->addServicePriceLbl->AutoSize = true;

this->addServicePriceLbl->Font = (gcnew System::Drawing::Font(L"Yu Gothic UI Semibold", 9.25F, System::Drawing::FontStyle::Bold));

this->addServicePriceLbl->Location = System::Drawing::Point(52, 103);

this->addServicePriceLbl->Name = L"addServicePriceLbl";

this->addServicePriceLbl->Size = System::Drawing::Size(40, 17);

this->addServicePriceLbl->TabIndex = 11;

this->addServicePriceLbl->Text = L"Цена";

this->addServicePriceLbl->TextAlign = System::Drawing::ContentAlignment::TopCenter;

//

// addServices

//

this->addServices->AutoSize = true;

this->addServices->Font = (gcnew System::Drawing::Font(L"Yu Gothic UI Semibold", 15.25F, System::Drawing::FontStyle::Bold));

this->addServices->Location = System::Drawing::Point(27, 21);

this->addServices->Name = L"addServices";

this->addServices->Size = System::Drawing::Size(178, 30);

this->addServices->TabIndex = 9;

this->addServices->Text = L"Добавить услугу";

//

// addServiceNameLbl

//

this->addServiceNameLbl->AutoSize = true;

this->addServiceNameLbl->Font = (gcnew System::Drawing::Font(L"Yu Gothic UI Semibold", 9.25F, System::Drawing::FontStyle::Bold));

this->addServiceNameLbl->Location = System::Drawing::Point(29, 70);

this->addServiceNameLbl->Name = L"addServiceNameLbl";

this->addServiceNameLbl->Size = System::Drawing::Size(68, 17);

this->addServiceNameLbl->TabIndex = 10;

this->addServiceNameLbl->Text = L"Название";

//

// serviceNameTB

//

this->serviceNameTB->Location = System::Drawing::Point(108, 67);

this->serviceNameTB->Name = L"serviceNameTB";

this->serviceNameTB->Size = System::Drawing::Size(204, 24);

this->serviceNameTB->TabIndex = 6;

//

// servicePriceTB

//

this->servicePriceTB->Location = System::Drawing::Point(108, 101);

this->servicePriceTB->Name = L"servicePriceTB";

this->servicePriceTB->Size = System::Drawing::Size(204, 24);

this->servicePriceTB->TabIndex = 7;

//

// addServiceBtn

//

this->addServiceBtn->Location = System::Drawing::Point(212, 134);

this->addServiceBtn->Name = L"addServiceBtn";

this->addServiceBtn->Size = System::Drawing::Size(100, 39);

this->addServiceBtn->TabIndex = 8;

this->addServiceBtn->Text = L"Добавить";

this->addServiceBtn->UseVisualStyleBackColor = true;

this->addServiceBtn->Click += gcnew System::EventHandler(this, &adminForm::addServiceBtn\_Click);

//

// changeServiceTab

//

this->changeServiceTab->BackColor = System::Drawing::Color::Gainsboro;

this->changeServiceTab->Controls->Add(this->changeServicePanel);

this->changeServiceTab->Location = System::Drawing::Point(4, 22);

this->changeServiceTab->Name = L"changeServiceTab";

this->changeServiceTab->Padding = System::Windows::Forms::Padding(3);

this->changeServiceTab->Size = System::Drawing::Size(575, 686);

this->changeServiceTab->TabIndex = 4;

this->changeServiceTab->Text = L"Изменить услугу";

//

// changeServicePanel

//

this->changeServicePanel->Controls->Add(this->deleteServices);

this->changeServicePanel->Controls->Add(this->changeServicePriceLbl);

this->changeServicePanel->Controls->Add(this->changeServiceNameLbl);

this->changeServicePanel->Controls->Add(this->changeServiceNameTB);

this->changeServicePanel->Controls->Add(this->changeServicePriceTB);

this->changeServicePanel->Controls->Add(this->changeServiceFormBtn);

this->changeServicePanel->Font = (gcnew System::Drawing::Font(L"Yu Gothic UI Semibold", 9.25F, System::Drawing::FontStyle::Bold));

this->changeServicePanel->Location = System::Drawing::Point(6, 6);

this->changeServicePanel->Name = L"changeServicePanel";

this->changeServicePanel->Size = System::Drawing::Size(563, 674);

this->changeServicePanel->TabIndex = 25;

//

// deleteServices

//

this->deleteServices->AutoSize = true;

this->deleteServices->Font = (gcnew System::Drawing::Font(L"Yu Gothic UI Semibold", 15.25F, System::Drawing::FontStyle::Bold));

this->deleteServices->Location = System::Drawing::Point(30, 18);

this->deleteServices->Name = L"deleteServices";

this->deleteServices->Size = System::Drawing::Size(179, 30);

this->deleteServices->TabIndex = 9;

this->deleteServices->Text = L"Изменить услугу";

//

// changeServicePriceLbl

//

this->changeServicePriceLbl->AutoSize = true;

this->changeServicePriceLbl->Font = (gcnew System::Drawing::Font(L"Yu Gothic UI Semibold", 9.25F, System::Drawing::FontStyle::Bold));

this->changeServicePriceLbl->Location = System::Drawing::Point(32, 105);

this->changeServicePriceLbl->Name = L"changeServicePriceLbl";

this->changeServicePriceLbl->Size = System::Drawing::Size(40, 17);

this->changeServicePriceLbl->TabIndex = 11;

this->changeServicePriceLbl->Text = L"Цена";

this->changeServicePriceLbl->TextAlign = System::Drawing::ContentAlignment::TopCenter;

//

// changeServiceNameLbl

//

this->changeServiceNameLbl->AutoSize = true;

this->changeServiceNameLbl->Font = (gcnew System::Drawing::Font(L"Yu Gothic UI Semibold", 9.25F, System::Drawing::FontStyle::Bold));

this->changeServiceNameLbl->Location = System::Drawing::Point(29, 67);

this->changeServiceNameLbl->Name = L"changeServiceNameLbl";

this->changeServiceNameLbl->Size = System::Drawing::Size(68, 17);

this->changeServiceNameLbl->TabIndex = 10;

this->changeServiceNameLbl->Text = L"Название";

//

// changeServiceNameTB

//

this->changeServiceNameTB->Location = System::Drawing::Point(103, 64);

this->changeServiceNameTB->Name = L"changeServiceNameTB";

this->changeServiceNameTB->Size = System::Drawing::Size(100, 24);

this->changeServiceNameTB->TabIndex = 6;

//

// changeServicePriceTB

//

this->changeServicePriceTB->Location = System::Drawing::Point(103, 98);

this->changeServicePriceTB->Name = L"changeServicePriceTB";

this->changeServicePriceTB->Size = System::Drawing::Size(100, 24);

this->changeServicePriceTB->TabIndex = 7;

//

// changeServiceFormBtn

//

this->changeServiceFormBtn->Location = System::Drawing::Point(30, 137);

this->changeServiceFormBtn->Name = L"changeServiceFormBtn";

this->changeServiceFormBtn->Size = System::Drawing::Size(173, 41);

this->changeServiceFormBtn->TabIndex = 8;

this->changeServiceFormBtn->Text = L"Изменить";

this->changeServiceFormBtn->UseVisualStyleBackColor = true;

this->changeServiceFormBtn->Click += gcnew System::EventHandler(this, &adminForm::changeServiceFormBtn\_Click);

//

// showReportTab

//

this->showReportTab->BackColor = System::Drawing::Color::Gainsboro;

this->showReportTab->Controls->Add(this->reportPanel);

this->showReportTab->Location = System::Drawing::Point(4, 22);

this->showReportTab->Name = L"showReportTab";

this->showReportTab->Padding = System::Windows::Forms::Padding(3);

this->showReportTab->Size = System::Drawing::Size(575, 686);

this->showReportTab->TabIndex = 5;

this->showReportTab->Text = L"Составить отчёт";

//

// reportPanel

//

this->reportPanel->Controls->Add(this->averageСheckCB);

this->reportPanel->Controls->Add(this->clientsNumberCB);

this->reportPanel->Controls->Add(this->profitCB);

this->reportPanel->Controls->Add(this->proceedsCB);

this->reportPanel->Controls->Add(this->ExpensesCB);

this->reportPanel->Controls->Add(this->makeReportRTB);

this->reportPanel->Controls->Add(this->makeReportBtn);

this->reportPanel->Controls->Add(this->reportPeriodCB);

this->reportPanel->Controls->Add(this->reportPeriodLbl);

this->reportPanel->Controls->Add(this->financialStatementsLbl);

this->reportPanel->Location = System::Drawing::Point(6, 6);

this->reportPanel->Name = L"reportPanel";

this->reportPanel->Size = System::Drawing::Size(563, 677);

this->reportPanel->TabIndex = 24;

//

// averageСheckCB

//

this->averageСheckCB->AutoSize = true;

this->averageСheckCB->Location = System::Drawing::Point(208, 148);

this->averageСheckCB->Name = L"averageСheckCB";

this->averageСheckCB->Size = System::Drawing::Size(89, 17);

this->averageСheckCB->TabIndex = 28;

this->averageСheckCB->Text = L"Средний чек";

this->averageСheckCB->UseVisualStyleBackColor = true;

//

// clientsNumberCB

//

this->clientsNumberCB->AutoSize = true;

this->clientsNumberCB->Location = System::Drawing::Point(208, 128);

this->clientsNumberCB->Name = L"clientsNumberCB";

this->clientsNumberCB->Size = System::Drawing::Size(135, 17);

this->clientsNumberCB->TabIndex = 27;

this->clientsNumberCB->Text = L"Количество клиентов";

this->clientsNumberCB->UseVisualStyleBackColor = true;

//

// profitCB

//

this->profitCB->AutoSize = true;

this->profitCB->Location = System::Drawing::Point(208, 110);

this->profitCB->Name = L"profitCB";

this->profitCB->Size = System::Drawing::Size(72, 17);

this->profitCB->TabIndex = 26;

this->profitCB->Text = L"Прибыль";

this->profitCB->UseVisualStyleBackColor = true;

//

// proceedsCB

//

this->proceedsCB->AutoSize = true;

this->proceedsCB->Location = System::Drawing::Point(208, 91);

this->proceedsCB->Name = L"proceedsCB";

this->proceedsCB->Size = System::Drawing::Size(69, 17);

this->proceedsCB->TabIndex = 25;

this->proceedsCB->Text = L"Выручка";

this->proceedsCB->UseVisualStyleBackColor = true;

//

// ExpensesCB

//

this->ExpensesCB->AutoSize = true;

this->ExpensesCB->Location = System::Drawing::Point(208, 70);

this->ExpensesCB->Name = L"ExpensesCB";

this->ExpensesCB->Size = System::Drawing::Size(69, 17);

this->ExpensesCB->TabIndex = 24;

this->ExpensesCB->Text = L"Затраты";

this->ExpensesCB->UseVisualStyleBackColor = true;

//

// makeReportRTB

//

this->makeReportRTB->Location = System::Drawing::Point(28, 181);

this->makeReportRTB->Name = L"makeReportRTB";

this->makeReportRTB->Size = System::Drawing::Size(391, 270);

this->makeReportRTB->TabIndex = 23;

this->makeReportRTB->Text = L"";

//

// makeReportBtn

//

this->makeReportBtn->Location = System::Drawing::Point(28, 100);

this->makeReportBtn->Name = L"makeReportBtn";

this->makeReportBtn->Size = System::Drawing::Size(168, 65);

this->makeReportBtn->TabIndex = 22;

this->makeReportBtn->Text = L"Составить отчёт";

this->makeReportBtn->UseVisualStyleBackColor = true;

this->makeReportBtn->Click += gcnew System::EventHandler(this, &adminForm::makeReportBtn\_Click);

//

// reportPeriodCB

//

this->reportPeriodCB->FormattingEnabled = true;

this->reportPeriodCB->Items->AddRange(gcnew cli::array< System::Object^ >(3) { L"День", L"Месяц", L"Год" });

this->reportPeriodCB->Location = System::Drawing::Point(75, 70);

this->reportPeriodCB->Name = L"reportPeriodCB";

this->reportPeriodCB->Size = System::Drawing::Size(121, 21);

this->reportPeriodCB->TabIndex = 19;

//

// reportPeriodLbl

//

this->reportPeriodLbl->AutoSize = true;

this->reportPeriodLbl->Location = System::Drawing::Point(25, 75);

this->reportPeriodLbl->Name = L"reportPeriodLbl";

this->reportPeriodLbl->Size = System::Drawing::Size(45, 13);

this->reportPeriodLbl->TabIndex = 18;

this->reportPeriodLbl->Text = L"Период";

//

// financialStatementsLbl

//

this->financialStatementsLbl->AutoSize = true;

this->financialStatementsLbl->Font = (gcnew System::Drawing::Font(L"Yu Gothic UI Semibold", 15.25F, System::Drawing::FontStyle::Bold));

this->financialStatementsLbl->Location = System::Drawing::Point(30, 22);

this->financialStatementsLbl->Name = L"financialStatementsLbl";

this->financialStatementsLbl->Size = System::Drawing::Size(249, 30);

this->financialStatementsLbl->TabIndex = 0;

this->financialStatementsLbl->Text = L"Финансовая отчётность";

//

// deleteSpecidicServiceOrderBtn

//

this->deleteSpecidicServiceOrderBtn->Font = (gcnew System::Drawing::Font(L"Yu Gothic UI Semibold", 15.25F, System::Drawing::FontStyle::Bold));

this->deleteSpecidicServiceOrderBtn->Location = System::Drawing::Point(13, 488);

this->deleteSpecidicServiceOrderBtn->Name = L"deleteSpecidicServiceOrderBtn";

this->deleteSpecidicServiceOrderBtn->Size = System::Drawing::Size(268, 70);

this->deleteSpecidicServiceOrderBtn->TabIndex = 30;

this->deleteSpecidicServiceOrderBtn->Text = L"Удалить запись на услугу";

this->deleteSpecidicServiceOrderBtn->UseVisualStyleBackColor = true;

this->deleteSpecidicServiceOrderBtn->Click += gcnew System::EventHandler(this, &adminForm::deleteSpecidicServiceOrderBtn\_Click);

//

// deleteSpecificUser

//

this->deleteSpecificUser->Font = (gcnew System::Drawing::Font(L"Yu Gothic UI Semibold", 15.25F, System::Drawing::FontStyle::Bold));

this->deleteSpecificUser->Location = System::Drawing::Point(13, 216);

this->deleteSpecificUser->Name = L"deleteSpecificUser";

this->deleteSpecificUser->Size = System::Drawing::Size(268, 43);

this->deleteSpecificUser->TabIndex = 29;

this->deleteSpecificUser->Text = L"Удалить пользователя";

this->deleteSpecificUser->UseVisualStyleBackColor = true;

this->deleteSpecificUser->Click += gcnew System::EventHandler(this, &adminForm::deleteSpecificUser\_Click);

//

// showReportBtn

//

this->showReportBtn->Font = (gcnew System::Drawing::Font(L"Yu Gothic UI Semibold", 15.25F, System::Drawing::FontStyle::Bold));

this->showReportBtn->Location = System::Drawing::Point(13, 575);

this->showReportBtn->Name = L"showReportBtn";

this->showReportBtn->Size = System::Drawing::Size(268, 70);

this->showReportBtn->TabIndex = 28;

this->showReportBtn->Text = L"Показать/составить отчёт";

this->showReportBtn->UseVisualStyleBackColor = true;

this->showReportBtn->Click += gcnew System::EventHandler(this, &adminForm::showReportBtn\_Click);

//

// changeServiceBtn

//

this->changeServiceBtn->Font = (gcnew System::Drawing::Font(L"Yu Gothic UI Semibold", 15.25F, System::Drawing::FontStyle::Bold));

this->changeServiceBtn->Location = System::Drawing::Point(13, 420);

this->changeServiceBtn->Name = L"changeServiceBtn";

this->changeServiceBtn->Size = System::Drawing::Size(268, 43);

this->changeServiceBtn->TabIndex = 27;

this->changeServiceBtn->Text = L"Изменить услугу";

this->changeServiceBtn->UseVisualStyleBackColor = true;

this->changeServiceBtn->Click += gcnew System::EventHandler(this, &adminForm::changeServiceBtn\_Click);

//

// addServiceBtnFromAdminPanel

//

this->addServiceBtnFromAdminPanel->Font = (gcnew System::Drawing::Font(L"Yu Gothic UI Semibold", 15.25F, System::Drawing::FontStyle::Bold));

this->addServiceBtnFromAdminPanel->Location = System::Drawing::Point(13, 352);

this->addServiceBtnFromAdminPanel->Name = L"addServiceBtnFromAdminPanel";

this->addServiceBtnFromAdminPanel->Size = System::Drawing::Size(268, 43);

this->addServiceBtnFromAdminPanel->TabIndex = 26;

this->addServiceBtnFromAdminPanel->Text = L"Добавить услугу";

this->addServiceBtnFromAdminPanel->UseVisualStyleBackColor = true;

this->addServiceBtnFromAdminPanel->Click += gcnew System::EventHandler(this, &adminForm::addServiceBtnFromAdminPanel\_Click);

//

// delOrderBtn

//

this->delOrderBtn->Font = (gcnew System::Drawing::Font(L"Yu Gothic UI Semibold", 15.25F, System::Drawing::FontStyle::Bold));

this->delOrderBtn->Location = System::Drawing::Point(13, 284);

this->delOrderBtn->Name = L"delOrderBtn";

this->delOrderBtn->Size = System::Drawing::Size(268, 43);

this->delOrderBtn->TabIndex = 25;

this->delOrderBtn->Text = L"Удалить услугу";

this->delOrderBtn->UseVisualStyleBackColor = true;

this->delOrderBtn->Click += gcnew System::EventHandler(this, &adminForm::delOrderBtn\_Click);

//

// changeUserBtn

//

this->changeUserBtn->Font = (gcnew System::Drawing::Font(L"Yu Gothic UI Semibold", 15.25F, System::Drawing::FontStyle::Bold));

this->changeUserBtn->Location = System::Drawing::Point(13, 148);

this->changeUserBtn->Name = L"changeUserBtn";

this->changeUserBtn->Size = System::Drawing::Size(268, 43);

this->changeUserBtn->TabIndex = 24;

this->changeUserBtn->Text = L"Изменить пользователя";

this->changeUserBtn->UseVisualStyleBackColor = true;

this->changeUserBtn->Click += gcnew System::EventHandler(this, &adminForm::changeUserBtn\_Click);

//

// addUserBtn

//

this->addUserBtn->Font = (gcnew System::Drawing::Font(L"Yu Gothic UI Semibold", 15.25F, System::Drawing::FontStyle::Bold));

this->addUserBtn->Location = System::Drawing::Point(13, 80);

this->addUserBtn->Name = L"addUserBtn";

this->addUserBtn->Size = System::Drawing::Size(268, 43);

this->addUserBtn->TabIndex = 23;

this->addUserBtn->Text = L"Добавить пользователя";

this->addUserBtn->UseVisualStyleBackColor = true;

this->addUserBtn->Click += gcnew System::EventHandler(this, &adminForm::addUserBtn\_Click);

//

// AdminLbl

//

this->AdminLbl->AutoSize = true;

this->AdminLbl->Location = System::Drawing::Point(13, 13);

this->AdminLbl->Name = L"AdminLbl";

this->AdminLbl->Size = System::Drawing::Size(194, 13);

this->AdminLbl->TabIndex = 0;

this->AdminLbl->Text = L"Панель управления администратора";

//

// adminForm

//

this->AutoScaleDimensions = System::Drawing::SizeF(6, 13);

this->AutoScaleMode = System::Windows::Forms::AutoScaleMode::Font;

this->ClientSize = System::Drawing::Size(898, 732);

this->Controls->Add(this->adminPanel);

this->Name = L"adminForm";

this->Text = L"Админ";

this->FormClosing += gcnew System::Windows::Forms::FormClosingEventHandler(this, &adminForm::adminForm\_FormClosing);

this->adminPanel->ResumeLayout(false);

this->adminPanel->PerformLayout();

this->mainTabSekector->ResumeLayout(false);

this->mainTab->ResumeLayout(false);

this->mainTab->PerformLayout();

(cli::safe\_cast<System::ComponentModel::ISupportInitialize^>(this->usersDataGrid))->EndInit();

(cli::safe\_cast<System::ComponentModel::ISupportInitialize^>(this->servicesDataGrid))->EndInit();

(cli::safe\_cast<System::ComponentModel::ISupportInitialize^>(this->ordersDataGrid))->EndInit();

this->addUserTab->ResumeLayout(false);

this->userRegistrationPanel->ResumeLayout(false);

this->userRegistrationPanel->PerformLayout();

this->addUserSexGP->ResumeLayout(false);

this->addUserSexGP->PerformLayout();

this->specialityPanel->ResumeLayout(false);

this->specialityPanel->PerformLayout();

this->changeUserTab->ResumeLayout(false);

this->changeUserPanel->ResumeLayout(false);

this->changeUserPanel->PerformLayout();

this->changeUserSpecialityPanel->ResumeLayout(false);

this->changeUserSpecialityPanel->PerformLayout();

this->changeUserSexGP->ResumeLayout(false);

this->changeUserSexGP->PerformLayout();

this->addServiceTab->ResumeLayout(false);

this->addServicePanel->ResumeLayout(false);

this->addServicePanel->PerformLayout();

this->changeServiceTab->ResumeLayout(false);

this->changeServicePanel->ResumeLayout(false);

this->changeServicePanel->PerformLayout();

this->showReportTab->ResumeLayout(false);

this->reportPanel->ResumeLayout(false);

this->reportPanel->PerformLayout();

this->ResumeLayout(false);

}

#pragma endregion

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

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

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

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

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

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

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

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

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

private: System::Void addUserTypeCB\_SelectionChangeCommitted(System::Object^ sender, System::EventArgs^ e);

private: System::Void addUserTypeCB\_SelectedIndexChanged(System::Object^ sender, System::EventArgs^ e);

private: System::Void changeUserIdCB\_SelectionChangeCommitted(System::Object^ sender, System::EventArgs^ e);

private: System::Void changeUserTypeCB\_SelectedIndexChanged(System::Object^ sender, System::EventArgs^ e);

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

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

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

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

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

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

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

private: System::Void clearAllGrids();

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

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

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

private: System::Void mainTabSekector\_Selected(System::Object^ sender, System::Windows::Forms::TabControlEventArgs^ e);

private: System::Void adminForm\_FormClosing(System::Object^ sender, System::Windows::Forms::FormClosingEventArgs^ e);

};

}

#endif ADMINFORM\_H

----------------

menuForm

----------------

//Timurziev

#ifndef MENUFORM\_H

#define MENUFORM\_H

#include "MockDatabase.h"

#include "checkFunctions.h"

namespace bshop {

using namespace System;

using namespace System::ComponentModel;

using namespace System::Collections;

using namespace System::Windows::Forms;

using namespace System::Data;

using namespace System::Drawing;

/// <summary>

/// Сводка для menuForm

/// </summary>

public ref class menuForm : public System::Windows::Forms::Form

{

public:

menuForm(int ID, int userStatus)

{

InitializeComponent();

this->ID = ID;

this->userStatus = userStatus;

}

menuForm(void)

{

InitializeComponent();

}

protected:

/// <summary>

/// Освободить все используемые ресурсы.

/// </summary>

~menuForm()

{

if (components)

{

delete components;

}

}

private:

int ID;

int userStatus;

private: System::Windows::Forms::Panel^ adminPanel;

//MockDatabase\* db = MockDatabase::getInstance();

private: System::Windows::Forms::DataGridView^ ordersDataGrid;

private: System::Windows::Forms::DataGridView^ servicesDataGrid;

private: System::Windows::Forms::DataGridView^ usersDataGrid;

private: System::Windows::Forms::Label^ AdminLbl;

private: System::Windows::Forms::Label^ label3;

private: System::Windows::Forms::Label^ label2;

private: System::Windows::Forms::Label^ label1;

private: System::Windows::Forms::Button^ updateGridsBtn;

private: System::Windows::Forms::Panel^ clientPanel;

private: System::Windows::Forms::Label^ ClientNameLbl;

private: System::Windows::Forms::Label^ ordersList;

private: System::Windows::Forms::Button^ exitFromClientPanel;

private: System::Windows::Forms::Label^ selectDateLbl;

private: System::Windows::Forms::Label^ timeStatusLbl;

private: System::Windows::Forms::ListBox^ availableTimeLB;

private: System::Windows::Forms::Button^ updateServiceOrderBtn;

private: System::Windows::Forms::MonthCalendar^ Calendar;

private: System::Windows::Forms::TextBox^ costTB;

private: System::Windows::Forms::ComboBox^ employeeIDCB;

private: System::Windows::Forms::ComboBox^ serviceIDCB;

private: System::Windows::Forms::Label^ orderCostLbl;

private: System::Windows::Forms::DataGridView^ orderInitGrid;

private: System::Windows::Forms::DataGridViewTextBoxColumn^ serviceID;

private: System::Windows::Forms::DataGridViewTextBoxColumn^ employeeOerderID;

private: System::Windows::Forms::DataGridViewTextBoxColumn^ clientIDOrder;

private: System::Windows::Forms::DataGridViewTextBoxColumn^ cost;

private: System::Windows::Forms::DataGridViewTextBoxColumn^ sssorderStatus;

private: System::Windows::Forms::DataGridViewTextBoxColumn^ ordersDate;

private: System::Windows::Forms::DataGridViewTextBoxColumn^ orderTime;

private: System::Windows::Forms::Label^ employeeIDFromOrderLbl;

private: System::Windows::Forms::Label^ serviceIDfromOrderLbl;

private: System::Windows::Forms::Panel^ employeePanel;

private: System::Windows::Forms::Label^ label13;

private: System::Windows::Forms::TextBox^ achievementsEmployeeTB;

private: System::Windows::Forms::TextBox^ specialityEmployeeTB;

private: System::Windows::Forms::TextBox^ nameEmployeeTB;

private: System::Windows::Forms::Label^ orderEmployeeLbl;

private: System::Windows::Forms::ComboBox^ orderEmployeeIdCB;

private: System::Windows::Forms::CheckBox^ orderEmployeeCheckCB;

private: System::Windows::Forms::DataGridView^ employeeDataGrid;

private: System::Windows::Forms::DataGridViewTextBoxColumn^ employeeIdDG;

private: System::Windows::Forms::DataGridViewTextBoxColumn^ dateEmployeeDG;

private: System::Windows::Forms::DataGridViewTextBoxColumn^ serviceEmployeeDG;

private: System::Windows::Forms::DataGridViewTextBoxColumn^ clientNameEmployeeDG;

private: System::Windows::Forms::DataGridViewTextBoxColumn^ costOfServiceEmployeeDG;

private: System::Windows::Forms::DataGridViewTextBoxColumn^ ordersTime;

private: System::Windows::Forms::Label^ achievementsEmployeeLbl;

private: System::Windows::Forms::Label^ specialityEmployeeLbl;

private: System::Windows::Forms::Label^ nameEmployeeLbl;

private: System::Windows::Forms::Button^ exitFromEmployeePanelBtn;

private: System::Windows::Forms::Label^ label6;

private: System::Windows::Forms::Label^ label5;

private: System::Windows::Forms::Label^ selectDataLbl;

private: System::Windows::Forms::Label^ selectedDateEmployeeLbl;

private: System::Windows::Forms::TextBox^ selectedDateEmployeeTB;

private: System::Windows::Forms::MonthCalendar^ employeeCalendar;

private: System::Windows::Forms::Button^ updateOrdersGridBtn;

private: System::Windows::Forms::DataGridViewTextBoxColumn^ orderCurStatus;

private: System::Windows::Forms::Button^ showReportBtn;

private: System::Windows::Forms::Button^ changeServiceBtn;

private: System::Windows::Forms::Button^ addServiceBtnFromAdminPanel;

private: System::Windows::Forms::Button^ delOrderBtn;

private: System::Windows::Forms::Button^ changeUserBtn;

private: System::Windows::Forms::Button^ addUserBtn;

private: System::Windows::Forms::Panel^ changeUserPanel;

private: System::Windows::Forms::Label^ label9;

private: System::Windows::Forms::Label^ label8;

private: System::Windows::Forms::Label^ UserIdLbl;

private: System::Windows::Forms::ComboBox^ changeUserTypeCB;

private: System::Windows::Forms::Label^ changeUserTypeLbl;

private: System::Windows::Forms::Button^ changeUserFormBtn;

private: System::Windows::Forms::Label^ changePassLbl;

private: System::Windows::Forms::Label^ changeEmailLbl;

private: System::Windows::Forms::Label^ changePhonenumberLbl;

private: System::Windows::Forms::Label^ changeDateLbl;

private: System::Windows::Forms::TextBox^ changeUserPasswordTB;

private: System::Windows::Forms::TextBox^ changeUserMailTB;

private: System::Windows::Forms::Panel^ reportPanel;

private: System::Windows::Forms::Button^ updateReportBtn;

private: System::Windows::Forms::RichTextBox^ makeReportRTB;

private: System::Windows::Forms::Button^ makeReportBtn;

private: System::Windows::Forms::ComboBox^ reportPeriodCB;

private: System::Windows::Forms::Label^ reportPeriodLbl;

private: System::Windows::Forms::Button^ saveReportBtn;

private: System::Windows::Forms::Label^ financialStatementsLbl;

private: System::Windows::Forms::Panel^ changeServicePanel;

private: System::Windows::Forms::Label^ changeServicePriceLbl;

private: System::Windows::Forms::Label^ deleteServices;

private: System::Windows::Forms::Label^ changeServiceNameLbl;

private: System::Windows::Forms::TextBox^ changeServiceNameTB;

private: System::Windows::Forms::TextBox^ changeServicePriceTB;

private: System::Windows::Forms::Button^ changeServiceFormBtn;

private: System::Windows::Forms::Panel^ addServicePanel;

private: System::Windows::Forms::Button^ loadServicesBtn;

private: System::Windows::Forms::Button^ updateServicesBtn;

private: System::Windows::Forms::Button^ saveServicesBtn;

private: System::Windows::Forms::RichTextBox^ servicesRTB;

private: System::Windows::Forms::Label^ addServicePriceLbl;

private: System::Windows::Forms::Label^ addServices;

private: System::Windows::Forms::Label^ addServiceNameLbl;

private: System::Windows::Forms::TextBox^ serviceNameTB;

private: System::Windows::Forms::TextBox^ servicePriceTB;

private: System::Windows::Forms::Button^ addServiceBtn;

private: System::Windows::Forms::RadioButton^ addUserMaleRB;

private: System::Windows::Forms::RadioButton^ addUserFemaleRB;

private: System::Windows::Forms::GroupBox^ addUserSexGP;

private: System::Windows::Forms::TextBox^ addUserNameTB;

private: System::Windows::Forms::TextBox^ addUserMailTB;

private: System::Windows::Forms::TextBox^ addUserPasswordTB;

private: System::Windows::Forms::Label^ addDateLbl;

private: System::Windows::Forms::Label^ addPhonenumberLbl;

private: System::Windows::Forms::Label^ addEmailLbl;

private: System::Windows::Forms::Label^ addPassLbl;

private: System::Windows::Forms::Button^ addUserFormBtn;

private: System::Windows::Forms::Label^ addUserTypeLbl;

private: System::Windows::Forms::Label^ addUser;

private: System::Windows::Forms::Panel^ specialityPanel;

private: System::Windows::Forms::Panel^ userRegistrationPanel;

private: System::Windows::Forms::ComboBox^ changeUserIdCB;

private: System::Windows::Forms::MaskedTextBox^ addUserPhonenumberTB;

private: System::Windows::Forms::MaskedTextBox^ addUserDateTB;

private: System::Windows::Forms::GroupBox^ changeUserSexGP;

private: System::Windows::Forms::RadioButton^ changeUserFemaleRB;

private: System::Windows::Forms::RadioButton^ changeUserMaleRB;

private: System::Windows::Forms::MaskedTextBox^ changeUserPhonenumberTB;

private: System::Windows::Forms::MaskedTextBox^ changeUserDateTB;

private: System::Windows::Forms::Panel^ changeUserSpecialityPanel;

private: System::Windows::Forms::Label^ label4;

private: System::Windows::Forms::ComboBox^ changeUserSpecialityCB;

private: System::Windows::Forms::TextBox^ changeUserNameTB;

private: System::Windows::Forms::Label^ addStatusLbl;

private: System::Windows::Forms::ComboBox^ addUserSpecialityCB;

private: System::Windows::Forms::ComboBox^ addUserTypeCB;

private: System::Windows::Forms::Label^ label10;

private: System::Windows::Forms::TextBox^ changePersonalAchievementsTB;

private: System::Windows::Forms::Label^ label7;

private: System::Windows::Forms::TextBox^ changeUserExperienceTB;

private: System::Windows::Forms::DataGridViewTextBoxColumn^ orderID;

private: System::Windows::Forms::DataGridViewTextBoxColumn^ employeeID;

private: System::Windows::Forms::DataGridViewTextBoxColumn^ clientID;

private: System::Windows::Forms::DataGridViewTextBoxColumn^ orderStatus;

private: System::Windows::Forms::DataGridViewTextBoxColumn^ servicesID;

private: System::Windows::Forms::DataGridViewTextBoxColumn^ serviceName;

private: System::Windows::Forms::DataGridViewTextBoxColumn^ userID;

private: System::Windows::Forms::DataGridViewTextBoxColumn^ userName;

private: System::Windows::Forms::DataGridViewTextBoxColumn^ userCurrentStatus;

private: System::Windows::Forms::Button^ deleteSpecificUser;

private: System::Windows::Forms::Button^ deleteSpecidicServiceOrderBtn;

private: System::Windows::Forms::CheckBox^ averageСheckCB;

private: System::Windows::Forms::CheckBox^ clientsNumberCB;

private: System::Windows::Forms::CheckBox^ profitCB;

private: System::Windows::Forms::CheckBox^ proceedsCB;

private: System::Windows::Forms::CheckBox^ ExpensesCB;

/// <summary>

/// Обязательная переменная конструктора.

/// </summary>

System::ComponentModel::Container ^components;

#pragma region Windows Form Designer generated code

/// <summary>

/// Требуемый метод для поддержки конструктора — не изменяйте

/// содержимое этого метода с помощью редактора кода.

/// </summary>

void InitializeComponent(void)

{

System::Windows::Forms::Button^ acceptOrderBtn;

System::Windows::Forms::Label^ changeNameLbl;

System::Windows::Forms::Label^ addNameLbl;

System::ComponentModel::ComponentResourceManager^ resources = (gcnew System::ComponentModel::ComponentResourceManager(menuForm::typeid));

this->adminPanel = (gcnew System::Windows::Forms::Panel());

this->deleteSpecidicServiceOrderBtn = (gcnew System::Windows::Forms::Button());

this->deleteSpecificUser = (gcnew System::Windows::Forms::Button());

this->showReportBtn = (gcnew System::Windows::Forms::Button());

this->changeServiceBtn = (gcnew System::Windows::Forms::Button());

this->addServiceBtnFromAdminPanel = (gcnew System::Windows::Forms::Button());

this->delOrderBtn = (gcnew System::Windows::Forms::Button());

this->changeUserBtn = (gcnew System::Windows::Forms::Button());

this->addUserBtn = (gcnew System::Windows::Forms::Button());

this->AdminLbl = (gcnew System::Windows::Forms::Label());

this->ordersDataGrid = (gcnew System::Windows::Forms::DataGridView());

this->orderID = (gcnew System::Windows::Forms::DataGridViewTextBoxColumn());

this->employeeID = (gcnew System::Windows::Forms::DataGridViewTextBoxColumn());

this->clientID = (gcnew System::Windows::Forms::DataGridViewTextBoxColumn());

this->orderStatus = (gcnew System::Windows::Forms::DataGridViewTextBoxColumn());

this->servicesDataGrid = (gcnew System::Windows::Forms::DataGridView());

this->servicesID = (gcnew System::Windows::Forms::DataGridViewTextBoxColumn());

this->serviceName = (gcnew System::Windows::Forms::DataGridViewTextBoxColumn());

this->usersDataGrid = (gcnew System::Windows::Forms::DataGridView());

this->userID = (gcnew System::Windows::Forms::DataGridViewTextBoxColumn());

this->userName = (gcnew System::Windows::Forms::DataGridViewTextBoxColumn());

this->userCurrentStatus = (gcnew System::Windows::Forms::DataGridViewTextBoxColumn());

this->label3 = (gcnew System::Windows::Forms::Label());

this->label2 = (gcnew System::Windows::Forms::Label());

this->label1 = (gcnew System::Windows::Forms::Label());

this->updateGridsBtn = (gcnew System::Windows::Forms::Button());

this->changeServicePanel = (gcnew System::Windows::Forms::Panel());

this->deleteServices = (gcnew System::Windows::Forms::Label());

this->changeServicePriceLbl = (gcnew System::Windows::Forms::Label());

this->changeServiceNameLbl = (gcnew System::Windows::Forms::Label());

this->changeServiceNameTB = (gcnew System::Windows::Forms::TextBox());

this->changeServicePriceTB = (gcnew System::Windows::Forms::TextBox());

this->changeServiceFormBtn = (gcnew System::Windows::Forms::Button());

this->changeUserPanel = (gcnew System::Windows::Forms::Panel());

this->changeUserPhonenumberTB = (gcnew System::Windows::Forms::MaskedTextBox());

this->changeUserDateTB = (gcnew System::Windows::Forms::MaskedTextBox());

this->changeUserSpecialityPanel = (gcnew System::Windows::Forms::Panel());

this->label10 = (gcnew System::Windows::Forms::Label());

this->changePersonalAchievementsTB = (gcnew System::Windows::Forms::TextBox());

this->label7 = (gcnew System::Windows::Forms::Label());

this->changeUserExperienceTB = (gcnew System::Windows::Forms::TextBox());

this->label4 = (gcnew System::Windows::Forms::Label());

this->changeUserSpecialityCB = (gcnew System::Windows::Forms::ComboBox());

this->changeUserSexGP = (gcnew System::Windows::Forms::GroupBox());

this->changeUserFemaleRB = (gcnew System::Windows::Forms::RadioButton());

this->changeUserMaleRB = (gcnew System::Windows::Forms::RadioButton());

this->changeUserIdCB = (gcnew System::Windows::Forms::ComboBox());

this->label9 = (gcnew System::Windows::Forms::Label());

this->label8 = (gcnew System::Windows::Forms::Label());

this->UserIdLbl = (gcnew System::Windows::Forms::Label());

this->changeUserTypeCB = (gcnew System::Windows::Forms::ComboBox());

this->changeUserTypeLbl = (gcnew System::Windows::Forms::Label());

this->changeUserFormBtn = (gcnew System::Windows::Forms::Button());

this->changePassLbl = (gcnew System::Windows::Forms::Label());

this->changeEmailLbl = (gcnew System::Windows::Forms::Label());

this->changePhonenumberLbl = (gcnew System::Windows::Forms::Label());

this->changeDateLbl = (gcnew System::Windows::Forms::Label());

this->changeUserPasswordTB = (gcnew System::Windows::Forms::TextBox());

this->changeUserMailTB = (gcnew System::Windows::Forms::TextBox());

this->changeUserNameTB = (gcnew System::Windows::Forms::TextBox());

this->userRegistrationPanel = (gcnew System::Windows::Forms::Panel());

this->addUserPhonenumberTB = (gcnew System::Windows::Forms::MaskedTextBox());

this->addUserDateTB = (gcnew System::Windows::Forms::MaskedTextBox());

this->addUserSexGP = (gcnew System::Windows::Forms::GroupBox());

this->addUserFemaleRB = (gcnew System::Windows::Forms::RadioButton());

this->addUserMaleRB = (gcnew System::Windows::Forms::RadioButton());

this->specialityPanel = (gcnew System::Windows::Forms::Panel());

this->addStatusLbl = (gcnew System::Windows::Forms::Label());

this->addUserSpecialityCB = (gcnew System::Windows::Forms::ComboBox());

this->addUserTypeCB = (gcnew System::Windows::Forms::ComboBox());

this->addUserTypeLbl = (gcnew System::Windows::Forms::Label());

this->addUserFormBtn = (gcnew System::Windows::Forms::Button());

this->addPassLbl = (gcnew System::Windows::Forms::Label());

this->addEmailLbl = (gcnew System::Windows::Forms::Label());

this->addPhonenumberLbl = (gcnew System::Windows::Forms::Label());

this->addDateLbl = (gcnew System::Windows::Forms::Label());

this->addUserPasswordTB = (gcnew System::Windows::Forms::TextBox());

this->addUserMailTB = (gcnew System::Windows::Forms::TextBox());

this->addUserNameTB = (gcnew System::Windows::Forms::TextBox());

this->addUser = (gcnew System::Windows::Forms::Label());

this->reportPanel = (gcnew System::Windows::Forms::Panel());

this->averageСheckCB = (gcnew System::Windows::Forms::CheckBox());

this->clientsNumberCB = (gcnew System::Windows::Forms::CheckBox());

this->profitCB = (gcnew System::Windows::Forms::CheckBox());

this->proceedsCB = (gcnew System::Windows::Forms::CheckBox());

this->ExpensesCB = (gcnew System::Windows::Forms::CheckBox());

this->updateReportBtn = (gcnew System::Windows::Forms::Button());

this->makeReportRTB = (gcnew System::Windows::Forms::RichTextBox());

this->makeReportBtn = (gcnew System::Windows::Forms::Button());

this->reportPeriodCB = (gcnew System::Windows::Forms::ComboBox());

this->reportPeriodLbl = (gcnew System::Windows::Forms::Label());

this->saveReportBtn = (gcnew System::Windows::Forms::Button());

this->financialStatementsLbl = (gcnew System::Windows::Forms::Label());

this->addServicePanel = (gcnew System::Windows::Forms::Panel());

this->loadServicesBtn = (gcnew System::Windows::Forms::Button());

this->updateServicesBtn = (gcnew System::Windows::Forms::Button());

this->saveServicesBtn = (gcnew System::Windows::Forms::Button());

this->servicesRTB = (gcnew System::Windows::Forms::RichTextBox());

this->addServicePriceLbl = (gcnew System::Windows::Forms::Label());

this->addServices = (gcnew System::Windows::Forms::Label());

this->addServiceNameLbl = (gcnew System::Windows::Forms::Label());

this->serviceNameTB = (gcnew System::Windows::Forms::TextBox());

this->servicePriceTB = (gcnew System::Windows::Forms::TextBox());

this->addServiceBtn = (gcnew System::Windows::Forms::Button());

this->clientPanel = (gcnew System::Windows::Forms::Panel());

this->ClientNameLbl = (gcnew System::Windows::Forms::Label());

this->ordersList = (gcnew System::Windows::Forms::Label());

this->exitFromClientPanel = (gcnew System::Windows::Forms::Button());

this->timeStatusLbl = (gcnew System::Windows::Forms::Label());

this->availableTimeLB = (gcnew System::Windows::Forms::ListBox());

this->updateServiceOrderBtn = (gcnew System::Windows::Forms::Button());

this->Calendar = (gcnew System::Windows::Forms::MonthCalendar());

this->costTB = (gcnew System::Windows::Forms::TextBox());

this->employeeIDCB = (gcnew System::Windows::Forms::ComboBox());

this->serviceIDCB = (gcnew System::Windows::Forms::ComboBox());

this->selectDateLbl = (gcnew System::Windows::Forms::Label());

this->orderCostLbl = (gcnew System::Windows::Forms::Label());

this->orderInitGrid = (gcnew System::Windows::Forms::DataGridView());

this->serviceID = (gcnew System::Windows::Forms::DataGridViewTextBoxColumn());

this->employeeOerderID = (gcnew System::Windows::Forms::DataGridViewTextBoxColumn());

this->clientIDOrder = (gcnew System::Windows::Forms::DataGridViewTextBoxColumn());

this->cost = (gcnew System::Windows::Forms::DataGridViewTextBoxColumn());

this->sssorderStatus = (gcnew System::Windows::Forms::DataGridViewTextBoxColumn());

this->ordersDate = (gcnew System::Windows::Forms::DataGridViewTextBoxColumn());

this->orderTime = (gcnew System::Windows::Forms::DataGridViewTextBoxColumn());

this->employeeIDFromOrderLbl = (gcnew System::Windows::Forms::Label());

this->serviceIDfromOrderLbl = (gcnew System::Windows::Forms::Label());

this->employeePanel = (gcnew System::Windows::Forms::Panel());

this->updateOrdersGridBtn = (gcnew System::Windows::Forms::Button());

this->label6 = (gcnew System::Windows::Forms::Label());

this->label5 = (gcnew System::Windows::Forms::Label());

this->selectDataLbl = (gcnew System::Windows::Forms::Label());

this->selectedDateEmployeeLbl = (gcnew System::Windows::Forms::Label());

this->selectedDateEmployeeTB = (gcnew System::Windows::Forms::TextBox());

this->employeeCalendar = (gcnew System::Windows::Forms::MonthCalendar());

this->label13 = (gcnew System::Windows::Forms::Label());

this->achievementsEmployeeTB = (gcnew System::Windows::Forms::TextBox());

this->specialityEmployeeTB = (gcnew System::Windows::Forms::TextBox());

this->nameEmployeeTB = (gcnew System::Windows::Forms::TextBox());

this->orderEmployeeLbl = (gcnew System::Windows::Forms::Label());

this->orderEmployeeIdCB = (gcnew System::Windows::Forms::ComboBox());

this->orderEmployeeCheckCB = (gcnew System::Windows::Forms::CheckBox());

this->employeeDataGrid = (gcnew System::Windows::Forms::DataGridView());

this->employeeIdDG = (gcnew System::Windows::Forms::DataGridViewTextBoxColumn());

this->dateEmployeeDG = (gcnew System::Windows::Forms::DataGridViewTextBoxColumn());

this->serviceEmployeeDG = (gcnew System::Windows::Forms::DataGridViewTextBoxColumn());

this->clientNameEmployeeDG = (gcnew System::Windows::Forms::DataGridViewTextBoxColumn());

this->costOfServiceEmployeeDG = (gcnew System::Windows::Forms::DataGridViewTextBoxColumn());

this->ordersTime = (gcnew System::Windows::Forms::DataGridViewTextBoxColumn());

this->orderCurStatus = (gcnew System::Windows::Forms::DataGridViewTextBoxColumn());

this->achievementsEmployeeLbl = (gcnew System::Windows::Forms::Label());

this->specialityEmployeeLbl = (gcnew System::Windows::Forms::Label());

this->nameEmployeeLbl = (gcnew System::Windows::Forms::Label());

this->exitFromEmployeePanelBtn = (gcnew System::Windows::Forms::Button());

acceptOrderBtn = (gcnew System::Windows::Forms::Button());

changeNameLbl = (gcnew System::Windows::Forms::Label());

addNameLbl = (gcnew System::Windows::Forms::Label());

this->adminPanel->SuspendLayout();

(cli::safe\_cast<System::ComponentModel::ISupportInitialize^>(this->ordersDataGrid))->BeginInit();

(cli::safe\_cast<System::ComponentModel::ISupportInitialize^>(this->servicesDataGrid))->BeginInit();

(cli::safe\_cast<System::ComponentModel::ISupportInitialize^>(this->usersDataGrid))->BeginInit();

this->changeServicePanel->SuspendLayout();

this->changeUserPanel->SuspendLayout();

this->changeUserSpecialityPanel->SuspendLayout();

this->changeUserSexGP->SuspendLayout();

this->userRegistrationPanel->SuspendLayout();

this->addUserSexGP->SuspendLayout();

this->specialityPanel->SuspendLayout();

this->reportPanel->SuspendLayout();

this->addServicePanel->SuspendLayout();

this->clientPanel->SuspendLayout();

(cli::safe\_cast<System::ComponentModel::ISupportInitialize^>(this->orderInitGrid))->BeginInit();

this->employeePanel->SuspendLayout();

(cli::safe\_cast<System::ComponentModel::ISupportInitialize^>(this->employeeDataGrid))->BeginInit();

this->SuspendLayout();

//

// acceptOrderBtn

//

acceptOrderBtn->Location = System::Drawing::Point(580, 262);

acceptOrderBtn->Name = L"acceptOrderBtn";

acceptOrderBtn->Size = System::Drawing::Size(160, 36);

acceptOrderBtn->TabIndex = 13;

acceptOrderBtn->Text = L"Подтвердить";

acceptOrderBtn->UseVisualStyleBackColor = true;

acceptOrderBtn->Click += gcnew System::EventHandler(this, &menuForm::acceptOrderBtn\_Click);

//

// changeNameLbl

//

changeNameLbl->AutoSize = true;

changeNameLbl->Location = System::Drawing::Point(224, 83);

changeNameLbl->Name = L"changeNameLbl";

changeNameLbl->Size = System::Drawing::Size(35, 17);

changeNameLbl->TabIndex = 45;

changeNameLbl->Text = L"Имя";

//

// addNameLbl

//

addNameLbl->AutoSize = true;

addNameLbl->Location = System::Drawing::Point(98, 51);

addNameLbl->Name = L"addNameLbl";

addNameLbl->Size = System::Drawing::Size(35, 17);

addNameLbl->TabIndex = 27;

addNameLbl->Text = L"Имя";

//

// adminPanel

//

this->adminPanel->Controls->Add(this->deleteSpecidicServiceOrderBtn);

this->adminPanel->Controls->Add(this->deleteSpecificUser);

this->adminPanel->Controls->Add(this->showReportBtn);

this->adminPanel->Controls->Add(this->changeServiceBtn);

this->adminPanel->Controls->Add(this->addServiceBtnFromAdminPanel);

this->adminPanel->Controls->Add(this->delOrderBtn);

this->adminPanel->Controls->Add(this->changeUserBtn);

this->adminPanel->Controls->Add(this->addUserBtn);

this->adminPanel->Controls->Add(this->AdminLbl);

this->adminPanel->Controls->Add(this->ordersDataGrid);

this->adminPanel->Controls->Add(this->servicesDataGrid);

this->adminPanel->Controls->Add(this->usersDataGrid);

this->adminPanel->Controls->Add(this->label3);

this->adminPanel->Controls->Add(this->label2);

this->adminPanel->Controls->Add(this->label1);

this->adminPanel->Controls->Add(this->updateGridsBtn);

this->adminPanel->Location = System::Drawing::Point(990, 228);

this->adminPanel->Name = L"adminPanel";

this->adminPanel->Size = System::Drawing::Size(651, 709);

this->adminPanel->TabIndex = 1;

//

// deleteSpecidicServiceOrderBtn

//

this->deleteSpecidicServiceOrderBtn->Location = System::Drawing::Point(16, 260);

this->deleteSpecidicServiceOrderBtn->Name = L"deleteSpecidicServiceOrderBtn";

this->deleteSpecidicServiceOrderBtn->Size = System::Drawing::Size(153, 23);

this->deleteSpecidicServiceOrderBtn->TabIndex = 30;

this->deleteSpecidicServiceOrderBtn->Text = L"Удалить запись на услугу";

this->deleteSpecidicServiceOrderBtn->UseVisualStyleBackColor = true;

this->deleteSpecidicServiceOrderBtn->Click += gcnew System::EventHandler(this, &menuForm::deleteSpecidicServiceOrderBtn\_Click);

//

// deleteSpecificUser

//

this->deleteSpecificUser->Location = System::Drawing::Point(16, 144);

this->deleteSpecificUser->Name = L"deleteSpecificUser";

this->deleteSpecificUser->Size = System::Drawing::Size(153, 23);

this->deleteSpecificUser->TabIndex = 29;

this->deleteSpecificUser->Text = L"Удалить пользователя";

this->deleteSpecificUser->UseVisualStyleBackColor = true;

this->deleteSpecificUser->Click += gcnew System::EventHandler(this, &menuForm::deleteSpecificUser\_Click);

//

// showReportBtn

//

this->showReportBtn->Location = System::Drawing::Point(17, 290);

this->showReportBtn->Name = L"showReportBtn";

this->showReportBtn->Size = System::Drawing::Size(153, 23);

this->showReportBtn->TabIndex = 28;

this->showReportBtn->Text = L"Показать/составить отчёт";

this->showReportBtn->UseVisualStyleBackColor = true;

this->showReportBtn->Click += gcnew System::EventHandler(this, &menuForm::showReportBtn\_Click);

//

// changeServiceBtn

//

this->changeServiceBtn->Location = System::Drawing::Point(16, 231);

this->changeServiceBtn->Name = L"changeServiceBtn";

this->changeServiceBtn->Size = System::Drawing::Size(153, 23);

this->changeServiceBtn->TabIndex = 27;

this->changeServiceBtn->Text = L"Изменить услугу";

this->changeServiceBtn->UseVisualStyleBackColor = true;

this->changeServiceBtn->Click += gcnew System::EventHandler(this, &menuForm::changeServiceBtn\_Click);

//

// addServiceBtnFromAdminPanel

//

this->addServiceBtnFromAdminPanel->Location = System::Drawing::Point(16, 202);

this->addServiceBtnFromAdminPanel->Name = L"addServiceBtnFromAdminPanel";

this->addServiceBtnFromAdminPanel->Size = System::Drawing::Size(153, 23);

this->addServiceBtnFromAdminPanel->TabIndex = 26;

this->addServiceBtnFromAdminPanel->Text = L"Добавить услугу";

this->addServiceBtnFromAdminPanel->UseVisualStyleBackColor = true;

this->addServiceBtnFromAdminPanel->Click += gcnew System::EventHandler(this, &menuForm::addServiceBtnFromAdminPanel\_Click);

//

// delOrderBtn

//

this->delOrderBtn->Location = System::Drawing::Point(16, 173);

this->delOrderBtn->Name = L"delOrderBtn";

this->delOrderBtn->Size = System::Drawing::Size(153, 23);

this->delOrderBtn->TabIndex = 25;

this->delOrderBtn->Text = L"Удалить услугу";

this->delOrderBtn->UseVisualStyleBackColor = true;

this->delOrderBtn->Click += gcnew System::EventHandler(this, &menuForm::delOrderBtn\_Click);

//

// changeUserBtn

//

this->changeUserBtn->Location = System::Drawing::Point(16, 115);

this->changeUserBtn->Name = L"changeUserBtn";

this->changeUserBtn->Size = System::Drawing::Size(153, 23);

this->changeUserBtn->TabIndex = 24;

this->changeUserBtn->Text = L"Изменить пользователя";

this->changeUserBtn->UseVisualStyleBackColor = true;

this->changeUserBtn->Click += gcnew System::EventHandler(this, &menuForm::changeUserBtn\_Click);

//

// addUserBtn

//

this->addUserBtn->Location = System::Drawing::Point(16, 86);

this->addUserBtn->Name = L"addUserBtn";

this->addUserBtn->Size = System::Drawing::Size(153, 23);

this->addUserBtn->TabIndex = 23;

this->addUserBtn->Text = L"Добавить пользователя";

this->addUserBtn->UseVisualStyleBackColor = true;

this->addUserBtn->Click += gcnew System::EventHandler(this, &menuForm::addUserBtn\_Click\_1);

//

// AdminLbl

//

this->AdminLbl->AutoSize = true;

this->AdminLbl->Location = System::Drawing::Point(13, 13);

this->AdminLbl->Name = L"AdminLbl";

this->AdminLbl->Size = System::Drawing::Size(36, 13);

this->AdminLbl->TabIndex = 0;

this->AdminLbl->Text = L"Admin";

//

// ordersDataGrid

//

this->ordersDataGrid->AllowUserToAddRows = false;

this->ordersDataGrid->AllowUserToDeleteRows = false;

this->ordersDataGrid->BackgroundColor = System::Drawing::SystemColors::ButtonHighlight;

this->ordersDataGrid->ColumnHeadersHeightSizeMode = System::Windows::Forms::DataGridViewColumnHeadersHeightSizeMode::AutoSize;

this->ordersDataGrid->Columns->AddRange(gcnew cli::array< System::Windows::Forms::DataGridViewColumn^ >(4) {

this->orderID,

this->employeeID, this->clientID, this->orderStatus

});

this->ordersDataGrid->Location = System::Drawing::Point(292, 498);

this->ordersDataGrid->MultiSelect = false;

this->ordersDataGrid->Name = L"ordersDataGrid";

this->ordersDataGrid->Size = System::Drawing::Size(543, 176);

this->ordersDataGrid->TabIndex = 9;

//

// orderID

//

this->orderID->HeaderText = L"ID";

this->orderID->Name = L"orderID";

//

// employeeID

//

this->employeeID->HeaderText = L"employee";

this->employeeID->Name = L"employeeID";

//

// clientID

//

this->clientID->HeaderText = L"client";

this->clientID->Name = L"clientID";

//

// orderStatus

//

this->orderStatus->HeaderText = L"status";

this->orderStatus->Name = L"orderStatus";

//

// servicesDataGrid

//

this->servicesDataGrid->AllowUserToAddRows = false;

this->servicesDataGrid->AllowUserToDeleteRows = false;

this->servicesDataGrid->BackgroundColor = System::Drawing::SystemColors::ButtonHighlight;

this->servicesDataGrid->ColumnHeadersHeightSizeMode = System::Windows::Forms::DataGridViewColumnHeadersHeightSizeMode::AutoSize;

this->servicesDataGrid->Columns->AddRange(gcnew cli::array< System::Windows::Forms::DataGridViewColumn^ >(2) {

this->servicesID,

this->serviceName

});

this->servicesDataGrid->Location = System::Drawing::Point(292, 290);

this->servicesDataGrid->MultiSelect = false;

this->servicesDataGrid->Name = L"servicesDataGrid";

this->servicesDataGrid->Size = System::Drawing::Size(543, 178);

this->servicesDataGrid->TabIndex = 8;

//

// servicesID

//

this->servicesID->HeaderText = L"ID";

this->servicesID->Name = L"servicesID";

//

// serviceName

//

this->serviceName->HeaderText = L"Name";

this->serviceName->Name = L"serviceName";

//

// usersDataGrid

//

this->usersDataGrid->AllowUserToAddRows = false;

this->usersDataGrid->AllowUserToDeleteRows = false;

this->usersDataGrid->BackgroundColor = System::Drawing::SystemColors::ButtonHighlight;

this->usersDataGrid->ColumnHeadersHeightSizeMode = System::Windows::Forms::DataGridViewColumnHeadersHeightSizeMode::AutoSize;

this->usersDataGrid->Columns->AddRange(gcnew cli::array< System::Windows::Forms::DataGridViewColumn^ >(3) {

this->userID,

this->userName, this->userCurrentStatus

});

this->usersDataGrid->EditMode = System::Windows::Forms::DataGridViewEditMode::EditProgrammatically;

this->usersDataGrid->Location = System::Drawing::Point(292, 92);

this->usersDataGrid->MultiSelect = false;

this->usersDataGrid->Name = L"usersDataGrid";

this->usersDataGrid->Size = System::Drawing::Size(543, 168);

this->usersDataGrid->TabIndex = 7;

//

// userID

//

this->userID->HeaderText = L"ID";

this->userID->Name = L"userID";

//

// userName

//

this->userName->HeaderText = L"Name";

this->userName->Name = L"userName";

//

// userCurrentStatus

//

this->userCurrentStatus->HeaderText = L"Status";

this->userCurrentStatus->Name = L"userCurrentStatus";

//

// label3

//

this->label3->AutoSize = true;

this->label3->Location = System::Drawing::Point(292, 482);

this->label3->Name = L"label3";

this->label3->Size = System::Drawing::Size(36, 13);

this->label3->TabIndex = 15;

this->label3->Text = L"orders";

//

// label2

//

this->label2->AutoSize = true;

this->label2->Location = System::Drawing::Point(292, 274);

this->label2->Name = L"label2";

this->label2->Size = System::Drawing::Size(46, 13);

this->label2->TabIndex = 14;

this->label2->Text = L"services";

//

// label1

//

this->label1->AutoSize = true;

this->label1->Location = System::Drawing::Point(292, 73);

this->label1->Name = L"label1";

this->label1->Size = System::Drawing::Size(32, 13);

this->label1->TabIndex = 13;

this->label1->Text = L"users";

//

// updateGridsBtn

//

this->updateGridsBtn->BackgroundImage = (cli::safe\_cast<System::Drawing::Image^>(resources->GetObject(L"updateGridsBtn.BackgroundImage")));

this->updateGridsBtn->BackgroundImageLayout = System::Windows::Forms::ImageLayout::Zoom;

this->updateGridsBtn->Location = System::Drawing::Point(795, 31);

this->updateGridsBtn->Name = L"updateGridsBtn";

this->updateGridsBtn->Size = System::Drawing::Size(40, 40);

this->updateGridsBtn->TabIndex = 12;

this->updateGridsBtn->UseVisualStyleBackColor = true;

this->updateGridsBtn->Click += gcnew System::EventHandler(this, &menuForm::updateGridsBtn\_Click);

//

// changeServicePanel

//

this->changeServicePanel->Controls->Add(this->deleteServices);

this->changeServicePanel->Controls->Add(this->changeServicePriceLbl);

this->changeServicePanel->Controls->Add(this->changeServiceNameLbl);

this->changeServicePanel->Controls->Add(this->changeServiceNameTB);

this->changeServicePanel->Controls->Add(this->changeServicePriceTB);

this->changeServicePanel->Controls->Add(this->changeServiceFormBtn);

this->changeServicePanel->Font = (gcnew System::Drawing::Font(L"Yu Gothic UI Semibold", 9.25F, System::Drawing::FontStyle::Bold));

this->changeServicePanel->Location = System::Drawing::Point(990, 304);

this->changeServicePanel->Name = L"changeServicePanel";

this->changeServicePanel->Size = System::Drawing::Size(651, 692);

this->changeServicePanel->TabIndex = 25;

//

// deleteServices

//

this->deleteServices->AutoSize = true;

this->deleteServices->Font = (gcnew System::Drawing::Font(L"Yu Gothic UI Semibold", 15.25F, System::Drawing::FontStyle::Bold));

this->deleteServices->Location = System::Drawing::Point(23, 21);

this->deleteServices->Name = L"deleteServices";

this->deleteServices->Size = System::Drawing::Size(179, 30);

this->deleteServices->TabIndex = 9;

this->deleteServices->Text = L"Изменить услугу";

//

// changeServicePriceLbl

//

this->changeServicePriceLbl->AutoSize = true;

this->changeServicePriceLbl->Font = (gcnew System::Drawing::Font(L"Yu Gothic UI Semibold", 9.25F, System::Drawing::FontStyle::Bold));

this->changeServicePriceLbl->Location = System::Drawing::Point(25, 108);

this->changeServicePriceLbl->Name = L"changeServicePriceLbl";

this->changeServicePriceLbl->Size = System::Drawing::Size(40, 17);

this->changeServicePriceLbl->TabIndex = 11;

this->changeServicePriceLbl->Text = L"Цена";

this->changeServicePriceLbl->TextAlign = System::Drawing::ContentAlignment::TopCenter;

//

// changeServiceNameLbl

//

this->changeServiceNameLbl->AutoSize = true;

this->changeServiceNameLbl->Font = (gcnew System::Drawing::Font(L"Yu Gothic UI Semibold", 9.25F, System::Drawing::FontStyle::Bold));

this->changeServiceNameLbl->Location = System::Drawing::Point(22, 70);

this->changeServiceNameLbl->Name = L"changeServiceNameLbl";

this->changeServiceNameLbl->Size = System::Drawing::Size(68, 17);

this->changeServiceNameLbl->TabIndex = 10;

this->changeServiceNameLbl->Text = L"Название";

//

// changeServiceNameTB

//

this->changeServiceNameTB->Location = System::Drawing::Point(96, 67);

this->changeServiceNameTB->Name = L"changeServiceNameTB";

this->changeServiceNameTB->Size = System::Drawing::Size(100, 24);

this->changeServiceNameTB->TabIndex = 6;

//

// changeServicePriceTB

//

this->changeServicePriceTB->Location = System::Drawing::Point(96, 101);

this->changeServicePriceTB->Name = L"changeServicePriceTB";

this->changeServicePriceTB->Size = System::Drawing::Size(100, 24);

this->changeServicePriceTB->TabIndex = 7;

//

// changeServiceFormBtn

//

this->changeServiceFormBtn->Location = System::Drawing::Point(23, 140);

this->changeServiceFormBtn->Name = L"changeServiceFormBtn";

this->changeServiceFormBtn->Size = System::Drawing::Size(173, 23);

this->changeServiceFormBtn->TabIndex = 8;

this->changeServiceFormBtn->Text = L"Изменить";

this->changeServiceFormBtn->UseVisualStyleBackColor = true;

this->changeServiceFormBtn->Click += gcnew System::EventHandler(this, &menuForm::changeServiceFormBtn\_Click);

//

// changeUserPanel

//

this->changeUserPanel->Controls->Add(this->changeUserPhonenumberTB);

this->changeUserPanel->Controls->Add(this->changeUserDateTB);

this->changeUserPanel->Controls->Add(this->changeUserSpecialityPanel);

this->changeUserPanel->Controls->Add(this->changeUserSexGP);

this->changeUserPanel->Controls->Add(this->changeUserIdCB);

this->changeUserPanel->Controls->Add(this->label9);

this->changeUserPanel->Controls->Add(this->label8);

this->changeUserPanel->Controls->Add(this->UserIdLbl);

this->changeUserPanel->Controls->Add(this->changeUserTypeCB);

this->changeUserPanel->Controls->Add(this->changeUserTypeLbl);

this->changeUserPanel->Controls->Add(this->changeUserFormBtn);

this->changeUserPanel->Controls->Add(this->changePassLbl);

this->changeUserPanel->Controls->Add(this->changeEmailLbl);

this->changeUserPanel->Controls->Add(this->changePhonenumberLbl);

this->changeUserPanel->Controls->Add(this->changeDateLbl);

this->changeUserPanel->Controls->Add(changeNameLbl);

this->changeUserPanel->Controls->Add(this->changeUserPasswordTB);

this->changeUserPanel->Controls->Add(this->changeUserMailTB);

this->changeUserPanel->Controls->Add(this->changeUserNameTB);

this->changeUserPanel->Font = (gcnew System::Drawing::Font(L"Yu Gothic UI Semibold", 9.25F, System::Drawing::FontStyle::Bold));

this->changeUserPanel->Location = System::Drawing::Point(990, 633);

this->changeUserPanel->Name = L"changeUserPanel";

this->changeUserPanel->Size = System::Drawing::Size(651, 696);

this->changeUserPanel->TabIndex = 22;

this->changeUserPanel->VisibleChanged += gcnew System::EventHandler(this, &menuForm::changeUserPanel\_VisibleChanged);

//

// changeUserPhonenumberTB

//

this->changeUserPhonenumberTB->Location = System::Drawing::Point(271, 202);

this->changeUserPhonenumberTB->Mask = L"(999) 000-0000";

this->changeUserPhonenumberTB->Name = L"changeUserPhonenumberTB";

this->changeUserPhonenumberTB->Size = System::Drawing::Size(121, 24);

this->changeUserPhonenumberTB->TabIndex = 62;

//

// changeUserDateTB

//

this->changeUserDateTB->Location = System::Drawing::Point(271, 176);

this->changeUserDateTB->Mask = L"00/00/0000";

this->changeUserDateTB->Name = L"changeUserDateTB";

this->changeUserDateTB->Size = System::Drawing::Size(121, 24);

this->changeUserDateTB->TabIndex = 63;

this->changeUserDateTB->ValidatingType = System::DateTime::typeid;

//

// changeUserSpecialityPanel

//

this->changeUserSpecialityPanel->Controls->Add(this->label10);

this->changeUserSpecialityPanel->Controls->Add(this->changePersonalAchievementsTB);

this->changeUserSpecialityPanel->Controls->Add(this->label7);

this->changeUserSpecialityPanel->Controls->Add(this->changeUserExperienceTB);

this->changeUserSpecialityPanel->Controls->Add(this->label4);

this->changeUserSpecialityPanel->Controls->Add(this->changeUserSpecialityCB);

this->changeUserSpecialityPanel->Location = System::Drawing::Point(73, 310);

this->changeUserSpecialityPanel->Name = L"changeUserSpecialityPanel";

this->changeUserSpecialityPanel->Size = System::Drawing::Size(344, 192);

this->changeUserSpecialityPanel->TabIndex = 61;

this->changeUserSpecialityPanel->Visible = false;

//

// label10

//

this->label10->AutoSize = true;

this->label10->Location = System::Drawing::Point(45, 63);

this->label10->Name = L"label10";

this->label10->Size = System::Drawing::Size(138, 17);

this->label10->TabIndex = 43;

this->label10->Text = L"Личные достижения";

//

// changePersonalAchievementsTB

//

this->changePersonalAchievementsTB->Location = System::Drawing::Point(197, 63);

this->changePersonalAchievementsTB->Multiline = true;

this->changePersonalAchievementsTB->Name = L"changePersonalAchievementsTB";

this->changePersonalAchievementsTB->Size = System::Drawing::Size(122, 75);

this->changePersonalAchievementsTB->TabIndex = 42;

//

// label7

//

this->label7->AutoSize = true;

this->label7->Location = System::Drawing::Point(144, 36);

this->label7->Name = L"label7";

this->label7->Size = System::Drawing::Size(39, 17);

this->label7->TabIndex = 41;

this->label7->Text = L"Стаж";

//

// changeUserExperienceTB

//

this->changeUserExperienceTB->Location = System::Drawing::Point(197, 33);

this->changeUserExperienceTB->Name = L"changeUserExperienceTB";

this->changeUserExperienceTB->Size = System::Drawing::Size(122, 24);

this->changeUserExperienceTB->TabIndex = 40;

//

// label4

//

this->label4->Anchor = System::Windows::Forms::AnchorStyles::None;

this->label4->AutoSize = true;

this->label4->Location = System::Drawing::Point(108, 9);

this->label4->Name = L"label4";

this->label4->Size = System::Drawing::Size(77, 17);

this->label4->TabIndex = 33;

this->label4->Text = L"Должность";

//

// changeUserSpecialityCB

//

this->changeUserSpecialityCB->Anchor = System::Windows::Forms::AnchorStyles::None;

this->changeUserSpecialityCB->FormattingEnabled = true;

this->changeUserSpecialityCB->Items->AddRange(gcnew cli::array< System::Object^ >(4) {

L"BARBER", L"ADMINISTRATOR", L"CLEANER",

L"DIRECTOR"

});

this->changeUserSpecialityCB->Location = System::Drawing::Point(197, 3);

this->changeUserSpecialityCB->Name = L"changeUserSpecialityCB";

this->changeUserSpecialityCB->Size = System::Drawing::Size(122, 23);

this->changeUserSpecialityCB->TabIndex = 26;

//

// changeUserSexGP

//

this->changeUserSexGP->Controls->Add(this->changeUserFemaleRB);

this->changeUserSexGP->Controls->Add(this->changeUserMaleRB);

this->changeUserSexGP->Location = System::Drawing::Point(217, 107);

this->changeUserSexGP->Name = L"changeUserSexGP";

this->changeUserSexGP->Size = System::Drawing::Size(175, 58);

this->changeUserSexGP->TabIndex = 60;

this->changeUserSexGP->TabStop = false;

this->changeUserSexGP->Text = L"Пол";

//

// changeUserFemaleRB

//

this->changeUserFemaleRB->AutoSize = true;

this->changeUserFemaleRB->Location = System::Drawing::Point(69, 33);

this->changeUserFemaleRB->Name = L"changeUserFemaleRB";

this->changeUserFemaleRB->Size = System::Drawing::Size(82, 21);

this->changeUserFemaleRB->TabIndex = 40;

this->changeUserFemaleRB->TabStop = true;

this->changeUserFemaleRB->Text = L"Женский";

this->changeUserFemaleRB->UseVisualStyleBackColor = true;

//

// changeUserMaleRB

//

this->changeUserMaleRB->AutoSize = true;

this->changeUserMaleRB->Location = System::Drawing::Point(69, 13);

this->changeUserMaleRB->Name = L"changeUserMaleRB";

this->changeUserMaleRB->Size = System::Drawing::Size(84, 21);

this->changeUserMaleRB->TabIndex = 39;

this->changeUserMaleRB->TabStop = true;

this->changeUserMaleRB->Text = L"Мужской";

this->changeUserMaleRB->UseVisualStyleBackColor = true;

//

// changeUserIdCB

//

this->changeUserIdCB->FormattingEnabled = true;

this->changeUserIdCB->Location = System::Drawing::Point(57, 106);

this->changeUserIdCB->Name = L"changeUserIdCB";

this->changeUserIdCB->Size = System::Drawing::Size(76, 23);

this->changeUserIdCB->TabIndex = 59;

this->changeUserIdCB->SelectionChangeCommitted += gcnew System::EventHandler(this, &menuForm::changeUserIdCB\_SelectionChangeCommitted);

//

// label9

//

this->label9->AutoSize = true;

this->label9->BorderStyle = System::Windows::Forms::BorderStyle::Fixed3D;

this->label9->Location = System::Drawing::Point(34, 71);

this->label9->Name = L"label9";

this->label9->Size = System::Drawing::Size(165, 36);

this->label9->TabIndex = 58;

this->label9->Text = L" Введите id\r\n для изменения данных";

//

// label8

//

this->label8->AutoSize = true;

this->label8->Font = (gcnew System::Drawing::Font(L"Microsoft Sans Serif", 15.75F, System::Drawing::FontStyle::Regular, System::Drawing::GraphicsUnit::Point,

static\_cast<System::Byte>(204)));

this->label8->Location = System::Drawing::Point(68, 21);

this->label8->Name = L"label8";

this->label8->Size = System::Drawing::Size(335, 25);

this->label8->TabIndex = 57;

this->label8->Text = L"Изменить данные пользователя";

//

// UserIdLbl

//

this->UserIdLbl->AutoSize = true;

this->UserIdLbl->Font = (gcnew System::Drawing::Font(L"Microsoft Sans Serif", 8.25F, System::Drawing::FontStyle::Bold, System::Drawing::GraphicsUnit::Point,

static\_cast<System::Byte>(204)));

this->UserIdLbl->Location = System::Drawing::Point(31, 111);

this->UserIdLbl->Name = L"UserIdLbl";

this->UserIdLbl->Size = System::Drawing::Size(20, 13);

this->UserIdLbl->TabIndex = 56;

this->UserIdLbl->Text = L"ID";

//

// changeUserTypeCB

//

this->changeUserTypeCB->FormattingEnabled = true;

this->changeUserTypeCB->Items->AddRange(gcnew cli::array< System::Object^ >(3) { L"ADMIN", L"EMPLOYEE", L"CLIENT" });

this->changeUserTypeCB->Location = System::Drawing::Point(271, 284);

this->changeUserTypeCB->Name = L"changeUserTypeCB";

this->changeUserTypeCB->Size = System::Drawing::Size(121, 23);

this->changeUserTypeCB->TabIndex = 54;

this->changeUserTypeCB->SelectedIndexChanged += gcnew System::EventHandler(this, &menuForm::changeUserTypeCB\_SelectedIndexChanged);

//

// changeUserTypeLbl

//

this->changeUserTypeLbl->AutoSize = true;

this->changeUserTypeLbl->Location = System::Drawing::Point(141, 286);

this->changeUserTypeLbl->Name = L"changeUserTypeLbl";

this->changeUserTypeLbl->Size = System::Drawing::Size(120, 17);

this->changeUserTypeLbl->TabIndex = 53;

this->changeUserTypeLbl->Text = L"Тип пользователя";

//

// changeUserFormBtn

//

this->changeUserFormBtn->Cursor = System::Windows::Forms::Cursors::Hand;

this->changeUserFormBtn->Location = System::Drawing::Point(436, 512);

this->changeUserFormBtn->Name = L"changeUserFormBtn";

this->changeUserFormBtn->Size = System::Drawing::Size(121, 23);

this->changeUserFormBtn->TabIndex = 52;

this->changeUserFormBtn->Text = L"Изменить";

this->changeUserFormBtn->UseVisualStyleBackColor = true;

this->changeUserFormBtn->Click += gcnew System::EventHandler(this, &menuForm::changeUserFormBtn\_Click);

//

// changePassLbl

//

this->changePassLbl->AutoSize = true;

this->changePassLbl->Location = System::Drawing::Point(201, 261);

this->changePassLbl->Name = L"changePassLbl";

this->changePassLbl->Size = System::Drawing::Size(55, 17);

this->changePassLbl->TabIndex = 50;

this->changePassLbl->Text = L"Пароль";

//

// changeEmailLbl

//

this->changeEmailLbl->AutoSize = true;

this->changeEmailLbl->Location = System::Drawing::Point(211, 233);

this->changeEmailLbl->Name = L"changeEmailLbl";

this->changeEmailLbl->Size = System::Drawing::Size(45, 17);

this->changeEmailLbl->TabIndex = 49;

this->changeEmailLbl->Text = L"e-mail";

//

// changePhonenumberLbl

//

this->changePhonenumberLbl->AutoSize = true;

this->changePhonenumberLbl->Location = System::Drawing::Point(144, 207);

this->changePhonenumberLbl->Name = L"changePhonenumberLbl";

this->changePhonenumberLbl->Size = System::Drawing::Size(114, 17);

this->changePhonenumberLbl->TabIndex = 48;

this->changePhonenumberLbl->Text = L"Номер телефона";

//

// changeDateLbl

//

this->changeDateLbl->AutoSize = true;

this->changeDateLbl->Location = System::Drawing::Point(151, 179);

this->changeDateLbl->Name = L"changeDateLbl";

this->changeDateLbl->Size = System::Drawing::Size(105, 17);

this->changeDateLbl->TabIndex = 47;

this->changeDateLbl->Text = L"Дата рождения";

//

// changeUserPasswordTB

//

this->changeUserPasswordTB->Location = System::Drawing::Point(271, 257);

this->changeUserPasswordTB->Name = L"changeUserPasswordTB";

this->changeUserPasswordTB->Size = System::Drawing::Size(120, 24);

this->changeUserPasswordTB->TabIndex = 43;

//

// changeUserMailTB

//

this->changeUserMailTB->Location = System::Drawing::Point(271, 230);

this->changeUserMailTB->Name = L"changeUserMailTB";

this->changeUserMailTB->Size = System::Drawing::Size(121, 24);

this->changeUserMailTB->TabIndex = 42;

//

// changeUserNameTB

//

this->changeUserNameTB->Location = System::Drawing::Point(271, 76);

this->changeUserNameTB->Name = L"changeUserNameTB";

this->changeUserNameTB->Size = System::Drawing::Size(121, 24);

this->changeUserNameTB->TabIndex = 37;

//

// userRegistrationPanel

//

this->userRegistrationPanel->Controls->Add(this->addUserPhonenumberTB);

this->userRegistrationPanel->Controls->Add(this->addUserDateTB);

this->userRegistrationPanel->Controls->Add(this->addUserSexGP);

this->userRegistrationPanel->Controls->Add(this->specialityPanel);

this->userRegistrationPanel->Controls->Add(this->addUserTypeCB);

this->userRegistrationPanel->Controls->Add(this->addUserTypeLbl);

this->userRegistrationPanel->Controls->Add(this->addUserFormBtn);

this->userRegistrationPanel->Controls->Add(this->addPassLbl);

this->userRegistrationPanel->Controls->Add(this->addEmailLbl);

this->userRegistrationPanel->Controls->Add(this->addPhonenumberLbl);

this->userRegistrationPanel->Controls->Add(this->addDateLbl);

this->userRegistrationPanel->Controls->Add(addNameLbl);

this->userRegistrationPanel->Controls->Add(this->addUserPasswordTB);

this->userRegistrationPanel->Controls->Add(this->addUserMailTB);

this->userRegistrationPanel->Controls->Add(this->addUserNameTB);

this->userRegistrationPanel->Controls->Add(this->addUser);

this->userRegistrationPanel->Font = (gcnew System::Drawing::Font(L"Yu Gothic UI Semibold", 9.25F, System::Drawing::FontStyle::Bold));

this->userRegistrationPanel->Location = System::Drawing::Point(990, 562);

this->userRegistrationPanel->Name = L"userRegistrationPanel";

this->userRegistrationPanel->Size = System::Drawing::Size(651, 686);

this->userRegistrationPanel->TabIndex = 26;

//

// addUserPhonenumberTB

//

this->addUserPhonenumberTB->Location = System::Drawing::Point(141, 201);

this->addUserPhonenumberTB->Mask = L"(999) 000-0000";

this->addUserPhonenumberTB->Name = L"addUserPhonenumberTB";

this->addUserPhonenumberTB->Size = System::Drawing::Size(124, 24);

this->addUserPhonenumberTB->TabIndex = 28;

//

// addUserDateTB

//

this->addUserDateTB->Location = System::Drawing::Point(141, 171);

this->addUserDateTB->Mask = L"00/00/0000";

this->addUserDateTB->Name = L"addUserDateTB";

this->addUserDateTB->Size = System::Drawing::Size(124, 24);

this->addUserDateTB->TabIndex = 28;

this->addUserDateTB->ValidatingType = System::DateTime::typeid;

//

// addUserSexGP

//

this->addUserSexGP->Controls->Add(this->addUserFemaleRB);

this->addUserSexGP->Controls->Add(this->addUserMaleRB);

this->addUserSexGP->Location = System::Drawing::Point(88, 78);

this->addUserSexGP->Name = L"addUserSexGP";

this->addUserSexGP->Size = System::Drawing::Size(175, 58);

this->addUserSexGP->TabIndex = 41;

this->addUserSexGP->TabStop = false;

this->addUserSexGP->Text = L"Пол";

//

// addUserFemaleRB

//

this->addUserFemaleRB->AutoSize = true;

this->addUserFemaleRB->Location = System::Drawing::Point(69, 33);

this->addUserFemaleRB->Name = L"addUserFemaleRB";

this->addUserFemaleRB->Size = System::Drawing::Size(82, 21);

this->addUserFemaleRB->TabIndex = 40;

this->addUserFemaleRB->TabStop = true;

this->addUserFemaleRB->Text = L"Женский";

this->addUserFemaleRB->UseVisualStyleBackColor = true;

//

// addUserMaleRB

//

this->addUserMaleRB->AutoSize = true;

this->addUserMaleRB->Location = System::Drawing::Point(69, 13);

this->addUserMaleRB->Name = L"addUserMaleRB";

this->addUserMaleRB->Size = System::Drawing::Size(84, 21);

this->addUserMaleRB->TabIndex = 39;

this->addUserMaleRB->TabStop = true;

this->addUserMaleRB->Text = L"Мужской";

this->addUserMaleRB->UseVisualStyleBackColor = true;

//

// specialityPanel

//

this->specialityPanel->Controls->Add(this->addStatusLbl);

this->specialityPanel->Controls->Add(this->addUserSpecialityCB);

this->specialityPanel->Location = System::Drawing::Point(42, 287);

this->specialityPanel->Name = L"specialityPanel";

this->specialityPanel->Size = System::Drawing::Size(242, 38);

this->specialityPanel->TabIndex = 38;

this->specialityPanel->Visible = false;

//

// addStatusLbl

//

this->addStatusLbl->Anchor = System::Windows::Forms::AnchorStyles::None;

this->addStatusLbl->AutoSize = true;

this->addStatusLbl->Location = System::Drawing::Point(16, 11);

this->addStatusLbl->Name = L"addStatusLbl";

this->addStatusLbl->Size = System::Drawing::Size(77, 17);

this->addStatusLbl->TabIndex = 33;

this->addStatusLbl->Text = L"Должность";

//

// addUserSpecialityCB

//

this->addUserSpecialityCB->Anchor = System::Windows::Forms::AnchorStyles::None;

this->addUserSpecialityCB->FormattingEnabled = true;

this->addUserSpecialityCB->Items->AddRange(gcnew cli::array< System::Object^ >(4) {

L"BARBER", L"ADMINISTRATOR", L"CLEANER",

L"DIRECTOR"

});

this->addUserSpecialityCB->Location = System::Drawing::Point(99, 10);

this->addUserSpecialityCB->Name = L"addUserSpecialityCB";

this->addUserSpecialityCB->Size = System::Drawing::Size(124, 23);

this->addUserSpecialityCB->TabIndex = 26;

//

// addUserTypeCB

//

this->addUserTypeCB->FormattingEnabled = true;

this->addUserTypeCB->Items->AddRange(gcnew cli::array< System::Object^ >(3) { L"ADMIN", L"EMPLOYEE", L"CLIENT" });

this->addUserTypeCB->Location = System::Drawing::Point(141, 142);

this->addUserTypeCB->Name = L"addUserTypeCB";

this->addUserTypeCB->Size = System::Drawing::Size(124, 23);

this->addUserTypeCB->TabIndex = 36;

this->addUserTypeCB->SelectedIndexChanged += gcnew System::EventHandler(this, &menuForm::addUserTypeCB\_SelectedIndexChanged);

this->addUserTypeCB->SelectionChangeCommitted += gcnew System::EventHandler(this, &menuForm::addUserTypeCB\_SelectionChangeCommitted);

//

// addUserTypeLbl

//

this->addUserTypeLbl->AutoSize = true;

this->addUserTypeLbl->Location = System::Drawing::Point(14, 145);

this->addUserTypeLbl->Name = L"addUserTypeLbl";

this->addUserTypeLbl->Size = System::Drawing::Size(120, 17);

this->addUserTypeLbl->TabIndex = 35;

this->addUserTypeLbl->Text = L"Тип пользователя";

//

// addUserFormBtn

//

this->addUserFormBtn->Cursor = System::Windows::Forms::Cursors::Hand;

this->addUserFormBtn->Location = System::Drawing::Point(141, 331);

this->addUserFormBtn->Name = L"addUserFormBtn";

this->addUserFormBtn->Size = System::Drawing::Size(124, 23);

this->addUserFormBtn->TabIndex = 34;

this->addUserFormBtn->Text = L"Добавить";

this->addUserFormBtn->UseVisualStyleBackColor = true;

this->addUserFormBtn->Click += gcnew System::EventHandler(this, &menuForm::addUserFormBtn\_Click);

//

// addPassLbl

//

this->addPassLbl->AutoSize = true;

this->addPassLbl->Location = System::Drawing::Point(80, 264);

this->addPassLbl->Name = L"addPassLbl";

this->addPassLbl->Size = System::Drawing::Size(55, 17);

this->addPassLbl->TabIndex = 32;

this->addPassLbl->Text = L"Пароль";

//

// addEmailLbl

//

this->addEmailLbl->AutoSize = true;

this->addEmailLbl->Location = System::Drawing::Point(88, 234);

this->addEmailLbl->Name = L"addEmailLbl";

this->addEmailLbl->Size = System::Drawing::Size(45, 17);

this->addEmailLbl->TabIndex = 31;

this->addEmailLbl->Text = L"e-mail";

//

// addPhonenumberLbl

//

this->addPhonenumberLbl->AutoSize = true;

this->addPhonenumberLbl->Location = System::Drawing::Point(21, 204);

this->addPhonenumberLbl->Name = L"addPhonenumberLbl";

this->addPhonenumberLbl->Size = System::Drawing::Size(114, 17);

this->addPhonenumberLbl->TabIndex = 30;

this->addPhonenumberLbl->Text = L"Номер телефона";

//

// addDateLbl

//

this->addDateLbl->AutoSize = true;

this->addDateLbl->Location = System::Drawing::Point(30, 174);

this->addDateLbl->Name = L"addDateLbl";

this->addDateLbl->Size = System::Drawing::Size(105, 17);

this->addDateLbl->TabIndex = 29;

this->addDateLbl->Text = L"Дата рождения";

//

// addUserPasswordTB

//

this->addUserPasswordTB->Location = System::Drawing::Point(141, 261);

this->addUserPasswordTB->Name = L"addUserPasswordTB";

this->addUserPasswordTB->Size = System::Drawing::Size(124, 24);

this->addUserPasswordTB->TabIndex = 25;

//

// addUserMailTB

//

this->addUserMailTB->Location = System::Drawing::Point(141, 231);

this->addUserMailTB->Name = L"addUserMailTB";

this->addUserMailTB->Size = System::Drawing::Size(124, 24);

this->addUserMailTB->TabIndex = 24;

//

// addUserNameTB

//

this->addUserNameTB->Location = System::Drawing::Point(141, 48);

this->addUserNameTB->Name = L"addUserNameTB";

this->addUserNameTB->Size = System::Drawing::Size(124, 24);

this->addUserNameTB->TabIndex = 19;

//

// addUser

//

this->addUser->AutoSize = true;

this->addUser->Font = (gcnew System::Drawing::Font(L"Agency FB", 15.75F, System::Drawing::FontStyle::Regular, System::Drawing::GraphicsUnit::Point,

static\_cast<System::Byte>(0)));

this->addUser->Location = System::Drawing::Point(30, 11);

this->addUser->Name = L"addUser";

this->addUser->Size = System::Drawing::Size(321, 25);

this->addUser->TabIndex = 37;

this->addUser->Text = L"Добавить нового пользователя";

//

// reportPanel

//

this->reportPanel->Controls->Add(this->averageСheckCB);

this->reportPanel->Controls->Add(this->clientsNumberCB);

this->reportPanel->Controls->Add(this->profitCB);

this->reportPanel->Controls->Add(this->proceedsCB);

this->reportPanel->Controls->Add(this->ExpensesCB);

this->reportPanel->Controls->Add(this->updateReportBtn);

this->reportPanel->Controls->Add(this->makeReportRTB);

this->reportPanel->Controls->Add(this->makeReportBtn);

this->reportPanel->Controls->Add(this->reportPeriodCB);

this->reportPanel->Controls->Add(this->reportPeriodLbl);

this->reportPanel->Controls->Add(this->saveReportBtn);

this->reportPanel->Controls->Add(this->financialStatementsLbl);

this->reportPanel->Location = System::Drawing::Point(990, 484);

this->reportPanel->Name = L"reportPanel";

this->reportPanel->Size = System::Drawing::Size(651, 676);

this->reportPanel->TabIndex = 24;

//

// averageСheckCB

//

this->averageСheckCB->AutoSize = true;

this->averageСheckCB->Location = System::Drawing::Point(288, 158);

this->averageСheckCB->Name = L"averageСheckCB";

this->averageСheckCB->Size = System::Drawing::Size(89, 17);

this->averageСheckCB->TabIndex = 28;

this->averageСheckCB->Text = L"Средний чек";

this->averageСheckCB->UseVisualStyleBackColor = true;

//

// clientsNumberCB

//

this->clientsNumberCB->AutoSize = true;

this->clientsNumberCB->Location = System::Drawing::Point(288, 138);

this->clientsNumberCB->Name = L"clientsNumberCB";

this->clientsNumberCB->Size = System::Drawing::Size(135, 17);

this->clientsNumberCB->TabIndex = 27;

this->clientsNumberCB->Text = L"Количество клиентов";

this->clientsNumberCB->UseVisualStyleBackColor = true;

//

// profitCB

//

this->profitCB->AutoSize = true;

this->profitCB->Location = System::Drawing::Point(288, 120);

this->profitCB->Name = L"profitCB";

this->profitCB->Size = System::Drawing::Size(72, 17);

this->profitCB->TabIndex = 26;

this->profitCB->Text = L"Прибыль";

this->profitCB->UseVisualStyleBackColor = true;

//

// proceedsCB

//

this->proceedsCB->AutoSize = true;

this->proceedsCB->Location = System::Drawing::Point(288, 101);

this->proceedsCB->Name = L"proceedsCB";

this->proceedsCB->Size = System::Drawing::Size(69, 17);

this->proceedsCB->TabIndex = 25;

this->proceedsCB->Text = L"Выручка";

this->proceedsCB->UseVisualStyleBackColor = true;

//

// ExpensesCB

//

this->ExpensesCB->AutoSize = true;

this->ExpensesCB->Location = System::Drawing::Point(288, 80);

this->ExpensesCB->Name = L"ExpensesCB";

this->ExpensesCB->Size = System::Drawing::Size(69, 17);

this->ExpensesCB->TabIndex = 24;

this->ExpensesCB->Text = L"Затраты";

this->ExpensesCB->UseVisualStyleBackColor = true;

//

// updateReportBtn

//

this->updateReportBtn->BackgroundImage = (cli::safe\_cast<System::Drawing::Image^>(resources->GetObject(L"updateReportBtn.BackgroundImage")));

this->updateReportBtn->BackgroundImageLayout = System::Windows::Forms::ImageLayout::Zoom;

this->updateReportBtn->Location = System::Drawing::Point(368, 193);

this->updateReportBtn->Name = L"updateReportBtn";

this->updateReportBtn->Size = System::Drawing::Size(40, 40);

this->updateReportBtn->TabIndex = 18;

this->updateReportBtn->UseVisualStyleBackColor = true;

//

// makeReportRTB

//

this->makeReportRTB->Location = System::Drawing::Point(23, 241);

this->makeReportRTB->Name = L"makeReportRTB";

this->makeReportRTB->Size = System::Drawing::Size(385, 265);

this->makeReportRTB->TabIndex = 23;

this->makeReportRTB->Text = L"";

//

// makeReportBtn

//

this->makeReportBtn->Location = System::Drawing::Point(70, 123);

this->makeReportBtn->Name = L"makeReportBtn";

this->makeReportBtn->Size = System::Drawing::Size(121, 34);

this->makeReportBtn->TabIndex = 22;

this->makeReportBtn->Text = L"Составить отчёт";

this->makeReportBtn->UseVisualStyleBackColor = true;

this->makeReportBtn->Click += gcnew System::EventHandler(this, &menuForm::makeReportBtn\_Click);

//

// reportPeriodCB

//

this->reportPeriodCB->FormattingEnabled = true;

this->reportPeriodCB->Items->AddRange(gcnew cli::array< System::Object^ >(3) { L"День", L"Месяц", L"Год" });

this->reportPeriodCB->Location = System::Drawing::Point(70, 83);

this->reportPeriodCB->Name = L"reportPeriodCB";

this->reportPeriodCB->Size = System::Drawing::Size(121, 21);

this->reportPeriodCB->TabIndex = 19;

//

// reportPeriodLbl

//

this->reportPeriodLbl->AutoSize = true;

this->reportPeriodLbl->Location = System::Drawing::Point(20, 88);

this->reportPeriodLbl->Name = L"reportPeriodLbl";

this->reportPeriodLbl->Size = System::Drawing::Size(45, 13);

this->reportPeriodLbl->TabIndex = 18;

this->reportPeriodLbl->Text = L"Период";

//

// saveReportBtn

//

this->saveReportBtn->BackgroundImage = (cli::safe\_cast<System::Drawing::Image^>(resources->GetObject(L"saveReportBtn.BackgroundImage")));

this->saveReportBtn->BackgroundImageLayout = System::Windows::Forms::ImageLayout::Zoom;

this->saveReportBtn->Location = System::Drawing::Point(317, 193);

this->saveReportBtn->Name = L"saveReportBtn";

this->saveReportBtn->Size = System::Drawing::Size(40, 40);

this->saveReportBtn->TabIndex = 17;

this->saveReportBtn->UseVisualStyleBackColor = true;

//

// financialStatementsLbl

//

this->financialStatementsLbl->AutoSize = true;

this->financialStatementsLbl->Font = (gcnew System::Drawing::Font(L"Microsoft Sans Serif", 15.75F, System::Drawing::FontStyle::Regular,

System::Drawing::GraphicsUnit::Point, static\_cast<System::Byte>(204)));

this->financialStatementsLbl->Location = System::Drawing::Point(90, 20);

this->financialStatementsLbl->Name = L"financialStatementsLbl";

this->financialStatementsLbl->Size = System::Drawing::Size(252, 25);

this->financialStatementsLbl->TabIndex = 0;

this->financialStatementsLbl->Text = L"Финансовая отчётность";

//

// addServicePanel

//

this->addServicePanel->Controls->Add(this->loadServicesBtn);

this->addServicePanel->Controls->Add(this->updateServicesBtn);

this->addServicePanel->Controls->Add(this->saveServicesBtn);

this->addServicePanel->Controls->Add(this->servicesRTB);

this->addServicePanel->Controls->Add(this->addServicePriceLbl);

this->addServicePanel->Controls->Add(this->addServices);

this->addServicePanel->Controls->Add(this->addServiceNameLbl);

this->addServicePanel->Controls->Add(this->serviceNameTB);

this->addServicePanel->Controls->Add(this->servicePriceTB);

this->addServicePanel->Controls->Add(this->addServiceBtn);

this->addServicePanel->Font = (gcnew System::Drawing::Font(L"Yu Gothic UI Semibold", 9.25F, System::Drawing::FontStyle::Bold));

this->addServicePanel->Location = System::Drawing::Point(990, 390);

this->addServicePanel->Name = L"addServicePanel";

this->addServicePanel->Size = System::Drawing::Size(651, 698);

this->addServicePanel->TabIndex = 27;

//

// loadServicesBtn

//

this->loadServicesBtn->Location = System::Drawing::Point(305, 278);

this->loadServicesBtn->Name = L"loadServicesBtn";

this->loadServicesBtn->Size = System::Drawing::Size(58, 40);

this->loadServicesBtn->TabIndex = 17;

this->loadServicesBtn->Text = L"load";

this->loadServicesBtn->UseVisualStyleBackColor = true;

this->loadServicesBtn->Click += gcnew System::EventHandler(this, &menuForm::loadServicesBtn\_Click);

//

// updateServicesBtn

//

this->updateServicesBtn->BackgroundImage = (cli::safe\_cast<System::Drawing::Image^>(resources->GetObject(L"updateServicesBtn.BackgroundImage")));

this->updateServicesBtn->BackgroundImageLayout = System::Windows::Forms::ImageLayout::Zoom;

this->updateServicesBtn->Location = System::Drawing::Point(213, 278);

this->updateServicesBtn->Name = L"updateServicesBtn";

this->updateServicesBtn->Size = System::Drawing::Size(40, 40);

this->updateServicesBtn->TabIndex = 16;

this->updateServicesBtn->UseVisualStyleBackColor = true;

this->updateServicesBtn->Click += gcnew System::EventHandler(this, &menuForm::updateServicesBtn\_Click);

//

// saveServicesBtn

//

this->saveServicesBtn->BackgroundImage = (cli::safe\_cast<System::Drawing::Image^>(resources->GetObject(L"saveServicesBtn.BackgroundImage")));

this->saveServicesBtn->BackgroundImageLayout = System::Windows::Forms::ImageLayout::Zoom;

this->saveServicesBtn->Location = System::Drawing::Point(259, 278);

this->saveServicesBtn->Name = L"saveServicesBtn";

this->saveServicesBtn->Size = System::Drawing::Size(40, 40);

this->saveServicesBtn->TabIndex = 16;

this->saveServicesBtn->UseVisualStyleBackColor = true;

this->saveServicesBtn->Click += gcnew System::EventHandler(this, &menuForm::saveServicesBtn\_Click);

//

// servicesRTB

//

this->servicesRTB->Location = System::Drawing::Point(13, 189);

this->servicesRTB->Name = L"servicesRTB";

this->servicesRTB->Size = System::Drawing::Size(350, 83);

this->servicesRTB->TabIndex = 12;

this->servicesRTB->Text = L"";

//

// addServicePriceLbl

//

this->addServicePriceLbl->AutoSize = true;

this->addServicePriceLbl->Font = (gcnew System::Drawing::Font(L"Yu Gothic UI Semibold", 9.25F, System::Drawing::FontStyle::Bold));

this->addServicePriceLbl->Location = System::Drawing::Point(31, 112);

this->addServicePriceLbl->Name = L"addServicePriceLbl";

this->addServicePriceLbl->Size = System::Drawing::Size(40, 17);

this->addServicePriceLbl->TabIndex = 11;

this->addServicePriceLbl->Text = L"Цена";

this->addServicePriceLbl->TextAlign = System::Drawing::ContentAlignment::TopCenter;

//

// addServices

//

this->addServices->AutoSize = true;

this->addServices->Font = (gcnew System::Drawing::Font(L"Yu Gothic UI Semibold", 15.25F, System::Drawing::FontStyle::Bold));

this->addServices->Location = System::Drawing::Point(8, 28);

this->addServices->Name = L"addServices";

this->addServices->Size = System::Drawing::Size(178, 30);

this->addServices->TabIndex = 9;

this->addServices->Text = L"Добавить услугу";

//

// addServiceNameLbl

//

this->addServiceNameLbl->AutoSize = true;

this->addServiceNameLbl->Font = (gcnew System::Drawing::Font(L"Yu Gothic UI Semibold", 9.25F, System::Drawing::FontStyle::Bold));

this->addServiceNameLbl->Location = System::Drawing::Point(8, 79);

this->addServiceNameLbl->Name = L"addServiceNameLbl";

this->addServiceNameLbl->Size = System::Drawing::Size(68, 17);

this->addServiceNameLbl->TabIndex = 10;

this->addServiceNameLbl->Text = L"Название";

//

// serviceNameTB

//

this->serviceNameTB->Location = System::Drawing::Point(87, 76);

this->serviceNameTB->Name = L"serviceNameTB";

this->serviceNameTB->Size = System::Drawing::Size(204, 24);

this->serviceNameTB->TabIndex = 6;

//

// servicePriceTB

//

this->servicePriceTB->Location = System::Drawing::Point(87, 110);

this->servicePriceTB->Name = L"servicePriceTB";

this->servicePriceTB->Size = System::Drawing::Size(204, 24);

this->servicePriceTB->TabIndex = 7;

//

// addServiceBtn

//

this->addServiceBtn->Location = System::Drawing::Point(191, 142);

this->addServiceBtn->Name = L"addServiceBtn";

this->addServiceBtn->Size = System::Drawing::Size(100, 31);

this->addServiceBtn->TabIndex = 8;

this->addServiceBtn->Text = L"Добавить";

this->addServiceBtn->UseVisualStyleBackColor = true;

this->addServiceBtn->Click += gcnew System::EventHandler(this, &menuForm::addServiceBtn\_Click);

//

// clientPanel

//

this->clientPanel->Controls->Add(this->ClientNameLbl);

this->clientPanel->Controls->Add(this->ordersList);

this->clientPanel->Controls->Add(this->exitFromClientPanel);

this->clientPanel->Controls->Add(this->timeStatusLbl);

this->clientPanel->Controls->Add(this->availableTimeLB);

this->clientPanel->Controls->Add(this->updateServiceOrderBtn);

this->clientPanel->Controls->Add(acceptOrderBtn);

this->clientPanel->Controls->Add(this->Calendar);

this->clientPanel->Controls->Add(this->costTB);

this->clientPanel->Controls->Add(this->employeeIDCB);

this->clientPanel->Controls->Add(this->serviceIDCB);

this->clientPanel->Controls->Add(this->selectDateLbl);

this->clientPanel->Controls->Add(this->orderCostLbl);

this->clientPanel->Controls->Add(this->orderInitGrid);

this->clientPanel->Controls->Add(this->employeeIDFromOrderLbl);

this->clientPanel->Controls->Add(this->serviceIDfromOrderLbl);

this->clientPanel->Font = (gcnew System::Drawing::Font(L"Yu Gothic UI Semibold", 9.25F, System::Drawing::FontStyle::Bold, System::Drawing::GraphicsUnit::Point,

static\_cast<System::Byte>(204)));

this->clientPanel->Location = System::Drawing::Point(990, 165);

this->clientPanel->Name = L"clientPanel";

this->clientPanel->Size = System::Drawing::Size(651, 751);

this->clientPanel->TabIndex = 17;

this->clientPanel->Visible = false;

this->clientPanel->VisibleChanged += gcnew System::EventHandler(this, &menuForm::clientPanel\_VisibleChanged);

//

// ClientNameLbl

//

this->ClientNameLbl->AutoSize = true;

this->ClientNameLbl->Font = (gcnew System::Drawing::Font(L"Yu Gothic UI Semibold", 12.25F, System::Drawing::FontStyle::Bold, System::Drawing::GraphicsUnit::Point,

static\_cast<System::Byte>(204)));

this->ClientNameLbl->Location = System::Drawing::Point(45, 44);

this->ClientNameLbl->Name = L"ClientNameLbl";

this->ClientNameLbl->Size = System::Drawing::Size(100, 23);

this->ClientNameLbl->TabIndex = 35;

this->ClientNameLbl->Text = L"ClientName";

//

// ordersList

//

this->ordersList->AutoSize = true;

this->ordersList->Location = System::Drawing::Point(46, 80);

this->ordersList->Name = L"ordersList";

this->ordersList->Size = System::Drawing::Size(151, 17);

this->ordersList->TabIndex = 34;

this->ordersList->Text = L"Список ваших записей";

//

// exitFromClientPanel

//

this->exitFromClientPanel->Location = System::Drawing::Point(580, 309);

this->exitFromClientPanel->Name = L"exitFromClientPanel";

this->exitFromClientPanel->Size = System::Drawing::Size(160, 36);

this->exitFromClientPanel->TabIndex = 33;

this->exitFromClientPanel->Text = L"Выход";

this->exitFromClientPanel->UseVisualStyleBackColor = true;

this->exitFromClientPanel->Click += gcnew System::EventHandler(this, &menuForm::exitFromClientPanel\_Click);

//

// timeStatusLbl

//

this->timeStatusLbl->AutoSize = true;

this->timeStatusLbl->Font = (gcnew System::Drawing::Font(L"Microsoft Sans Serif", 14.25F, System::Drawing::FontStyle::Italic, System::Drawing::GraphicsUnit::Point,

static\_cast<System::Byte>(204)));

this->timeStatusLbl->ForeColor = System::Drawing::Color::DarkRed;

this->timeStatusLbl->Location = System::Drawing::Point(580, 353);

this->timeStatusLbl->Name = L"timeStatusLbl";

this->timeStatusLbl->Size = System::Drawing::Size(0, 24);

this->timeStatusLbl->TabIndex = 31;

//

// availableTimeLB

//

this->availableTimeLB->Font = (gcnew System::Drawing::Font(L"Yu Gothic UI Semibold", 11, System::Drawing::FontStyle::Bold, System::Drawing::GraphicsUnit::Point,

static\_cast<System::Byte>(204)));

this->availableTimeLB->FormatString = L"t";

this->availableTimeLB->FormattingEnabled = true;

this->availableTimeLB->ItemHeight = 20;

this->availableTimeLB->Items->AddRange(gcnew cli::array< System::Object^ >(8) {

L"12:30", L"12:31", L"12:32", L"12:33", L"12:34",

L"12:35", L"12:36", L"12:37"

});

this->availableTimeLB->Location = System::Drawing::Point(217, 394);

this->availableTimeLB->Name = L"availableTimeLB";

this->availableTimeLB->RightToLeft = System::Windows::Forms::RightToLeft::No;

this->availableTimeLB->Size = System::Drawing::Size(164, 24);

this->availableTimeLB->TabIndex = 30;

//

// updateServiceOrderBtn

//

this->updateServiceOrderBtn->BackgroundImage = (cli::safe\_cast<System::Drawing::Image^>(resources->GetObject(L"updateServiceOrderBtn.BackgroundImage")));

this->updateServiceOrderBtn->BackgroundImageLayout = System::Windows::Forms::ImageLayout::Zoom;

this->updateServiceOrderBtn->Location = System::Drawing::Point(700, 52);

this->updateServiceOrderBtn->Name = L"updateServiceOrderBtn";

this->updateServiceOrderBtn->Size = System::Drawing::Size(40, 40);

this->updateServiceOrderBtn->TabIndex = 27;

this->updateServiceOrderBtn->UseVisualStyleBackColor = true;

this->updateServiceOrderBtn->Click += gcnew System::EventHandler(this, &menuForm::updateServiceOrderBtn\_Click);

//

// Calendar

//

this->Calendar->Location = System::Drawing::Point(404, 259);

this->Calendar->MaxSelectionCount = 1;

this->Calendar->Name = L"Calendar";

this->Calendar->TabIndex = 2;

this->Calendar->DateSelected += gcnew System::Windows::Forms::DateRangeEventHandler(this, &menuForm::Calendar\_DateSelected);

//

// costTB

//

this->costTB->Location = System::Drawing::Point(216, 353);

this->costTB->Name = L"costTB";

this->costTB->ReadOnly = true;

this->costTB->Size = System::Drawing::Size(164, 24);

this->costTB->TabIndex = 11;

//

// employeeIDCB

//

this->employeeIDCB->FormattingEnabled = true;

this->employeeIDCB->Location = System::Drawing::Point(217, 307);

this->employeeIDCB->Name = L"employeeIDCB";

this->employeeIDCB->Size = System::Drawing::Size(163, 23);

this->employeeIDCB->TabIndex = 8;

//

// serviceIDCB

//

this->serviceIDCB->FormattingEnabled = true;

this->serviceIDCB->Location = System::Drawing::Point(217, 264);

this->serviceIDCB->Name = L"serviceIDCB";

this->serviceIDCB->Size = System::Drawing::Size(163, 23);

this->serviceIDCB->TabIndex = 7;

this->serviceIDCB->SelectionChangeCommitted += gcnew System::EventHandler(this, &menuForm::serviceIDCB\_SelectionChangeCommitted);

this->serviceIDCB->SelectedValueChanged += gcnew System::EventHandler(this, &menuForm::serviceIDCB\_SelectedValueChanged);

//

// selectDateLbl

//

this->selectDateLbl->AutoSize = true;

this->selectDateLbl->Location = System::Drawing::Point(44, 401);

this->selectDateLbl->Name = L"selectDateLbl";

this->selectDateLbl->Size = System::Drawing::Size(157, 17);

this->selectDateLbl->TabIndex = 32;

this->selectDateLbl->Text = L"Выберите дату и время";

//

// orderCostLbl

//

this->orderCostLbl->AutoSize = true;

this->orderCostLbl->Location = System::Drawing::Point(44, 360);

this->orderCostLbl->Name = L"orderCostLbl";

this->orderCostLbl->Size = System::Drawing::Size(75, 17);

this->orderCostLbl->TabIndex = 5;

this->orderCostLbl->Text = L"Стоимость";

//

// orderInitGrid

//

this->orderInitGrid->AllowUserToAddRows = false;

this->orderInitGrid->AllowUserToDeleteRows = false;

this->orderInitGrid->AutoSizeColumnsMode = System::Windows::Forms::DataGridViewAutoSizeColumnsMode::ColumnHeader;

this->orderInitGrid->BackgroundColor = System::Drawing::SystemColors::ButtonHighlight;

this->orderInitGrid->Columns->AddRange(gcnew cli::array< System::Windows::Forms::DataGridViewColumn^ >(7) {

this->serviceID,

this->employeeOerderID, this->clientIDOrder, this->cost, this->sssorderStatus, this->ordersDate, this->orderTime

});

this->orderInitGrid->EditMode = System::Windows::Forms::DataGridViewEditMode::EditProgrammatically;

this->orderInitGrid->GridColor = System::Drawing::SystemColors::ActiveCaptionText;

this->orderInitGrid->Location = System::Drawing::Point(46, 99);

this->orderInitGrid->Margin = System::Windows::Forms::Padding(2);

this->orderInitGrid->Name = L"orderInitGrid";

this->orderInitGrid->ReadOnly = true;

this->orderInitGrid->RowHeadersWidth = 30;

this->orderInitGrid->Size = System::Drawing::Size(694, 137);

this->orderInitGrid->TabIndex = 0;

//

// serviceID

//

this->serviceID->HeaderText = L"Услуга";

this->serviceID->Name = L"serviceID";

this->serviceID->ReadOnly = true;

this->serviceID->Width = 73;

//

// employeeOerderID

//

this->employeeOerderID->HeaderText = L"Исполнитель";

this->employeeOerderID->Name = L"employeeOerderID";

this->employeeOerderID->ReadOnly = true;

this->employeeOerderID->Width = 115;

//

// clientIDOrder

//

this->clientIDOrder->HeaderText = L"Клиент";

this->clientIDOrder->Name = L"clientIDOrder";

this->clientIDOrder->ReadOnly = true;

this->clientIDOrder->Width = 77;

//

// cost

//

this->cost->HeaderText = L"Стоимость";

this->cost->Name = L"cost";

this->cost->ReadOnly = true;

//

// sssorderStatus

//

this->sssorderStatus->HeaderText = L"Статус заказа";

this->sssorderStatus->Name = L"sssorderStatus";

this->sssorderStatus->ReadOnly = true;

this->sssorderStatus->Width = 117;

//

// ordersDate

//

this->ordersDate->HeaderText = L"Дата";

this->ordersDate->Name = L"ordersDate";

this->ordersDate->ReadOnly = true;

this->ordersDate->Width = 62;

//

// orderTime

//

this->orderTime->HeaderText = L"Время";

this->orderTime->Name = L"orderTime";

this->orderTime->ReadOnly = true;

this->orderTime->Width = 73;

//

// employeeIDFromOrderLbl

//

this->employeeIDFromOrderLbl->AutoSize = true;

this->employeeIDFromOrderLbl->Location = System::Drawing::Point(44, 313);

this->employeeIDFromOrderLbl->Name = L"employeeIDFromOrderLbl";

this->employeeIDFromOrderLbl->Size = System::Drawing::Size(76, 17);

this->employeeIDFromOrderLbl->TabIndex = 3;

this->employeeIDFromOrderLbl->Text = L"Сотрудник";

//

// serviceIDfromOrderLbl

//

this->serviceIDfromOrderLbl->AutoSize = true;

this->serviceIDfromOrderLbl->Location = System::Drawing::Point(43, 265);

this->serviceIDfromOrderLbl->Name = L"serviceIDfromOrderLbl";

this->serviceIDfromOrderLbl->Size = System::Drawing::Size(113, 17);

this->serviceIDfromOrderLbl->TabIndex = 2;

this->serviceIDfromOrderLbl->Text = L"Выберите услугу";

//

// employeePanel

//

this->employeePanel->Controls->Add(this->updateOrdersGridBtn);

this->employeePanel->Controls->Add(this->label6);

this->employeePanel->Controls->Add(this->label5);

this->employeePanel->Controls->Add(this->selectDataLbl);

this->employeePanel->Controls->Add(this->selectedDateEmployeeLbl);

this->employeePanel->Controls->Add(this->selectedDateEmployeeTB);

this->employeePanel->Controls->Add(this->employeeCalendar);

this->employeePanel->Controls->Add(this->label13);

this->employeePanel->Controls->Add(this->achievementsEmployeeTB);

this->employeePanel->Controls->Add(this->specialityEmployeeTB);

this->employeePanel->Controls->Add(this->nameEmployeeTB);

this->employeePanel->Controls->Add(this->orderEmployeeLbl);

this->employeePanel->Controls->Add(this->orderEmployeeIdCB);

this->employeePanel->Controls->Add(this->orderEmployeeCheckCB);

this->employeePanel->Controls->Add(this->employeeDataGrid);

this->employeePanel->Controls->Add(this->achievementsEmployeeLbl);

this->employeePanel->Controls->Add(this->specialityEmployeeLbl);

this->employeePanel->Controls->Add(this->nameEmployeeLbl);

this->employeePanel->Controls->Add(this->exitFromEmployeePanelBtn);

this->employeePanel->Font = (gcnew System::Drawing::Font(L"Yu Gothic UI Semibold", 12.25F, System::Drawing::FontStyle::Bold));

this->employeePanel->Location = System::Drawing::Point(23, 28);

this->employeePanel->Name = L"employeePanel";

this->employeePanel->Size = System::Drawing::Size(913, 712);

this->employeePanel->TabIndex = 18;

this->employeePanel->Visible = false;

this->employeePanel->VisibleChanged += gcnew System::EventHandler(this, &menuForm::employeePanel\_VisibleChanged);

//

// updateOrdersGridBtn

//

this->updateOrdersGridBtn->BackgroundImage = (cli::safe\_cast<System::Drawing::Image^>(resources->GetObject(L"updateOrdersGridBtn.BackgroundImage")));

this->updateOrdersGridBtn->BackgroundImageLayout = System::Windows::Forms::ImageLayout::Zoom;

this->updateOrdersGridBtn->Location = System::Drawing::Point(761, 282);

this->updateOrdersGridBtn->Name = L"updateOrdersGridBtn";

this->updateOrdersGridBtn->Size = System::Drawing::Size(40, 40);

this->updateOrdersGridBtn->TabIndex = 28;

this->updateOrdersGridBtn->UseVisualStyleBackColor = true;

this->updateOrdersGridBtn->Click += gcnew System::EventHandler(this, &menuForm::updateOrdersGridBtn\_Click);

//

// label6

//

this->label6->Anchor = System::Windows::Forms::AnchorStyles::Left;

this->label6->AutoSize = true;

this->label6->Font = (gcnew System::Drawing::Font(L"Yu Gothic UI Semibold", 8, System::Drawing::FontStyle::Bold));

this->label6->Location = System::Drawing::Point(38, 524);

this->label6->Name = L"label6";

this->label6->Size = System::Drawing::Size(235, 13);

this->label6->TabIndex = 20;

this->label6->Text = L"---------------------------------------------------------";

//

// label5

//

this->label5->Anchor = System::Windows::Forms::AnchorStyles::Left;

this->label5->AutoSize = true;

this->label5->Font = (gcnew System::Drawing::Font(L"Yu Gothic UI Semibold", 8, System::Drawing::FontStyle::Bold));

this->label5->Location = System::Drawing::Point(38, 379);

this->label5->Name = L"label5";

this->label5->Size = System::Drawing::Size(235, 13);

this->label5->TabIndex = 19;

this->label5->Text = L"---------------------------------------------------------";

//

// selectDataLbl

//

this->selectDataLbl->Anchor = System::Windows::Forms::AnchorStyles::Left;

this->selectDataLbl->AutoSize = true;

this->selectDataLbl->Font = (gcnew System::Drawing::Font(L"Yu Gothic UI Semibold", 9.25F, System::Drawing::FontStyle::Bold));

this->selectDataLbl->Location = System::Drawing::Point(343, 289);

this->selectDataLbl->Name = L"selectDataLbl";

this->selectDataLbl->Size = System::Drawing::Size(102, 17);

this->selectDataLbl->TabIndex = 17;

this->selectDataLbl->Text = L"Выберите дату";

//

// selectedDateEmployeeLbl

//

this->selectedDateEmployeeLbl->Anchor = System::Windows::Forms::AnchorStyles::Left;

this->selectedDateEmployeeLbl->AutoSize = true;

this->selectedDateEmployeeLbl->Font = (gcnew System::Drawing::Font(L"Yu Gothic UI Semibold", 9.25F, System::Drawing::FontStyle::Bold));

this->selectedDateEmployeeLbl->Location = System::Drawing::Point(343, 497);

this->selectedDateEmployeeLbl->Name = L"selectedDateEmployeeLbl";

this->selectedDateEmployeeLbl->Size = System::Drawing::Size(114, 17);

this->selectedDateEmployeeLbl->TabIndex = 16;

this->selectedDateEmployeeLbl->Text = L"Выбранная дата:";

//

// selectedDateEmployeeTB

//

this->selectedDateEmployeeTB->Anchor = System::Windows::Forms::AnchorStyles::Left;

this->selectedDateEmployeeTB->Font = (gcnew System::Drawing::Font(L"Yu Gothic UI Semibold", 9.25F, System::Drawing::FontStyle::Bold));

this->selectedDateEmployeeTB->Location = System::Drawing::Point(346, 524);

this->selectedDateEmployeeTB->Name = L"selectedDateEmployeeTB";

this->selectedDateEmployeeTB->Size = System::Drawing::Size(164, 24);

this->selectedDateEmployeeTB->TabIndex = 15;

//

// employeeCalendar

//

this->employeeCalendar->Anchor = System::Windows::Forms::AnchorStyles::Left;

this->employeeCalendar->Location = System::Drawing::Point(346, 320);

this->employeeCalendar->MaxSelectionCount = 1;

this->employeeCalendar->Name = L"employeeCalendar";

this->employeeCalendar->TabIndex = 2;

this->employeeCalendar->DateChanged += gcnew System::Windows::Forms::DateRangeEventHandler(this, &menuForm::employeeCalendar\_DateChanged);

this->employeeCalendar->DateSelected += gcnew System::Windows::Forms::DateRangeEventHandler(this, &menuForm::employeeCalendar\_DateSelected);

//

// label13

//

this->label13->Anchor = System::Windows::Forms::AnchorStyles::Left;

this->label13->AutoSize = true;

this->label13->Font = (gcnew System::Drawing::Font(L"Yu Gothic UI Semibold", 8, System::Drawing::FontStyle::Bold));

this->label13->Location = System::Drawing::Point(40, 320);

this->label13->Name = L"label13";

this->label13->Size = System::Drawing::Size(235, 13);

this->label13->TabIndex = 12;

this->label13->Text = L"---------------------------------------------------------";

//

// achievementsEmployeeTB

//

this->achievementsEmployeeTB->Anchor = System::Windows::Forms::AnchorStyles::Left;

this->achievementsEmployeeTB->Font = (gcnew System::Drawing::Font(L"Yu Gothic UI Semibold", 9.25F, System::Drawing::FontStyle::Bold));

this->achievementsEmployeeTB->Location = System::Drawing::Point(42, 421);

this->achievementsEmployeeTB->Multiline = true;

this->achievementsEmployeeTB->Name = L"achievementsEmployeeTB";

this->achievementsEmployeeTB->ReadOnly = true;

this->achievementsEmployeeTB->Size = System::Drawing::Size(231, 93);

this->achievementsEmployeeTB->TabIndex = 10;

//

// specialityEmployeeTB

//

this->specialityEmployeeTB->Anchor = System::Windows::Forms::AnchorStyles::Left;

this->specialityEmployeeTB->Font = (gcnew System::Drawing::Font(L"Yu Gothic UI Semibold", 9.25F, System::Drawing::FontStyle::Bold));

this->specialityEmployeeTB->Location = System::Drawing::Point(173, 347);

this->specialityEmployeeTB->Name = L"specialityEmployeeTB";

this->specialityEmployeeTB->ReadOnly = true;

this->specialityEmployeeTB->Size = System::Drawing::Size(100, 24);

this->specialityEmployeeTB->TabIndex = 9;

//

// nameEmployeeTB

//

this->nameEmployeeTB->Anchor = System::Windows::Forms::AnchorStyles::Left;

this->nameEmployeeTB->Font = (gcnew System::Drawing::Font(L"Yu Gothic UI Semibold", 9.25F, System::Drawing::FontStyle::Bold));

this->nameEmployeeTB->Location = System::Drawing::Point(173, 286);

this->nameEmployeeTB->Name = L"nameEmployeeTB";

this->nameEmployeeTB->ReadOnly = true;

this->nameEmployeeTB->Size = System::Drawing::Size(100, 24);

this->nameEmployeeTB->TabIndex = 8;

//

// orderEmployeeLbl

//

this->orderEmployeeLbl->Anchor = System::Windows::Forms::AnchorStyles::Left;

this->orderEmployeeLbl->AutoSize = true;

this->orderEmployeeLbl->Font = (gcnew System::Drawing::Font(L"Yu Gothic UI Semibold", 9.25F, System::Drawing::FontStyle::Bold));

this->orderEmployeeLbl->Location = System::Drawing::Point(42, 551);

this->orderEmployeeLbl->Name = L"orderEmployeeLbl";

this->orderEmployeeLbl->Size = System::Drawing::Size(42, 17);

this->orderEmployeeLbl->TabIndex = 7;

this->orderEmployeeLbl->Text = L"Заказ";

//

// orderEmployeeIdCB

//

this->orderEmployeeIdCB->Anchor = System::Windows::Forms::AnchorStyles::Left;

this->orderEmployeeIdCB->Font = (gcnew System::Drawing::Font(L"Yu Gothic UI Semibold", 9.25F, System::Drawing::FontStyle::Bold));

this->orderEmployeeIdCB->FormattingEnabled = true;

this->orderEmployeeIdCB->Location = System::Drawing::Point(173, 545);

this->orderEmployeeIdCB->Name = L"orderEmployeeIdCB";

this->orderEmployeeIdCB->Size = System::Drawing::Size(100, 23);

this->orderEmployeeIdCB->TabIndex = 6;

this->orderEmployeeIdCB->SelectionChangeCommitted += gcnew System::EventHandler(this, &menuForm::orderEmployeeIdCB\_SelectionChangeCommitted);

//

// orderEmployeeCheckCB

//

this->orderEmployeeCheckCB->Anchor = System::Windows::Forms::AnchorStyles::Left;

this->orderEmployeeCheckCB->AutoSize = true;

this->orderEmployeeCheckCB->Font = (gcnew System::Drawing::Font(L"Yu Gothic UI Semibold", 9.25F, System::Drawing::FontStyle::Bold));

this->orderEmployeeCheckCB->Location = System::Drawing::Point(173, 574);

this->orderEmployeeCheckCB->Name = L"orderEmployeeCheckCB";

this->orderEmployeeCheckCB->Size = System::Drawing::Size(91, 21);

this->orderEmployeeCheckCB->TabIndex = 5;

this->orderEmployeeCheckCB->Text = L"Выполнен";

this->orderEmployeeCheckCB->UseVisualStyleBackColor = true;

this->orderEmployeeCheckCB->CheckedChanged += gcnew System::EventHandler(this, &menuForm::orderEmployeeCheckCB\_CheckedChanged);

//

// employeeDataGrid

//

this->employeeDataGrid->AllowUserToAddRows = false;

this->employeeDataGrid->AllowUserToDeleteRows = false;

this->employeeDataGrid->Anchor = System::Windows::Forms::AnchorStyles::Left;

this->employeeDataGrid->AutoSizeColumnsMode = System::Windows::Forms::DataGridViewAutoSizeColumnsMode::Fill;

this->employeeDataGrid->BackgroundColor = System::Drawing::SystemColors::ButtonHighlight;

this->employeeDataGrid->ColumnHeadersHeightSizeMode = System::Windows::Forms::DataGridViewColumnHeadersHeightSizeMode::AutoSize;

this->employeeDataGrid->Columns->AddRange(gcnew cli::array< System::Windows::Forms::DataGridViewColumn^ >(7) {

this->employeeIdDG,

this->dateEmployeeDG, this->serviceEmployeeDG, this->clientNameEmployeeDG, this->costOfServiceEmployeeDG, this->ordersTime, this->orderCurStatus

});

this->employeeDataGrid->Location = System::Drawing::Point(41, 77);

this->employeeDataGrid->Name = L"employeeDataGrid";

this->employeeDataGrid->Size = System::Drawing::Size(760, 195);

this->employeeDataGrid->TabIndex = 4;

//

// employeeIdDG

//

this->employeeIdDG->HeaderText = L"id";

this->employeeIdDG->Name = L"employeeIdDG";

//

// dateEmployeeDG

//

this->dateEmployeeDG->HeaderText = L"Дата";

this->dateEmployeeDG->Name = L"dateEmployeeDG";

//

// serviceEmployeeDG

//

this->serviceEmployeeDG->HeaderText = L"Услуга";

this->serviceEmployeeDG->Name = L"serviceEmployeeDG";

//

// clientNameEmployeeDG

//

this->clientNameEmployeeDG->HeaderText = L"Клиент";

this->clientNameEmployeeDG->Name = L"clientNameEmployeeDG";

//

// costOfServiceEmployeeDG

//

this->costOfServiceEmployeeDG->HeaderText = L"Стоимость";

this->costOfServiceEmployeeDG->Name = L"costOfServiceEmployeeDG";

//

// ordersTime

//

this->ordersTime->HeaderText = L"Время";

this->ordersTime->Name = L"ordersTime";

//

// orderCurStatus

//

this->orderCurStatus->HeaderText = L"Статус";

this->orderCurStatus->Name = L"orderCurStatus";

//

// achievementsEmployeeLbl

//

this->achievementsEmployeeLbl->Anchor = System::Windows::Forms::AnchorStyles::Left;

this->achievementsEmployeeLbl->AutoSize = true;

this->achievementsEmployeeLbl->Font = (gcnew System::Drawing::Font(L"Yu Gothic UI Semibold", 9.25F, System::Drawing::FontStyle::Bold));

this->achievementsEmployeeLbl->Location = System::Drawing::Point(38, 397);

this->achievementsEmployeeLbl->Name = L"achievementsEmployeeLbl";

this->achievementsEmployeeLbl->Size = System::Drawing::Size(138, 17);

this->achievementsEmployeeLbl->TabIndex = 3;

this->achievementsEmployeeLbl->Text = L"Личные достижения";

//

// specialityEmployeeLbl

//

this->specialityEmployeeLbl->Anchor = System::Windows::Forms::AnchorStyles::Left;

this->specialityEmployeeLbl->AutoSize = true;

this->specialityEmployeeLbl->Font = (gcnew System::Drawing::Font(L"Yu Gothic UI Semibold", 9.25F, System::Drawing::FontStyle::Bold));

this->specialityEmployeeLbl->Location = System::Drawing::Point(42, 350);

this->specialityEmployeeLbl->Name = L"specialityEmployeeLbl";

this->specialityEmployeeLbl->Size = System::Drawing::Size(77, 17);

this->specialityEmployeeLbl->TabIndex = 2;

this->specialityEmployeeLbl->Text = L"Должность";

//

// nameEmployeeLbl

//

this->nameEmployeeLbl->Anchor = System::Windows::Forms::AnchorStyles::Left;

this->nameEmployeeLbl->AutoSize = true;

this->nameEmployeeLbl->Font = (gcnew System::Drawing::Font(L"Yu Gothic UI Semibold", 9.25F, System::Drawing::FontStyle::Bold));

this->nameEmployeeLbl->Location = System::Drawing::Point(42, 293);

this->nameEmployeeLbl->Name = L"nameEmployeeLbl";

this->nameEmployeeLbl->Size = System::Drawing::Size(35, 17);

this->nameEmployeeLbl->TabIndex = 1;

this->nameEmployeeLbl->Text = L"Имя";

//

// exitFromEmployeePanelBtn

//

this->exitFromEmployeePanelBtn->Anchor = System::Windows::Forms::AnchorStyles::Left;

this->exitFromEmployeePanelBtn->Font = (gcnew System::Drawing::Font(L"Yu Gothic UI Semibold", 9.25F, System::Drawing::FontStyle::Bold));

this->exitFromEmployeePanelBtn->Location = System::Drawing::Point(707, 589);

this->exitFromEmployeePanelBtn->Name = L"exitFromEmployeePanelBtn";

this->exitFromEmployeePanelBtn->Size = System::Drawing::Size(94, 35);

this->exitFromEmployeePanelBtn->TabIndex = 0;

this->exitFromEmployeePanelBtn->Text = L"Выход";

this->exitFromEmployeePanelBtn->UseVisualStyleBackColor = true;

this->exitFromEmployeePanelBtn->Click += gcnew System::EventHandler(this, &menuForm::exitFromEmployeePanelBtn\_Click);

//

// menuForm

//

this->AutoScaleDimensions = System::Drawing::SizeF(6, 13);

this->AutoScaleMode = System::Windows::Forms::AutoScaleMode::Font;

this->ClientSize = System::Drawing::Size(1707, 912);

this->Controls->Add(this->employeePanel);

this->Controls->Add(this->changeUserPanel);

this->Controls->Add(this->userRegistrationPanel);

this->Controls->Add(this->reportPanel);

this->Controls->Add(this->addServicePanel);

this->Controls->Add(this->changeServicePanel);

this->Controls->Add(this->adminPanel);

this->Controls->Add(this->clientPanel);

this->MaximizeBox = false;

this->Name = L"menuForm";

this->ShowIcon = false;

this->ShowInTaskbar = false;

this->SizeGripStyle = System::Windows::Forms::SizeGripStyle::Hide;

this->Text = L"menuForm";

this->Load += gcnew System::EventHandler(this, &menuForm::menuForm\_Load);

this->VisibleChanged += gcnew System::EventHandler(this, &menuForm::menuForm\_VisibleChanged);

this->adminPanel->ResumeLayout(false);

this->adminPanel->PerformLayout();

(cli::safe\_cast<System::ComponentModel::ISupportInitialize^>(this->ordersDataGrid))->EndInit();

(cli::safe\_cast<System::ComponentModel::ISupportInitialize^>(this->servicesDataGrid))->EndInit();

(cli::safe\_cast<System::ComponentModel::ISupportInitialize^>(this->usersDataGrid))->EndInit();

this->changeServicePanel->ResumeLayout(false);

this->changeServicePanel->PerformLayout();

this->changeUserPanel->ResumeLayout(false);

this->changeUserPanel->PerformLayout();

this->changeUserSpecialityPanel->ResumeLayout(false);

this->changeUserSpecialityPanel->PerformLayout();

this->changeUserSexGP->ResumeLayout(false);

this->changeUserSexGP->PerformLayout();

this->userRegistrationPanel->ResumeLayout(false);

this->userRegistrationPanel->PerformLayout();

this->addUserSexGP->ResumeLayout(false);

this->addUserSexGP->PerformLayout();

this->specialityPanel->ResumeLayout(false);

this->specialityPanel->PerformLayout();

this->reportPanel->ResumeLayout(false);

this->reportPanel->PerformLayout();

this->addServicePanel->ResumeLayout(false);

this->addServicePanel->PerformLayout();

this->clientPanel->ResumeLayout(false);

this->clientPanel->PerformLayout();

(cli::safe\_cast<System::ComponentModel::ISupportInitialize^>(this->orderInitGrid))->EndInit();

this->employeePanel->ResumeLayout(false);

this->employeePanel->PerformLayout();

(cli::safe\_cast<System::ComponentModel::ISupportInitialize^>(this->employeeDataGrid))->EndInit();

this->ResumeLayout(false);

}

#pragma endregion

private: System::Void bshop::menuForm::updateAvailableTimes();

private: System::Void bshop::menuForm::updateOrdersGrid(std::list<ServiceOrder\*> orders);

private: System::Int32 bshop::menuForm::findID(int employeeIndex, bool isEmployee);

private: System::Void menuForm\_Load(System::Object^ sender, System::EventArgs^ e);

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

private: System::Void updateAvailableTimes(System::Object^ sender, System::EventArgs^ e);

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

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

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

private: System::Void Calendar\_DateSelected(System::Object^ sender, System::Windows::Forms::DateRangeEventArgs^ e);

private: System::Void serviceIDCB\_SelectedValueChanged(System::Object^ sender, System::EventArgs^ e);

private: System::Void menuForm\_VisibleChanged(System::Object^ sender, System::EventArgs^ e);

private: System::Void serviceIDCB\_SelectionChangeCommitted(System::Object^ sender, System::EventArgs^ e);

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

private: System::Void employeeCalendar\_DateSelected(System::Object^ sender, System::Windows::Forms::DateRangeEventArgs^ e);

private: System::Void employeePanel\_VisibleChanged(System::Object^ sender, System::EventArgs^ e);

private: System::Void orderEmployeeIdCB\_SelectionChangeCommitted(System::Object^ sender, System::EventArgs^ e);

private: System::Void orderEmployeeCheckCB\_CheckedChanged(System::Object^ sender, System::EventArgs^ e);

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

private: System::Void employeeCalendar\_DateChanged(System::Object^ sender, System::Windows::Forms::DateRangeEventArgs^ e);

private: System::Void bshop::menuForm::updateTimetable(MonthCalendar^ employeeCalendar, TextBox^ selectedDateEmployeeTB);

private: System::Void clientPanel\_VisibleChanged(System::Object^ sender, System::EventArgs^ e);

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

private: System::Void addUserBtn\_Click\_1(System::Object^ sender, System::EventArgs^ e);

private: System::Void addUserTypeCB\_SelectionChangeCommitted(System::Object^ sender, System::EventArgs^ e);

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

private: System::Void changeUserPanel\_VisibleChanged(System::Object^ sender, System::EventArgs^ e);

private: System::Void changeUserIdCB\_SelectionChangeCommitted(System::Object^ sender, System::EventArgs^ e);

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

private: System::Void changeUserTypeCB\_SelectedIndexChanged(System::Object^ sender, System::EventArgs^ e);

private: System::Void addUserTypeCB\_SelectedIndexChanged(System::Object^ sender, System::EventArgs^ e);

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

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

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

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

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

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

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

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

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

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

private: System::Void bshop::menuForm::clearAllGrids();

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

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

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

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

//private: System::Void reportPanel\_Paint(System::Object^ sender, System::Windows::Forms::PaintEventArgs^ e);

private: System::Void bshop::menuForm::closeAllPanels(System::Object^ sender, System::EventArgs^ e);

};

}

#endif MENUFORM\_H

----------------

mainForm.h

----------------

#ifndef MAINFORM\_H

#define MAINFORM\_H

#include "menuForm.h"

#include "adminForm.h"

#include "clientForm.h"

#include "employeeForm.h"

#include "MockDatabase.h"

#include "checkFunctions.h"

namespace bshop {

using namespace System;

using namespace System::ComponentModel;

using namespace System::Collections;

using namespace System::Windows::Forms;

using namespace System::Data;

using namespace System::Drawing;

/// <summary>

/// Сводка для mainForm

/// </summary>

public ref class mainForm : public System::Windows::Forms::Form

{

public:

mainForm(void)

{

InitializeComponent();

}

protected:

/// <summary>

/// Освободить все используемые ресурсы.

/// </summary>

~mainForm()

{

if (components)

{

delete components;

}

}

protected:

private:

/// <summary>

/// Обязательная переменная конструктора.

/// </summary>

//MockDatabase\* db = MockDatabase::getInstance();

System::ComponentModel::Container^ components;

#pragma region Windows Form Designer generated code

/// <summary>

/// Требуемый метод для поддержки конструктора — не изменяйте

/// содержимое этого метода с помощью редактора кода.

/// </summary>

void InitializeComponent(void)

{

System::Windows::Forms::Label^ addNameLbl;

this->lgnPanel = (gcnew System::Windows::Forms::Panel());

this->ragBtn = (gcnew System::Windows::Forms::Button());

this->lgnPasswordLbl = (gcnew System::Windows::Forms::Label());

this->lgnEmailLbl = (gcnew System::Windows::Forms::Label());

this->enterLbl = (gcnew System::Windows::Forms::Label());

this->lgnAcceptBtn = (gcnew System::Windows::Forms::Button());

this->lgnPasswordTB = (gcnew System::Windows::Forms::TextBox());

this->lgnEmailTB = (gcnew System::Windows::Forms::TextBox());

this->userRegistrationPanel = (gcnew System::Windows::Forms::Panel());

this->addUserPhonenumberTB = (gcnew System::Windows::Forms::MaskedTextBox());

this->addUserDateTB = (gcnew System::Windows::Forms::MaskedTextBox());

this->addUserSexGP = (gcnew System::Windows::Forms::GroupBox());

this->addUserFemaleRB = (gcnew System::Windows::Forms::RadioButton());

this->addUserMaleRB = (gcnew System::Windows::Forms::RadioButton());

this->addUserFormBtn = (gcnew System::Windows::Forms::Button());

this->addPassLbl = (gcnew System::Windows::Forms::Label());

this->addEmailLbl = (gcnew System::Windows::Forms::Label());

this->addPhonenumberLbl = (gcnew System::Windows::Forms::Label());

this->addDateLbl = (gcnew System::Windows::Forms::Label());

this->addUserPasswordTB = (gcnew System::Windows::Forms::TextBox());

this->addUserMailTB = (gcnew System::Windows::Forms::TextBox());

this->addUserNameTB = (gcnew System::Windows::Forms::TextBox());

this->addUser = (gcnew System::Windows::Forms::Label());

addNameLbl = (gcnew System::Windows::Forms::Label());

this->lgnPanel->SuspendLayout();

this->userRegistrationPanel->SuspendLayout();

this->addUserSexGP->SuspendLayout();

this->SuspendLayout();

//

// addNameLbl

//

addNameLbl->AutoSize = true;

addNameLbl->Location = System::Drawing::Point(98, 51);

addNameLbl->Name = L"addNameLbl";

addNameLbl->Size = System::Drawing::Size(35, 17);

addNameLbl->TabIndex = 27;

addNameLbl->Text = L"Имя";

//

// lgnPanel

//

this->lgnPanel->BackgroundImageLayout = System::Windows::Forms::ImageLayout::Zoom;

this->lgnPanel->Controls->Add(this->ragBtn);

this->lgnPanel->Controls->Add(this->lgnPasswordLbl);

this->lgnPanel->Controls->Add(this->lgnEmailLbl);

this->lgnPanel->Controls->Add(this->enterLbl);

this->lgnPanel->Controls->Add(this->lgnAcceptBtn);

this->lgnPanel->Controls->Add(this->lgnPasswordTB);

this->lgnPanel->Controls->Add(this->lgnEmailTB);

this->lgnPanel->Location = System::Drawing::Point(5, 4);

this->lgnPanel->Name = L"lgnPanel";

this->lgnPanel->Size = System::Drawing::Size(396, 436);

this->lgnPanel->TabIndex = 14;

this->lgnPanel->Paint += gcnew System::Windows::Forms::PaintEventHandler(this, &mainForm::panel3\_Paint);

//

// ragBtn

//

this->ragBtn->Location = System::Drawing::Point(80, 234);

this->ragBtn->Name = L"ragBtn";

this->ragBtn->Size = System::Drawing::Size(94, 23);

this->ragBtn->TabIndex = 6;

this->ragBtn->Text = L"Регистрация";

this->ragBtn->UseVisualStyleBackColor = true;

this->ragBtn->Click += gcnew System::EventHandler(this, &mainForm::ragBtn\_Click);

//

// lgnPasswordLbl

//

this->lgnPasswordLbl->AutoSize = true;

this->lgnPasswordLbl->Font = (gcnew System::Drawing::Font(L"Arial Narrow", 10.25F));

this->lgnPasswordLbl->Location = System::Drawing::Point(77, 204);

this->lgnPasswordLbl->Name = L"lgnPasswordLbl";

this->lgnPasswordLbl->Size = System::Drawing::Size(58, 17);

this->lgnPasswordLbl->TabIndex = 5;

this->lgnPasswordLbl->Text = L"Password";

this->lgnPasswordLbl->TextAlign = System::Drawing::ContentAlignment::TopCenter;

//

// lgnEmailLbl

//

this->lgnEmailLbl->AutoSize = true;

this->lgnEmailLbl->Font = (gcnew System::Drawing::Font(L"Arial Narrow", 10.25F));

this->lgnEmailLbl->Location = System::Drawing::Point(95, 180);

this->lgnEmailLbl->Name = L"lgnEmailLbl";

this->lgnEmailLbl->Size = System::Drawing::Size(40, 17);

this->lgnEmailLbl->TabIndex = 4;

this->lgnEmailLbl->Text = L"E-mail";

//

// enterLbl

//

this->enterLbl->AutoSize = true;

this->enterLbl->Font = (gcnew System::Drawing::Font(L"Agency FB", 48, System::Drawing::FontStyle::Regular, System::Drawing::GraphicsUnit::Point,

static\_cast<System::Byte>(0)));

this->enterLbl->Location = System::Drawing::Point(119, 99);

this->enterLbl->Name = L"enterLbl";

this->enterLbl->Size = System::Drawing::Size(147, 77);

this->enterLbl->TabIndex = 3;

this->enterLbl->Text = L"LOGIN";

this->enterLbl->Click += gcnew System::EventHandler(this, &mainForm::enterLbl\_Click);

//

// lgnAcceptBtn

//

this->lgnAcceptBtn->Location = System::Drawing::Point(191, 234);

this->lgnAcceptBtn->Name = L"lgnAcceptBtn";

this->lgnAcceptBtn->Size = System::Drawing::Size(94, 23);

this->lgnAcceptBtn->TabIndex = 2;

this->lgnAcceptBtn->Text = L"Войти";

this->lgnAcceptBtn->UseVisualStyleBackColor = true;

this->lgnAcceptBtn->Click += gcnew System::EventHandler(this, &mainForm::lgnAcceptBtn\_Click);

//

// lgnPasswordTB

//

this->lgnPasswordTB->Location = System::Drawing::Point(141, 204);

this->lgnPasswordTB->Name = L"lgnPasswordTB";

this->lgnPasswordTB->Size = System::Drawing::Size(100, 20);

this->lgnPasswordTB->TabIndex = 1;

//

// lgnEmailTB

//

this->lgnEmailTB->Location = System::Drawing::Point(141, 177);

this->lgnEmailTB->Name = L"lgnEmailTB";

this->lgnEmailTB->Size = System::Drawing::Size(100, 20);

this->lgnEmailTB->TabIndex = 0;

//

// userRegistrationPanel

//

this->userRegistrationPanel->Controls->Add(this->addUserPhonenumberTB);

this->userRegistrationPanel->Controls->Add(this->addUserDateTB);

this->userRegistrationPanel->Controls->Add(this->addUserSexGP);

this->userRegistrationPanel->Controls->Add(this->addUserFormBtn);

this->userRegistrationPanel->Controls->Add(this->addPassLbl);

this->userRegistrationPanel->Controls->Add(this->addEmailLbl);

this->userRegistrationPanel->Controls->Add(this->addPhonenumberLbl);

this->userRegistrationPanel->Controls->Add(this->addDateLbl);

this->userRegistrationPanel->Controls->Add(addNameLbl);

this->userRegistrationPanel->Controls->Add(this->addUserPasswordTB);

this->userRegistrationPanel->Controls->Add(this->addUserMailTB);

this->userRegistrationPanel->Controls->Add(this->addUserNameTB);

this->userRegistrationPanel->Controls->Add(this->addUser);

this->userRegistrationPanel->Font = (gcnew System::Drawing::Font(L"Yu Gothic UI Semibold", 9.25F, System::Drawing::FontStyle::Bold));

this->userRegistrationPanel->Location = System::Drawing::Point(12, 12);

this->userRegistrationPanel->Name = L"userRegistrationPanel";

this->userRegistrationPanel->Size = System::Drawing::Size(374, 399);

this->userRegistrationPanel->TabIndex = 27;

//

// addUserPhonenumberTB

//

this->addUserPhonenumberTB->Location = System::Drawing::Point(141, 176);

this->addUserPhonenumberTB->Mask = L"(999) 000-0000";

this->addUserPhonenumberTB->Name = L"addUserPhonenumberTB";

this->addUserPhonenumberTB->Size = System::Drawing::Size(124, 24);

this->addUserPhonenumberTB->TabIndex = 28;

//

// addUserDateTB

//

this->addUserDateTB->Location = System::Drawing::Point(141, 146);

this->addUserDateTB->Mask = L"00/00/0000";

this->addUserDateTB->Name = L"addUserDateTB";

this->addUserDateTB->Size = System::Drawing::Size(124, 24);

this->addUserDateTB->TabIndex = 28;

this->addUserDateTB->ValidatingType = System::DateTime::typeid;

//

// addUserSexGP

//

this->addUserSexGP->Controls->Add(this->addUserFemaleRB);

this->addUserSexGP->Controls->Add(this->addUserMaleRB);

this->addUserSexGP->Location = System::Drawing::Point(88, 78);

this->addUserSexGP->Name = L"addUserSexGP";

this->addUserSexGP->Size = System::Drawing::Size(175, 58);

this->addUserSexGP->TabIndex = 41;

this->addUserSexGP->TabStop = false;

this->addUserSexGP->Text = L"Пол";

//

// addUserFemaleRB

//

this->addUserFemaleRB->AutoSize = true;

this->addUserFemaleRB->Location = System::Drawing::Point(69, 33);

this->addUserFemaleRB->Name = L"addUserFemaleRB";

this->addUserFemaleRB->Size = System::Drawing::Size(82, 21);

this->addUserFemaleRB->TabIndex = 40;

this->addUserFemaleRB->TabStop = true;

this->addUserFemaleRB->Text = L"Женский";

this->addUserFemaleRB->UseVisualStyleBackColor = true;

//

// addUserMaleRB

//

this->addUserMaleRB->AutoSize = true;

this->addUserMaleRB->Location = System::Drawing::Point(69, 13);

this->addUserMaleRB->Name = L"addUserMaleRB";

this->addUserMaleRB->Size = System::Drawing::Size(84, 21);

this->addUserMaleRB->TabIndex = 39;

this->addUserMaleRB->TabStop = true;

this->addUserMaleRB->Text = L"Мужской";

this->addUserMaleRB->UseVisualStyleBackColor = true;

//

// addUserFormBtn

//

this->addUserFormBtn->Cursor = System::Windows::Forms::Cursors::Hand;

this->addUserFormBtn->Location = System::Drawing::Point(141, 266);

this->addUserFormBtn->Name = L"addUserFormBtn";

this->addUserFormBtn->Size = System::Drawing::Size(124, 23);

this->addUserFormBtn->TabIndex = 34;

this->addUserFormBtn->Text = L"Регистрация";

this->addUserFormBtn->UseVisualStyleBackColor = true;

this->addUserFormBtn->Click += gcnew System::EventHandler(this, &mainForm::addUserFormBtn\_Click);

//

// addPassLbl

//

this->addPassLbl->AutoSize = true;

this->addPassLbl->Location = System::Drawing::Point(80, 239);

this->addPassLbl->Name = L"addPassLbl";

this->addPassLbl->Size = System::Drawing::Size(55, 17);

this->addPassLbl->TabIndex = 32;

this->addPassLbl->Text = L"Пароль";

//

// addEmailLbl

//

this->addEmailLbl->AutoSize = true;

this->addEmailLbl->Location = System::Drawing::Point(88, 209);

this->addEmailLbl->Name = L"addEmailLbl";

this->addEmailLbl->Size = System::Drawing::Size(45, 17);

this->addEmailLbl->TabIndex = 31;

this->addEmailLbl->Text = L"e-mail";

//

// addPhonenumberLbl

//

this->addPhonenumberLbl->AutoSize = true;

this->addPhonenumberLbl->Location = System::Drawing::Point(21, 179);

this->addPhonenumberLbl->Name = L"addPhonenumberLbl";

this->addPhonenumberLbl->Size = System::Drawing::Size(114, 17);

this->addPhonenumberLbl->TabIndex = 30;

this->addPhonenumberLbl->Text = L"Номер телефона";

//

// addDateLbl

//

this->addDateLbl->AutoSize = true;

this->addDateLbl->Location = System::Drawing::Point(30, 149);

this->addDateLbl->Name = L"addDateLbl";

this->addDateLbl->Size = System::Drawing::Size(105, 17);

this->addDateLbl->TabIndex = 29;

this->addDateLbl->Text = L"Дата рождения";

//

// addUserPasswordTB

//

this->addUserPasswordTB->Location = System::Drawing::Point(141, 236);

this->addUserPasswordTB->Name = L"addUserPasswordTB";

this->addUserPasswordTB->Size = System::Drawing::Size(124, 24);

this->addUserPasswordTB->TabIndex = 25;

//

// addUserMailTB

//

this->addUserMailTB->Location = System::Drawing::Point(141, 206);

this->addUserMailTB->Name = L"addUserMailTB";

this->addUserMailTB->Size = System::Drawing::Size(124, 24);

this->addUserMailTB->TabIndex = 24;

//

// addUserNameTB

//

this->addUserNameTB->Location = System::Drawing::Point(141, 48);

this->addUserNameTB->Name = L"addUserNameTB";

this->addUserNameTB->Size = System::Drawing::Size(124, 24);

this->addUserNameTB->TabIndex = 19;

//

// addUser

//

this->addUser->AutoSize = true;

this->addUser->Font = (gcnew System::Drawing::Font(L"Agency FB", 15.75F, System::Drawing::FontStyle::Regular, System::Drawing::GraphicsUnit::Point,

static\_cast<System::Byte>(0)));

this->addUser->Location = System::Drawing::Point(125, 11);

this->addUser->Name = L"addUser";

this->addUser->Size = System::Drawing::Size(138, 25);

this->addUser->TabIndex = 37;

this->addUser->Text = L"Регистрация";

//

// mainForm

//

this->AutoScaleDimensions = System::Drawing::SizeF(6, 13);

this->AutoScaleMode = System::Windows::Forms::AutoScaleMode::Font;

this->ClientSize = System::Drawing::Size(388, 420);

this->Controls->Add(this->lgnPanel);

this->Controls->Add(this->userRegistrationPanel);

this->MaximizeBox = false;

this->Name = L"mainForm";

this->ShowIcon = false;

this->Text = L"Вход";

this->FormClosing += gcnew System::Windows::Forms::FormClosingEventHandler(this, &mainForm::mainForm\_FormClosing);

this->Load += gcnew System::EventHandler(this, &mainForm::mainForm\_Load);

this->lgnPanel->ResumeLayout(false);

this->lgnPanel->PerformLayout();

this->userRegistrationPanel->ResumeLayout(false);

this->userRegistrationPanel->PerformLayout();

this->addUserSexGP->ResumeLayout(false);

this->addUserSexGP->PerformLayout();

this->ResumeLayout(false);

}

#pragma endregion

private: System::Void mainForm\_Load(System::Object^ sender, System::EventArgs^ e);

private: System::Void textBox3\_TextChanged(System::Object^ sender, System::EventArgs^ e) {

}

private: System::Void panel3\_Paint(System::Object^ sender, System::Windows::Forms::PaintEventArgs^ e) {

}

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

}

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

private: System::Void availableTime\_SelectedIndexChanged(System::Object^ sender, System::EventArgs^ e) {

}

private: System::Void clientPanel\_Paint(System::Object^ sender, System::Windows::Forms::PaintEventArgs^ e) {

}

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

}

private: System::Void clientIDCB\_SelectedIndexChanged(System::Object^ sender, System::EventArgs^ e) {

}

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

}

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

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

private: System::Void mainForm\_FormClosing(System::Object^ sender, System::Windows::Forms::FormClosingEventArgs^ e);

};

}

#endif MAINFORM\_H

**Код исходных файлов.**

**clientForm.cpp**

-------------------

#include "clientForm.h"

MockDatabase\* DBClientForm = MockDatabase::getInstance();

static std::map<int, int> indexOfEmployeesMenu;

static std::map<int, int> indexOfClientsMenu;

static std::map<int, int> indexOfService;

System::Int32 bshop::clientForm::findID(int Index, bool isEmployee)

{

int res = 0;

if (isEmployee) {

for (const auto& kv : indexOfEmployeesMenu) {

if (kv.first == Index)

return res = kv.second;

}

}

else

{

for (const auto& kv : indexOfClientsMenu) {

if (kv.first == Index)

return res = kv.second;

}

}

return -1;

}

System::Void bshop::clientForm::clientForm\_FormClosing(System::Object^ sender, System::Windows::Forms::FormClosingEventArgs^ e)

{

Application::Exit();

}

System::Void bshop::clientForm::clientForm\_Load(System::Object^ sender, System::EventArgs^ e)

{

Calendar->SetDate(Calendar->TodayDate);

Calendar->MinDate = Calendar->TodayDate;

updateServiceOrderBtn\_Click(sender, e);

ClientNameLbl->Text = msclr::interop::marshal\_as<System::String^>(DBClientForm->getUser(ID)->getName());

}

#pragma region Работа с панелью клиента

System::Void bshop::clientForm::updateAvailableTimes()

{

availableTimeLB->Items->Clear();

std::string date = std::to\_string((Calendar->SelectionStart).Year) + "-" +

std::to\_string((Calendar->SelectionStart).Month) + "-" +

std::to\_string((Calendar->SelectionStart).Day);

std::list<std::string> times = DBClientForm->getOrdersTimes(date);

if (times.size() == 0)

{

timeStatusLbl->Text = "Всё забронировано!";

}

for each (std::string time in times)

{

availableTimeLB->Items->Add(msclr::interop::marshal\_as<System::String^>(time));

}

}

System::Void bshop::clientForm::acceptOrderBtn\_Click(System::Object^ sender, System::EventArgs^ e)

{

/\*std::string date = std::to\_string((Calendar->SelectionStart).Day) + "/" +

std::to\_string((Calendar->SelectionStart).Month) + "/" +

std::to\_string((Calendar->SelectionStart).Year);\*/

double cost;

int clientID = this->ID;

int employeeID;

int serviceID;

std::string time;

std::string date;

try

{

if (availableTimeLB->SelectedItem == nullptr ||

serviceIDCB->Text == "" ||

employeeIDCB->Text == "" ||

//clientIDCB->Text == "" ||

availableTimeLB->Items->Count == 0 ||

costTB->Text == "")

{

MessageBox::Show("Ошибка ввода!", "Ошибка!", MessageBoxButtons::OK, MessageBoxIcon::Error);

return;

}

date = std::to\_string((Calendar->SelectionStart).Year) + "-" +

std::to\_string((Calendar->SelectionStart).Month) + "-" +

std::to\_string((Calendar->SelectionStart).Day);

time = msclr::interop::marshal\_as<std::string>(availableTimeLB->SelectedItem->ToString());

int serviceDIndex = serviceIDCB->SelectedIndex;

if (serviceDIndex == -1)

{

serviceDIndex = serviceIDCB->Items->IndexOf(employeeIDCB->Text);

}

serviceID = indexOfService[serviceDIndex];

//serviceID = std::stoi(msclr::interop::marshal\_as<std::string>(serviceIDCB->Text));

int employeeIndex = employeeIDCB->SelectedIndex;

if (employeeIndex == -1)

{

employeeIndex = employeeIDCB->Items->IndexOf(employeeIDCB->Text);

}

employeeID = findID(employeeIndex, true);

cost = std::stod(msclr::interop::marshal\_as<std::string>(costTB->Text));

}

catch (std::exception ex)

{

MessageBox::Show("Ошибка ввода!", "Ошибка!", MessageBoxButtons::OK, MessageBoxIcon::Error);

}

DateTime dateDT((Calendar->SelectionStart).Year, (Calendar->SelectionStart).Month, (Calendar->SelectionStart).Day);

if (DBClientForm->getOrdersCount(date) >= 8)

{

MessageBox::Show("На данный день все места забронированы!", "Ошибка!", MessageBoxButtons::OK, MessageBoxIcon::Error);

return;

}

else

{

DBClientForm->addServiceOrder(date, time, serviceID, employeeID, clientID, cost, false);

updateAvailableTimes();

if (DBClientForm->getOrdersCount(date) >= 8)

{

Calendar->AddBoldedDate(dateDT);

Calendar->UpdateBoldedDates();

}

}

DBClientForm->writeServicesOrdersToFile();

}

System::Void bshop::clientForm::exitFromClientPanel\_Click(System::Object^ sender, System::EventArgs^ e)

{

DBClientForm->writeServicesOrdersToFile();

this->Close();

Application::Exit();

}

System::Void bshop::clientForm::Calendar\_DateSelected(System::Object^ sender, System::Windows::Forms::DateRangeEventArgs^ e)

{

updateAvailableTimes();

}

System::Void bshop::clientForm::serviceIDCB\_SelectionChangeCommitted(System::Object^ sender, System::EventArgs^ e)

{

int serviceDIndex = serviceIDCB->SelectedIndex;

if (serviceDIndex == -1)

{

serviceDIndex = serviceIDCB->Items->IndexOf(employeeIDCB->Text);

}

//int serviceID = std::stoi(msclr::interop::marshal\_as<std::string>(serviceIDCB->SelectedItem->ToString()));

int serviceID = indexOfService[serviceDIndex];

double cost = DBClientForm->getService(serviceID)->getPrice();

Client\* curClient = (Client\*)DBClientForm->getUser(this->ID);

if (this->userStatus == CLIENT)

{

cost = Math::Round((cost - ((cost / 100) \* curClient->getClientDiscount())), 2);

String^ costString = cost.ToString();

//costTB->Text = msclr::interop::marshal\_as<System::String^>(std::to\_string(cost));

costTB->Text = costString;

}

else

{

MessageBox::Show("Вы не клиент!", "Ошибка!", MessageBoxButtons::OK, MessageBoxIcon::Error);

return;

}

}

System::Void bshop::clientForm::updateServiceOrderBtn\_Click(System::Object^ sender, System::EventArgs^ e)

{

std::list<ServiceOrder\*> ServicesOrders = DBClientForm->getServiceOrdersList();

employeeIDCB->Items->Clear();

serviceIDCB->Items->Clear();

orderInitGrid->Rows->Clear();

int i = 0;

for each (ServiceOrder \* serviceOrder in ServicesOrders)

{

if (serviceOrder->getClientID() == ID)

{

std::string ID = std::to\_string(serviceOrder->getID());

Employee\* selectedEmployee = (Employee\*)DBClientForm->getUser(serviceOrder->getEmployeeID());

std::string employeeName = selectedEmployee != nullptr ? selectedEmployee->getName() :

"ПОЛЬЗОВАТЕЛЬ УДАЛЁН, ID:" + std::to\_string(serviceOrder->getEmployeeID());

Client\* selectedClient = (Client\*)DBClientForm->getUser(serviceOrder->getClientID());

std::string clientName = selectedClient != nullptr ? selectedClient->getName() :

"ПОЛЬЗОВАТЕЛЬ УДАЛЁН, ID:" + std::to\_string(serviceOrder->getClientID());

Service\* selectedService = (Service\*)DBClientForm->getService(serviceOrder->getServiceID());

std::string serviceName = selectedService != nullptr ? selectedService->getName() :

"УСЛУГА УДАЛЕНА, ID:" + std::to\_string(serviceOrder->getServiceID());

std::string cost = std::to\_string(serviceOrder->getCost());

std::string data = serviceOrder->getDate();

std::string time = serviceOrder->getTime();

std::string status = serviceOrder->getStatus().Equals(true) ? "Проведён" : "Не проведён";

std::vector<std::string> v = { ID, serviceName, employeeName,clientName, cost, status, data, time };

orderInitGrid->Rows->Add();

int k = orderInitGrid->ColumnCount;

for (int j = 0; j < k; j++)

{

orderInitGrid->Rows[i]->Cells[j]->Value = msclr::interop::marshal\_as<System::String^>(v[j]);

}

i++;

}

}

std::list<Service\*> Services = DBClientForm->getServiceList();

std::list<User\*> Users = DBClientForm->getUsersList();

for each (Service \* service in Services)

{

String^ serviceName = msclr::interop::marshal\_as<System::String^>(service->getName());

serviceIDCB->Items->Add(serviceName);

indexOfService[serviceIDCB->Items->Count - 1] = service->getID();

}

for each (User \* user in Users)

{

if (user->getStatus() == EMPLOYEE)

{

String^ userName = msclr::interop::marshal\_as<System::String^>(user->getName());

employeeIDCB->Items->Add(userName);

indexOfEmployeesMenu[(employeeIDCB->Items->Count - 1)] = user->getID();

}

if (user->getStatus() == CLIENT)

{

indexOfClientsMenu[(employeeIDCB->Items->Count - 1)] = user->getID();

}

}

}

#pragma endregion

-------------------

**Client.cpp**

-------------------

#include "Client.h"

std::string Client::getClientStatus()

{

switch (this->clientStatus) {

case REGULAR:

return "regular";

break;

case ONCE:

return "once";

break;

case NEW:

return "new";

break;

case VIP:

return "VIP";

break;

default:

return "none";

break;

}

}

int Client::getClientDiscount()

{

return (int)this->clientStatus;

}

int Client::getNumberOfVisits()

{

return this->numberOfVisits;

}

void Client::setClientStatus(clientStatusType clientStatus)

{

this->clientStatus = clientStatus;

}

void Client::addVisit()

{

}

void Client::getService()

{

}

std::string Client::getInfo()

{

std::string info = "{" + std::to\_string(getID()) + ";" + getName() + ";" + getSex() + ";" + getDate() + ";" + getPhonenumber() + ";" +

getEmail() + ";" + std::to\_string(getPassword()) + ";" + getStatusString() + ";" +

getClientStatus() + ";" + std::to\_string(getNumberOfVisits()) + "}" + "\n";

return info;

}

-------------------

**checkFunctions.cpp**

-------------------

#include "checkFunctions.h"

bool checkSex(std::string sex)

{

if (sex == "male") return true;

if (sex == "female") return false;

return false;

}

std::string statusToString(userType statusString)

{

if (statusString == EMPLOYEE) return "EMPLOYEE";

if (statusString == ADMIN) return "ADMIN";

if (statusString == CLIENT) return "CLIENT";

return "NONE";

}

std::string specialityToString(userSpeciality speciality)

{

if (speciality == BARBER) return "Барбер";

if (speciality == ADMINISTRATOR) return "Администратор";

if (speciality == CLEANER) return "Уборщик";

if (speciality == DIRECTOR) return "Директор";

return "NONE";

}

userType checkSelectedStatus(std::string statusString)

{

if (statusString == "EMPLOYEE" || statusString == "employee") return EMPLOYEE;

if (statusString == "ADMIN" || statusString == "admin") return ADMIN;

if (statusString == "CLIENT" || statusString == "client") return CLIENT;

return CLIENT;

}

userSpeciality checkSelectedSpeciality(std::string selectedSpeciality)

{

if (selectedSpeciality == "BARBER" || selectedSpeciality == "barber") return BARBER;

if (selectedSpeciality == "ADMINISTRATOR" || selectedSpeciality == "administrator") return ADMINISTRATOR;

if (selectedSpeciality == "CLEANER" || selectedSpeciality == "cleaner") return CLEANER;

if (selectedSpeciality == "DIRECTOR" || selectedSpeciality == "director") return DIRECTOR;

return DEF\_USER\_SPECIALITY;

}

clientStatusType checkClientStatus(int numberOfVisits)

{

if (numberOfVisits == 0) return NEW;

if ((numberOfVisits != 0) && (numberOfVisits <= 3)) return ONCE;

if (numberOfVisits >= 4) return REGULAR;

if (numberOfVisits > 10) return VIP;

return NEW;

}

-------------------

**adminForm.cpp**

-------------------

//Aphonin, Ankudinov, Timurziev, Dmitriev

#include "adminForm.h"

#include <Windows.h>

#include "checkFunctions.h"

#include "status.h"

#include <time.h>

#include <ctime>

#include <sstream>

#include <vector>

#include <fstream>

#include <iomanip>

MockDatabase\* DBAdminForm = MockDatabase::getInstance();

#pragma region Работа с пользователями из панели админа

System::Void bshop::adminForm::clearAllGrids()

{

usersDataGrid->Rows->Clear();

servicesDataGrid->Rows->Clear();

ordersDataGrid->Rows->Clear();

}

std::vector<std::string> splitToElems(const std::string& s, char delim) {

std::stringstream ss(s);

std::string item;

std::vector<std::string> elems;

while (std::getline(ss, item, delim)) {

elems.push\_back(item);

}

return elems;

}

System::Void bshop::adminForm::addUserBtn\_Click(System::Object^ sender, System::EventArgs^ e)

{

mainTabSekector->SelectTab(addUserTab);

/\*if (!userRegistrationPanel->Visible)

{

userRegistrationPanel->Visible = true;

userRegistrationPanel->Enabled = true;

}

else

{

userRegistrationPanel->Visible = false;

userRegistrationPanel->Enabled = false;

}\*/

}

System::Void bshop::adminForm::addUserTypeCB\_SelectionChangeCommitted(System::Object^ sender, System::EventArgs^ e)

{

}

System::Void bshop::adminForm::addUserFormBtn\_Click(System::Object^ sender, System::EventArgs^ e)

{

std::string name = msclr::interop::marshal\_as<std::string>(addUserNameTB->Text);

bool sex;

if (addUserMaleRB->Checked)

{

sex = true;

}

else

{

sex = false;

}

double experience = NULL;

std::string personalAchievements = "";

std::string date = msclr::interop::marshal\_as<std::string>(addUserDateTB->Text);

std::string phonenumber = msclr::interop::marshal\_as<std::string>(addUserPhonenumberTB->Text);

std::string email = msclr::interop::marshal\_as<std::string>(addUserMailTB->Text);

std::string password = msclr::interop::marshal\_as<std::string>(addUserPasswordTB->Text);

std::string statusString = msclr::interop::marshal\_as<std::string>(addUserTypeCB->Text);

if (name == "" || email == "" || statusString == "" || password == "")

{

MessageBox::Show("Есть пустые поля!", "Ошибка!", MessageBoxButtons::OK, MessageBoxIcon::Error);

return;

}

std::string selectedSpeciality;

if (addUserTypeCB->Text == "EMPLOYEE")

{

std::string selectedSpeciality = msclr::interop::marshal\_as<std::string>(addUserSpecialityCB->Text);

}

else

{

specialityPanel->Visible = false;

}

userType status = checkSelectedStatus(statusString);

userSpeciality speciality = checkSelectedSpeciality(selectedSpeciality);

if (DBAdminForm->emailIsUnique(email))

switch (status) {

case EMPLOYEE:

if (experience.Equals(NULL)) experience = DEF\_WORK\_EXPERIENCE;

if (personalAchievements.empty()) personalAchievements = DEF\_PERSONAL\_ACHIEVEMENTS;

DBAdminForm->addUser(name, sex, date, phonenumber, email, password, status, experience,

speciality, personalAchievements);

DBAdminForm->writeDatabaseToFile();

break;

case ADMIN:

DBAdminForm->addUser(name, sex, date, phonenumber, email, password, status);

DBAdminForm->writeDatabaseToFile();

break;

case CLIENT:

DBAdminForm->addUser(name, sex, date, phonenumber, email, password, status);

DBAdminForm->writeDatabaseToFile();

break;

default:

System::Windows::Forms::MessageBox::Show("There is no such status");

break;

}

else System::Windows::Forms::MessageBox::Show("Пользователь с таким e-mail уже существует");

}

System::Void bshop::adminForm::addUserTypeCB\_SelectedIndexChanged(System::Object^ sender, System::EventArgs^ e)

{

if (addUserTypeCB->Text == "EMPLOYEE")

{

specialityPanel->Visible = true;

specialityPanel->Enabled = true;

}

else

{

specialityPanel->Visible = false;

specialityPanel->Enabled = false;

}

}

System::Void bshop::adminForm::changeUserIdCB\_SelectionChangeCommitted(System::Object^ sender, System::EventArgs^ e)

{

int selectID = std::stoi(msclr::interop::marshal\_as<std::string>(changeUserIdCB->SelectedItem->ToString()));

User\* selectUser = DBAdminForm->getUser(selectID);

changeUserNameTB->Text = msclr::interop::marshal\_as<System::String^>(selectUser->getName());

changeUserMaleRB->Checked = (selectUser->getSex()) == ("true") ? true : false;

if (!changeUserMaleRB->Checked) changeUserFemaleRB->Checked = true;

changeUserDateTB->Text = msclr::interop::marshal\_as<System::String^>(selectUser->getDate());

changeUserPhonenumberTB->Text = msclr::interop::marshal\_as<System::String^>(selectUser->getPhonenumber());

changeUserMailTB->Text = msclr::interop::marshal\_as<System::String^>(selectUser->getEmail());

changeUserPasswordTB->Text = "Введите новый пароль!";

userType status = selectUser->getStatus();

changeUserTypeCB->Text = msclr::interop::marshal\_as<System::String^>(statusToString(status));

if (status == EMPLOYEE)

{

changeUserSpecialityPanel->Visible = true;

Employee\* selectEmployee = (Employee\*)selectUser;

changeUserSpecialityCB->Text = msclr::interop::marshal\_as<System::String^>(selectEmployee->getSpeciality());

changeUserExperienceTB->Text = msclr::interop::marshal\_as<System::String^>(std::to\_string(selectEmployee->getExperience()));

changePersonalAchievementsTB->Text = msclr::interop::marshal\_as<System::String^>(selectEmployee->getPersonalAchievements());

}

else

{

changeUserSpecialityPanel->Visible = false;

changeUserSpecialityPanel->Enabled = false;

}

}

System::Void bshop::adminForm::changeUserFormBtn\_Click(System::Object^ sender, System::EventArgs^ e)

{

if (changeUserNameTB->Text == "" ||

changeUserMailTB->Text == "" ||

changeUserPasswordTB->Text == "" ||

changeUserTypeCB->Text == "")

{

MessageBox::Show("Есть пустые поля!", "Ошибка!", MessageBoxButtons::OK, MessageBoxIcon::Error);

return;

}

int selectID = std::stoi(msclr::interop::marshal\_as<std::string>(changeUserIdCB->Text));

User\* selectUser = DBAdminForm->getUser(selectID);

selectUser->setName(msclr::interop::marshal\_as<std::string>(changeUserNameTB->Text));

selectUser->setSex(changeUserMaleRB->Checked);

selectUser->setDate(msclr::interop::marshal\_as<std::string>(changeUserDateTB->Text));

selectUser->setPhonenumber(msclr::interop::marshal\_as<std::string>(changeUserPhonenumberTB->Text));

selectUser->setEmail(msclr::interop::marshal\_as<std::string>(changeUserMailTB->Text));

//size\_t passwordHash = NULL;

#pragma warning(suppress : 4996)

std::string pass = msclr::interop::marshal\_as<std::string>(changeUserPasswordTB->Text);

//sscanf(msclr::interop::marshal\_as<std::string>(changeUserPasswordTB->Text).c\_str(), "%zu", &passwordHash);

std::hash<std::string> hasher;

size\_t passwordHash = hasher(pass);

selectUser->setPassword(passwordHash);

userType status = checkSelectedStatus(msclr::interop::marshal\_as<std::string>(changeUserTypeCB->Text));

selectUser->setStatus(status);

if (status == EMPLOYEE)

{

Employee\* selectEmployee = (Employee\*)selectUser;

selectEmployee->setSpeciality(checkSelectedSpeciality(msclr::interop::marshal\_as<std::string>(changeUserSpecialityCB->Text)));

selectEmployee->setExperience(std::stoi(msclr::interop::marshal\_as<std::string>(changeUserExperienceTB->Text)));

selectEmployee->setPersonalAchievements(msclr::interop::marshal\_as<std::string>(changePersonalAchievementsTB->Text));

DBAdminForm->writeDatabaseToFile();

}

}

System::Void bshop::adminForm::changeUserTypeCB\_SelectedIndexChanged(System::Object^ sender, System::EventArgs^ e)

{

if (changeUserTypeCB->Text == "EMPLOYEE")

{

changeUserSpecialityPanel->Visible = true;

changeUserSpecialityPanel->Enabled = true;

}

else

{

changeUserSpecialityPanel->Visible = false;

changeUserSpecialityPanel->Enabled = false;

}

}

System::Void bshop::adminForm::changeUserBtn\_Click(System::Object^ sender, System::EventArgs^ e)

{

mainTabSekector->SelectTab(changeUserTab);

/\*if (!changeUserPanel->Visible)

{

changeUserPanel->Visible = true;

changeUserPanel->Enabled = true;

}

else

{

changeUserPanel->Visible = false;

changeUserPanel->Enabled = false;

}\*/

}

System::Void bshop::adminForm::deleteSpecificUser\_Click(System::Object^ sender, System::EventArgs^ e)

{

if (mainTabSekector->SelectedTab != mainTab)

{

System::Windows::Forms::MessageBox::Show("Перейдите на главную панель!");

return;

}

if (usersDataGrid->SelectedRows->Count == 0)

{

System::Windows::Forms::MessageBox::Show("Вы ничего не выбрали!");

return;

}

bool isDel = false;

for (int i = 0; i < usersDataGrid->Rows->Count; i++)

{

if (usersDataGrid->Rows[i]->Selected == true)

{

int userID = std::stoi(msclr::interop::marshal\_as<std::string>(usersDataGrid->Rows[i]->Cells[0]->Value->ToString()));

try

{

isDel = DBAdminForm->deleteSpecificUser(userID);

}

catch (std::exception ex)

{

String^ exception = msclr::interop::marshal\_as<System::String^>(ex.what());

System::Windows::Forms::MessageBox::Show(exception);

}

}

}

if (isDel)

{

System::Windows::Forms::MessageBox::Show("Удаление прошло успешно!");

DBAdminForm->writeDatabaseToFile();

}

}

#pragma endregion

#pragma region Работа с услугами из панели админа

System::Void bshop::adminForm::addServiceBtn\_Click(System::Object^ sender, System::EventArgs^ e)

{

std::string name = msclr::interop::marshal\_as<std::string>(serviceNameTB->Text);

int price = std::stoi(msclr::interop::marshal\_as<std::string>(servicePriceTB->Text));

try

{

DBAdminForm->addService(name, price);

DBAdminForm->writeServicesToFile();

}

catch (std::exception ex)

{

String^ exception = msclr::interop::marshal\_as<System::String^>(ex.what());

System::Windows::Forms::MessageBox::Show(exception);

}

}

System::Void bshop::adminForm::saveServicesBtn\_Click(System::Object^ sender, System::EventArgs^ e)

{

DBAdminForm->writeServicesToFile();

}

System::Void bshop::adminForm::loadServicesBtn\_Click(System::Object^ sender, System::EventArgs^ e)

{

DBAdminForm->clearListOfServices();

DBAdminForm->readServicesFromFile();

}

System::Void bshop::adminForm::updateServicesBtn\_Click(System::Object^ sender, System::EventArgs^ e)

{

servicesRTB->Clear();

try

{

String^ info = msclr::interop::marshal\_as<System::String^>(DBAdminForm->getServices());

servicesRTB->AppendText(info);

}

catch (std::exception ex)

{

String^ exception = msclr::interop::marshal\_as<System::String^>(ex.what());

System::Windows::Forms::MessageBox::Show(exception);

}

}

System::Void bshop::adminForm::changeServiceBtn\_Click(System::Object^ sender, System::EventArgs^ e)

{

if (mainTabSekector->SelectedTab != mainTab)

{

System::Windows::Forms::MessageBox::Show("Перейдите на главную панель!");

return;

}

if (servicesDataGrid->SelectedRows->Count == 0)

{

System::Windows::Forms::MessageBox::Show("Вы ничего не выбрали!");

return;

}

mainTabSekector->SelectTab(changeServiceTab);

/\*if (!changeServicePanel->Visible)

{

changeServicePanel->Visible = true;

changeServicePanel->Enabled = true;

}

else

{

changeServicePanel->Visible = false;

changeServicePanel->Enabled = false;

}\*/

}

System::Void bshop::adminForm::changeServiceFormBtn\_Click(System::Object^ sender, System::EventArgs^ e)

{

for (int i = 0; i < servicesDataGrid->Rows->Count; i++)

{

if (servicesDataGrid->Rows[i]->Selected == true)

{

std::string name = msclr::interop::marshal\_as<std::string>(changeServiceNameTB->Text);

int price = std::stoi(msclr::interop::marshal\_as<std::string>(changeServicePriceTB->Text));

int serviseID = std::stoi(msclr::interop::marshal\_as<std::string>(servicesDataGrid->Rows[i]->Cells[0]->Value->ToString()));

try

{

Service\* buffService = DBAdminForm->getService(serviseID);

buffService->setName(name);

buffService->setPrice(price);

DBAdminForm->writeServicesToFile();

}

catch (std::exception ex)

{

String^ exception = msclr::interop::marshal\_as<System::String^>(ex.what());

System::Windows::Forms::MessageBox::Show(exception);

}

}

}

changeServicePanel->Visible = false;

changeServicePanel->Enabled = false;

}

System::Void bshop::adminForm::delOrderBtn\_Click(System::Object^ sender, System::EventArgs^ e)

{

if (mainTabSekector->SelectedTab != mainTab)

{

System::Windows::Forms::MessageBox::Show("Перейдите на главную панель!");

return;

}

if (servicesDataGrid->SelectedRows->Count == 0)

{

System::Windows::Forms::MessageBox::Show("Вы ничего не выбрали!");

return;

}

bool isDel = false;

for (int i = 0; i < servicesDataGrid->Rows->Count; i++)

{

if (servicesDataGrid->Rows[i]->Selected == true)

{

int serviseID = std::stoi(msclr::interop::marshal\_as<std::string>(servicesDataGrid->Rows[i]->Cells[0]->Value->ToString()));

try

{

isDel = DBAdminForm->deleteSpecificService(serviseID);

}

catch (std::exception ex)

{

String^ exception = msclr::interop::marshal\_as<System::String^>(ex.what());

System::Windows::Forms::MessageBox::Show(exception);

}

}

}

if (isDel)

{

System::Windows::Forms::MessageBox::Show("Удаление прошло успешно!");

DBAdminForm->writeServicesToFile();

}

}

System::Void bshop::adminForm::addServiceBtnFromAdminPanel\_Click(System::Object^ sender, System::EventArgs^ e)

{

mainTabSekector->SelectTab(addServiceTab);

/\*if (!addServicePanel->Visible)

{

addServicePanel->Visible = true;

addServicePanel->Enabled = true;

}

else

{

addServicePanel->Visible = false;

addServicePanel->Enabled = false;

}\*/

}

#pragma endregion

System::Void bshop::adminForm::updateGridsBtn\_Click(System::Object^ sender, System::EventArgs^ e)

{

clearAllGrids();

std::list<ServiceOrder\*> ServicesOrders = DBAdminForm->getServiceOrdersList();

std::list<Service\*> Services = DBAdminForm->getServiceList();

std::list<User\*> Users = DBAdminForm->getUsersList();

for each (User \* user in Users)

{

std::string ID = std::to\_string(user->getID());

std::string Name = user->getName();

std::string status = statusToString(user->getStatus());

std::vector<std::string> v = { ID, Name, status };

usersDataGrid->Rows->Add();

int i = usersDataGrid->Rows->Count;

int k = usersDataGrid->ColumnCount;

for (int j = 0; j < k; j++)

{

usersDataGrid->Rows[i - 1]->Cells[j]->Value = msclr::interop::marshal\_as<System::String^>(v[j]);

}

}

for each (Service \* service in Services)

{

std::string ID = std::to\_string(service->getID());

std::string Name = service->getName();

std::string cost = std::to\_string(service->getPrice());

std::vector<std::string> v = { ID, Name, cost };

servicesDataGrid->Rows->Add();

int i = servicesDataGrid->Rows->Count;

int k = servicesDataGrid->ColumnCount;

for (int j = 0; j < k; j++)

{

servicesDataGrid->Rows[i - 1]->Cells[j]->Value = msclr::interop::marshal\_as<System::String^>(v[j]);

}

}

for each (ServiceOrder \* serviceOrder in ServicesOrders)

{

std::string ID = std::to\_string(serviceOrder->getID());

Employee\* selectedEmployee = (Employee\*)DBAdminForm->getUser(serviceOrder->getEmployeeID());

std::string employeeName = selectedEmployee != nullptr ? selectedEmployee->getName() :

"ПОЛЬЗОВАТЕЛЬ УДАЛЁН, ID:" + std::to\_string(serviceOrder->getEmployeeID());

Client\* selectedClient = (Client\*)DBAdminForm->getUser(serviceOrder->getClientID());

std::string clientName = selectedClient != nullptr ? selectedClient->getName() :

"ПОЛЬЗОВАТЕЛЬ УДАЛЁН, ID:" + std::to\_string(serviceOrder->getClientID());

Service\* selectedService = (Service\*)DBAdminForm->getService(serviceOrder->getServiceID());

std::string serviceName = selectedService != nullptr ? selectedService->getName() :

"УСЛУГА УДАЛЕНА, ID:" + std::to\_string(serviceOrder->getServiceID());

std::string status = serviceOrder->getStatus().Equals(true) ? "Проведён" : "Не проведён";

std::vector<std::string> v = { ID, employeeName, clientName, status, serviceName } ;

ordersDataGrid->Rows->Add();

int i = ordersDataGrid->Rows->Count;

int k = ordersDataGrid->ColumnCount;

for (int j = 0; j < k; j++)

{

ordersDataGrid->Rows[i - 1]->Cells[j]->Value = msclr::interop::marshal\_as<System::String^>(v[j]);

}

}

}

System::Void bshop::adminForm::writeDataBtn\_Click(System::Object^ sender, System::EventArgs^ e)

{

DBAdminForm->writeDatabaseToFile();

DBAdminForm->writeServicesOrdersToFile();

DBAdminForm->writeServicesToFile();

}

System::Void bshop::adminForm::loadBtn\_Click(System::Object^ sender, System::EventArgs^ e)

{

DBAdminForm->clearListOfServices();

DBAdminForm->clearListOfServicesOrder();

DBAdminForm->clearListOfUsers();

DBAdminForm->readDatabaseFromFile();

DBAdminForm->readServicesFromFile();

DBAdminForm->readServicesOrdersFromFile();

}

System::Void bshop::adminForm::deleteSpecidicServiceOrderBtn\_Click(System::Object^ sender, System::EventArgs^ e)

{

if (mainTabSekector->SelectedTab != mainTab)

{

System::Windows::Forms::MessageBox::Show("Перейдите на главную панель!");

return;

}

if (ordersDataGrid->SelectedRows->Count == 0)

{

System::Windows::Forms::MessageBox::Show("Вы ничего не выбрали!");

return;

}

bool isDel = false;

for (int i = 0; i < ordersDataGrid->Rows->Count; i++)

{

if (ordersDataGrid->Rows[i]->Selected == true)

{

int serviseOrderID = std::stoi(msclr::interop::marshal\_as<std::string>(ordersDataGrid->Rows[i]->Cells[0]->Value->ToString()));

try

{

isDel = DBAdminForm->deleteSpecificServiceOrder(serviseOrderID);

}

catch (std::exception ex)

{

String^ exception = msclr::interop::marshal\_as<System::String^>(ex.what());

System::Windows::Forms::MessageBox::Show(exception);

}

}

}

if (isDel)

{

System::Windows::Forms::MessageBox::Show("Удаление прошло успешно!");

DBAdminForm->writeServicesOrdersToFile();

}

}

System::Void bshop::adminForm::showReportBtn\_Click(System::Object^ sender, System::EventArgs^ e)

{

mainTabSekector->SelectTab(showReportTab);

/\*if (!reportPanel->Visible)

{

reportPanel->Visible = true;

reportPanel->Enabled = true;

}

else

{

reportPanel->Visible = false;

reportPanel->Enabled = false;

}\*/

}

System::Void bshop::adminForm::makeReportBtn\_Click(System::Object^ sender, System::EventArgs^ e)

{

tm Tm;

time\_t t = std::time(nullptr);

errno\_t err = localtime\_s(&Tm, &t);

int curYear = Tm.tm\_year + 1900;

int curMonth = Tm.tm\_mon + 1;

int curDay = Tm.tm\_mday;

int clientsNumber = 0;

double salary = 0;

double proceeds = 0;

double Profit = 0;

double averageСheck = 0;

if (reportPeriodCB->Text == "Год")

{

for each (ServiceOrder \* serviceOrder in DBAdminForm->getServiceOrdersList())

{

std::string date = serviceOrder->getDate();

std::vector <std::string> elems = splitToElems(date, '-');

int buffYear = std::stoi(elems[0]);

int buffMonth = std::stoi(elems[1]);

int buffDay = std::stoi(elems[2]);

if ((buffYear == curYear && buffMonth <= curMonth && buffDay <= curDay) ||

(buffYear == curYear - 1 && buffMonth >= curMonth && buffDay >= buffDay))

{

clientsNumber++;

proceeds += serviceOrder->getCost();

}

}

for each (User \* user in DBAdminForm->getUsersList())

{

if (user->getStatus() == EMPLOYEE)

{

Employee\* buffEmployee = (Employee\*)user;

salary += buffEmployee->getSalary() \* 12;

}

}

Profit = proceeds - salary;

averageСheck = proceeds / clientsNumber;

String^ yearInfo = "";

if (ExpensesCB->Checked == true)

{

yearInfo += "Затраты за год составили: " + salary.ToString() + "\n";

};

if (proceedsCB->Checked == true)

{

yearInfo += "Выручка за год составила: " + proceeds.ToString() + "\n";

};

if (profitCB->Checked == true)

{

yearInfo += "Прибыль за год составила: " + Profit.ToString() + "\n";

};

if (clientsNumberCB->Checked == true)

{

yearInfo += "Количество клиентов за год составило: " + clientsNumber.ToString() + "\n";

};

if (averageСheckCB->Checked == true)

{

yearInfo += "Средний чек за год составил: " + averageСheck.ToString() + "\n";

};

makeReportRTB->Text = yearInfo;

}

if (reportPeriodCB->Text == "Месяц")

{

String^ monthInfo = "";

for each (ServiceOrder \* serviceOrder in DBAdminForm->getServiceOrdersList())

{

std::string date = serviceOrder->getDate();

std::vector <std::string> elems = splitToElems(date, '-');

int buffYear = std::stoi(elems[0]);

int buffMonth = std::stoi(elems[1]);

int buffDay = std::stoi(elems[2]);

if ((buffYear == curYear && buffMonth == curMonth - 1 && buffDay >= curDay) ||

(buffYear == curYear && buffMonth == curMonth && buffDay <= curDay))

{

clientsNumber++;

proceeds += serviceOrder->getCost();

}

}

for each (User \* user in DBAdminForm->getUsersList())

{

if (user->getStatus() == EMPLOYEE)

{

Employee\* buffEmployee = (Employee\*)user;

salary += buffEmployee->getSalary() \* 1;

}

}

Profit = proceeds - salary;

averageСheck = proceeds / clientsNumber;

if (ExpensesCB->Checked == true)

{

monthInfo += "Затраты за месяц составили: " + salary.ToString() + "\n";

};

if (proceedsCB->Checked == true)

{

monthInfo += "Выручка за месяц составила: " + proceeds.ToString() + "\n";

};

if (profitCB->Checked == true)

{

monthInfo += "Прибыль за месяц составила: " + Profit.ToString() + "\n";

};

if (clientsNumberCB->Checked == true)

{

monthInfo += "Количество клиентов за месяц составило: " + clientsNumber.ToString() + "\n";

};

if (averageСheckCB->Checked == true)

{

monthInfo += "Средний чек за месяц составил: " + averageСheck.ToString() + "\n";

};

makeReportRTB->Text = monthInfo;

}

if (reportPeriodCB->Text == "День")

{

String^ dayInfo = "";

for each (ServiceOrder \* serviceOrder in DBAdminForm->getServiceOrdersList())

{

std::string date = serviceOrder->getDate();

std::vector <std::string> elems = splitToElems(date, '-');

int buffYear = std::stoi(elems[0]);

int buffMonth = std::stoi(elems[1]);

int buffDay = std::stoi(elems[2]);

if (buffDay == curDay)

{

clientsNumber++;

proceeds += serviceOrder->getCost();

}

}

for each (User \* user in DBAdminForm->getUsersList())

{

if (user->getStatus() == EMPLOYEE)

{

Employee\* buffEmployee = (Employee\*)user;

salary += buffEmployee->getSalary() / 12;

}

}

Profit = proceeds - salary;

averageСheck = proceeds / clientsNumber;

if (ExpensesCB->Checked == true)

{

dayInfo += "Затраты за текущий день составили: неизвестно""\n";

};

if (proceedsCB->Checked == true)

{

dayInfo += "Выручка за текущий день составила: " + proceeds.ToString() + "\n";

};

if (profitCB->Checked == true)

{

dayInfo += "Прибыль за текущий день составила: неизвестно" + "\n";

};

if (clientsNumberCB->Checked == true)

{

dayInfo += "Количество клиентов за текущий день составило: " + clientsNumber.ToString() + "\n";

};

if (averageСheckCB->Checked == true)

{

dayInfo += "Средний чек за текущий день составил: " + averageСheck.ToString() + "\n";

};

makeReportRTB->Text = dayInfo;

}

}

System::Void bshop::adminForm::mainTabSekector\_Selected(System::Object^ sender, System::Windows::Forms::TabControlEventArgs^ e)

{

changeUserIdCB->Items->Clear();

for each (User \* user in DBAdminForm->getUsersList())

{

changeUserIdCB->Items->Add(user->getID());

}

}

System::Void bshop::adminForm::adminForm\_FormClosing(System::Object^ sender, System::Windows::Forms::FormClosingEventArgs^ e)

{

Application::Exit();

}

-------------------

**User.cpp**

-------------------

#include "User.h"

using namespace std;

User::User(int id, std::string name, bool sex, std::string date, std::string phonenumber, std::string email, size\_t password, userType status)

{

this->id= id;

this->name = name;

this->date = date;

this->sex = sex;

this->email = email;

this->phonenumber = phonenumber;

this->password = password;

this->status = status;

}

int User::getID()

{

return this->id;

}

std::string User::getDate()

{

return this->date;

}

size\_t User::getPassword()

{

return this->password;

}

std::string User::getInfo()

{

std::string info = "{" + std::to\_string(getID()) + ";" + getName() + ";" + getSex() + ";" + getDate() + ";" + getPhonenumber() + ";" + getEmail() + ";" + std::to\_string(getPassword()) + ";" + getStatusString() + "}" + "\n";

return info;

}

std::string User::getName()

{

return this->name;

}

std::string User::getSex()

{

std::string sex = "";

sex = this->sex ? "male" : "female";

return sex;

}

std::string User::getPhonenumber()

{

return this->phonenumber;

}

std::string User::getEmail()

{

return this->email;

}

userType User::getStatus()

{

return this->status;

}

std::string User::getStatusString()

{

switch (this->status) {

case EMPLOYEE:

return "employee";

break;

case ADMIN:

return "admin";

break;

case CLIENT:

return "client";

break;

default:

return "none";

break;

}

}

void User::setName(std::string name)

{

this->name = name;

}

void User::setSex(bool sex)

{

this->sex = sex;

}

void User::setPhonenumber(std::string phonenumber)

{

this->phonenumber = phonenumber;

}

void User::setEmail(std::string email)

{

this->email = email;

}

void User::setStatus(userType status)

{

this->status = status;

}

void User::setDate(std::string date)

{

this->date = date;

}

void User::setPassword(size\_t password)

{

this->password = password;

}

//bool operator==(User& left, const std::string& right)

//{

// if (left.email == right)

// return true;

// return false;

//}

//

//bool operator!=(User& left, const std::string& right)

//{

// if (left.email != right)

// return true;

// return false;

//}

//

//bool operator==(User& left, const std::string right)

//{

// if (left.email == right)

// return true;

// return false;

//}

//

//bool operator!=(User& left, const std::string right)

//{

// if (left.email != right)

// return true;

// return false;

//}

-------------------

**ServiceOrder.cpp**

-------------------

#include "ServiceOrder.h"

ServiceOrder::ServiceOrder(int ID, int serviceID, int employeeID, int clientID, std::string date, std::string time, double serviceCostWithDiscount, bool status)

{

this->ID = ID;

this->serviceID = serviceID;

this->employeeID = employeeID;

this->clientID = clientID;

this->date = date;

this->time = time;

this->status = status;

this->serviceCostWithDiscount = serviceCostWithDiscount;

}

std::string ServiceOrder::getTime()

{

return this->time;

}

int ServiceOrder::getID()

{

return this->ID;

}

std::string ServiceOrder::getDate()

{

return this->date;

}

double ServiceOrder::getCost()

{

return this->serviceCostWithDiscount;

}

void ServiceOrder::setStatus(bool status)

{

this->status = status;

}

bool ServiceOrder::getStatus()

{

return this->status;

}

double ServiceOrder::getServiceCostWithDiscount()

{

return this->serviceCostWithDiscount;

}

int ServiceOrder::getClientID()

{

return this->clientID;

}

int ServiceOrder::getEmployeeID()

{

return this->employeeID;

}

int ServiceOrder::getServiceID()

{

return this->serviceID;

}

std::string ServiceOrder::getInfo()

{

std::string status = getStatus().Equals(true) ? "true": "false";

std::string info = "{" + std::to\_string(getID()) + ";" + std::to\_string(getServiceID()) + ";" +

std::to\_string(getEmployeeID()) + ";" + std::to\_string(getClientID()) + ";" +

std::to\_string(getCost()) + ";" + status + ";" + getDate() + ";" + getTime() + "}" + "\n";

return info;

}

std::string ServiceOrder::getName()

{

return std::string();

}

-------------------

**Service.cpp**

-------------------

#include "Service.h"

Service::Service(int ID, std::string name, int price)

{

this->ID = ID;

this->name = name;

this->price = price;

}

int Service::getID()

{

return this->ID;

}

void Service::setPrice(int price)

{

this->price = price;

}

void Service::setName(std::string name)

{

this->name = name;

}

int Service::getPrice()

{

return this->price;

}

std::string Service::getInfo()

{

std::string info = "{" + std::to\_string(getID()) + ";" + getName() + ";" + std::to\_string(getPrice()) + "}" + "\n";

return info;

}

std::string Service::getName()

{

return this->name;

}

-------------------

**MockDatabase.cpp**

-------------------

#include "MockDatabase.h"

#include <string>

#include <sstream>

#include <vector>

#include <iostream>

#include <fstream>

#include <iomanip>

#define ARGUMENTS\_NUM\_EMPLOYEE 11

#define ARGUMENTS\_NUM\_ADMIN 8

#define ARGUMENTS\_NUM\_CLIENT 9

MockDatabase\* MockDatabase::db = nullptr;

std::string pathsArray[3] = { "employeeData.txt", "adminData.txt", "clientData.txt" };

std::vector<std::string> split\_to\_elem(const std::string& s, char delim) {

std::stringstream ss(s);

std::string item;

std::vector<std::string> elems;

while (std::getline(ss, item, delim)) {

elems.push\_back(item);

}

return elems;

}

void clearFilesbyPath(std::string pathsArray[])

{

std::ofstream ofs;

if (pathsArray->size() == 1)

{

std::ofstream ofs;

std::string path = pathsArray[0];

ofs.open(pathsArray[0], std::ios::trunc);

ofs.close();

}

else

{

for (int i = 0; i < pathsArray->size(); i++)

{

std::ofstream ofs;

std::string path = pathsArray[i];

ofs.open(pathsArray[i], std::ios::trunc);

ofs.close();

}

}

}

void clearFilesbyPath(std::string pathsArray)

{

std::ofstream ofs;

std::string path = pathsArray;

ofs.open(pathsArray, std::ios::trunc);

ofs.close();

}

#pragma region Work with user

void MockDatabase::addUser(std::string name, bool sex, std::string date, std::string phonenumber,

std::string email, std::string password, userType status, int numberOfVisits)

{

std::hash<std::string> hasher;

size\_t passwordHash = hasher(password);

//int id = this->listOfUsers.size() + 1;

int id = getMaxUserID() + 1;

switch (status) {

case ADMIN:

this->listOfUsers.push\_back(new Admin(id, name, sex, date, phonenumber, email, passwordHash, ADMIN));

break;

case CLIENT:

this->listOfUsers.push\_back(new Client(id, name, sex, date, phonenumber, email, passwordHash,

CLIENT, numberOfVisits, checkClientStatus(numberOfVisits)));

break;

default:

System::Windows::Forms::MessageBox::Show("There is no such status");

break;

}

}

void MockDatabase::addUser(std::string name, bool sex, std::string date, std::string phonenumber, std::string email,

std::string password, userType status, double experience, userSpeciality speciality, std::string personalAchievements)

{

std::hash<std::string> hasher;

size\_t passwordHash = hasher(password);

//int id = this->listOfUsers.size() + 1;

int id = getMaxUserID() + 1;

this->listOfUsers.push\_back(new Employee(id, name, sex, date, phonenumber, email, passwordHash,

EMPLOYEE, experience, speciality, personalAchievements));

}

void MockDatabase::loadUser(int ID, std::string name, bool sex, std::string date, std::string phonenumber,

std::string email, std::string password, userType status, int numberOfVisits)

{

size\_t passwordHash = NULL;

#pragma warning(suppress : 4996)

sscanf(password.c\_str(), "%zu", &passwordHash);

/\*std::hash<std::string> hasher;

size\_t passwordHash = hasher(password);\*/

switch (status) {

case ADMIN:

this->listOfUsers.push\_back(new Admin(ID, name, sex, date, phonenumber, email, passwordHash, ADMIN));

break;

case CLIENT:

this->listOfUsers.push\_back(new Client(ID, name, sex, date, phonenumber, email, passwordHash,

CLIENT, numberOfVisits, checkClientStatus(numberOfVisits)));

break;

default:

System::Windows::Forms::MessageBox::Show("There is no such status");

break;

}

}

void MockDatabase::loadUser(int ID, std::string name, bool sex, std::string date, std::string phonenumber,

std::string email, std::string password, userType status, double experience,

userSpeciality speciality, std::string personalAchievements)

{

size\_t passwordHash = NULL;

#pragma warning(suppress : 4996)

sscanf(password.c\_str(), "%zu", &passwordHash);

/\*std::hash<std::string> hasher;

size\_t passwordHash = hasher(password);\*/

this->listOfUsers.push\_back(new Employee(ID, name, sex, date, phonenumber, email, passwordHash,

EMPLOYEE, experience, speciality, personalAchievements));

}

bool MockDatabase::timeIsAvailable(std::string date, std::string time)

{

bool isAvailable = true;

for each (ServiceOrder \* serviceOrder in this->listOfServiceOrders)

{

if ((serviceOrder->getDate() == date) && (serviceOrder->getTime() == time))

{

isAvailable = false;

}

}

return isAvailable;

}

std::list<std::string> MockDatabase::getOrdersTimes(std::string date)

{

std::list<std::string> buff;

for each (std::string workingTime in availableTime)

{

if (timeIsAvailable(date, workingTime))

{

buff.push\_back(workingTime);

}

}

return buff;

}

Service\* MockDatabase::getService(int ID)

{

int id = ID;

for each (Service \* service in listOfService)

{

if (service->getID() == id)

{

return service;

}

}

return nullptr;

}

ServiceOrder\* MockDatabase::getServiceOrder(int ID)

{

int id = ID;

for each (ServiceOrder \* serviceOrder in listOfServiceOrders)

{

if (serviceOrder->getID() == id)

{

return serviceOrder;

}

}

return nullptr;

}

User\* MockDatabase::getUser(int ID)

{

int id = ID;

for each (User \* serviceOrder in listOfUsers)

{

if (serviceOrder->getID() == id)

{

return serviceOrder;

}

}

return nullptr;

}

MockDatabase\* MockDatabase::getInstance()

{

if (!db)

db = new MockDatabase();

return db;

}

//MockDatabase\* MockDatabase::db = nullptr;

//MockDatabase\* MockDatabase::getInstance()

//{

// if (!db)

// db = new MockDatabase();

// return db;

//}

std::string MockDatabase::getUsers()

{

std::string info = "";

for each (User \* user in this->listOfUsers)

{

info += user->getInfo();

}

return info;

}

void MockDatabase::clearListOfUsers()

{

listOfUsers.clear();

}

std::list<User\*> MockDatabase::getUsersList()

{

std::list<User\*> buff;

for each (User\* user in this->listOfUsers)

{

buff.push\_back(user);

}

return buff;

}

std::list<ServiceOrder\*> MockDatabase::getServiceOrdersList()

{

std::list<ServiceOrder\*> buff;

for each (ServiceOrder \* order in this->listOfServiceOrders)

{

buff.push\_back(order);

}

return buff;

}

std::list<Service\*> MockDatabase::getServiceList()

{

std::list<Service\*> buff;

for each (Service \* service in this->listOfService)

{

buff.push\_back(service);

}

return buff;

}

int MockDatabase::numberOfUsers()

{

return static\_cast<int>(this->listOfUsers.size());

}

int MockDatabase::findUser(std::string email, size\_t password)

{

bool isExist = false;

for each (User \* user in this->listOfUsers)

{

bool isCorrect = false;

isExist = user->getEmail() == email;

isCorrect = password.Equals(user->getPassword());

if (isExist)

{

if (isCorrect)

{

System::Windows::Forms::MessageBox::Show("Добро пожаловать!");

return user->getID();

}

else

{

System::Windows::Forms::MessageBox::Show("Неверный пароль!");

return NOT\_FOUND;

}

}

}

if (!isExist)

{

System::Windows::Forms::MessageBox::Show("Такого пользователя нет!");

return NOT\_FOUND;

}

return NOT\_FOUND;

}

bool MockDatabase::emailIsUnique(std::string email)

{

std::string emailS = email;

std::list<User\*>::iterator findIter = std::find\_if(listOfUsers.begin(), listOfUsers.end(),

[&email](User\* user) { return user->email == email;});

bool res = (listOfUsers.end() == findIter) || (listOfUsers.size() == 0) ? true : false;

return res;

}

int MockDatabase::getUserStatus(int ID)

{

std::string emailS = email;

std::list<User\*>::iterator findIter = std::find\_if(listOfUsers.begin(), listOfUsers.end(), [&ID](User\* user)

{

if (user->getID() == ID)

{

return true;

}

return false;

});

if (findIter != listOfUsers.end())

{

return (\*findIter)->getStatus();

}

else

{

return NOT\_FOUND;

}

}

bool writeDataAboutUser(std::string path, User\* user)

{

std::ofstream fout;

fout.open(path, std::ios::app);

if (!fout.is\_open())

{

MessageBox::Show("Ошибка открытия файла: " + msclr::interop::marshal\_as<System::String^>(path), "Ошибка!", MessageBoxButtons::OK, MessageBoxIcon::Error);

return false;

}

else

{

fout << user->getInfo();

return true;

}

fout.close();

}

bool MockDatabase::deleteSpecificUser(int ID)

{

bool isDelete = false;

for each (User \* user in this->listOfUsers)

{

if (user->getID() == ID)

{

listOfUsers.remove(user);

isDelete = true;

return isDelete;

}

}

return isDelete;

}

bool MockDatabase::deleteSpecificService(int ID)

{

bool isDelete = false;

for each (Service \* service in this->listOfService)

{

if (service->getID() == ID)

{

listOfService.remove(service);

isDelete = true;

return isDelete;

}

}

return isDelete;

}

bool MockDatabase::deleteSpecificServiceOrder(int ID)

{

bool isDelete = false;

for each (ServiceOrder \* serviceOrder in this->listOfServiceOrders)

{

if (serviceOrder->getID() == ID)

{

listOfServiceOrders.remove(serviceOrder);

isDelete = true;

return isDelete;

}

}

return isDelete;

}

std::list<int> MockDatabase::getOrdersID(Employee\* employee)

{

std::list<int> orders;

if (employee != nullptr)

{

for each (ServiceOrder \* serviceOrder in listOfServiceOrders)

{

if (serviceOrder->getEmployeeID() == employee->getID()) orders.push\_back(serviceOrder->getID());

}

}

return orders;

}

int MockDatabase::getMaxUserID()

{

if (listOfUsers.empty()) return 0;

int maxID = -1;

for each (User \* user in this->listOfUsers)

{

int userID = user->getID();

if (user->getID() > maxID) maxID = userID;

}

return maxID;

}

bool MockDatabase::readDataAboutUser(userType status, std::vector<std::string> elems)

{

int ID = std::stoi(elems[0]);

std::string name = elems[1];

bool sex = checkSex(elems[2]);

std::string date = elems[3];

std::string phonenumber = elems[4];

std::string email = elems[5];

std::string password = elems[6];

userSpeciality speciality;

double experience = NULL;

int numberOfVisits = NULL;

std::string personalAchievements = "";

switch (status) {

case EMPLOYEE:

speciality = checkSelectedSpeciality(elems[8]);

experience = std::stod(elems[9]);

personalAchievements = (elems[10]);

loadUser(ID, name, sex, date, phonenumber,

email, password, status, experience, speciality, personalAchievements);

return true;

break;

case ADMIN:

loadUser(ID, name, sex, date, phonenumber,

email, password, status);

return true;

break;

case CLIENT:

numberOfVisits = std::stoi(elems[9]);

loadUser(ID, name, sex, date, phonenumber,

email, password, status, numberOfVisits);

return true;

break;

default:

MessageBox::Show("Ошибка при считывании файла!", "Ошибка!", MessageBoxButtons::OK, MessageBoxIcon::Error);

return false;

break;

}

}

bool MockDatabase::readDataAboutServiceOrder(std::vector<std::string> elems)

{

int ID = std::stoi(elems[0]);

int serviceID = std::stoi(elems[1]);

int employeeID = std::stoi(elems[2]);

int clientID = std::stoi(elems[3]);

double cost = std::stod(elems[4]);

bool status = elems[5] == "true" ? true : false;

std::string date = elems[6];

std::string time = elems[7];

try

{

loadServiceOrder(ID, date, time, serviceID, employeeID, clientID, cost, status);

return true;

}

catch (std::exception ex)

{

return false;

}

}

#pragma endregion

#pragma region Work with service

void MockDatabase::loadService(int ID, std::string name, int price)

{

this->listOfService.push\_back(new Service(ID, name, price));

}

void MockDatabase::addService(std::string name, int price)

{

int id = getMaxServiceID() + 1;

this->listOfService.push\_back(new Service(id, name, price));

}

void MockDatabase::addServiceOrder(std::string date, std::string time, int serviceID, int employeeID, int clientID, double cost, bool status)

{

int id = getMaxServiceOrderID() + 1;

this->listOfServiceOrders.push\_back(new ServiceOrder(id, serviceID, employeeID, clientID, date, time, cost, status));

}

void MockDatabase::loadServiceOrder(int id, std::string date, std::string time, int serviceID, int employeeID, int clientID, double cost, bool status)

{

this->listOfServiceOrders.push\_back(new ServiceOrder(id, serviceID, employeeID, clientID, date, time, cost, status));

}

int MockDatabase::numberOfServices()

{

return static\_cast<int>(this->listOfService.size());

}

int MockDatabase::numberOfServicesOrder()

{

return static\_cast<int>(this->listOfServiceOrders.size());

}

bool writeDataAboutServices(std::string path, Service\* service)

{

std::ofstream fout;

fout.open(path, std::ios::app);

if (!fout.is\_open())

{

MessageBox::Show("Ошибка открытия файла: " + msclr::interop::marshal\_as<System::String^>(path), "Ошибка!", MessageBoxButtons::OK, MessageBoxIcon::Error);

return false;

}

else

{

fout << service->getInfo();

return true;

}

fout.close();

}

bool writeDataAboutServicesOrder(std::string path, ServiceOrder\* serviceOrder)

{

std::ofstream fout;

fout.open(path, std::ios::app);

if (!fout.is\_open())

{

MessageBox::Show("Ошибка открытия файла: " + msclr::interop::marshal\_as<System::String^>(path), "Ошибка!", MessageBoxButtons::OK, MessageBoxIcon::Error);

return false;

}

else

{

fout << serviceOrder->getInfo();

return true;

}

fout.close();

}

void MockDatabase::writeServicesToFile()

{

if (numberOfServices().Equals(0))

{

MessageBox::Show("Данных нет!\nНечего записывать", "Внимание!", MessageBoxButtons::OK, MessageBoxIcon::Information);

return;

}

else

{

clearFilesbyPath("services\_data.txt");

bool isEntrySuccessful = false;

for each (Service \* Service in this->listOfService)

{

isEntrySuccessful = writeDataAboutServices("services\_data.txt", Service);

}

if (isEntrySuccessful)

{

//MessageBox::Show("Данные успешно сохранены", "Сообщение!", MessageBoxButtons::OK, MessageBoxIcon::Information);

}

}

}

void MockDatabase::readServicesFromFile()

{

const char delim = '\n';

std::string path = "services\_data.txt";

std::ifstream fin;

bool isReadSuccessful = false;

fin.open(path);

if (!fin.is\_open())

{

MessageBox::Show("Ошибка открытия файла", "Объявление!", MessageBoxButtons::OK, MessageBoxIcon::Error);

return;

}

else

{

if (fin.peek() == EOF)

{

isReadSuccessful = true;

}

else

{

while (!fin.eof())

{

std::string buffString;

getline(fin, buffString, delim);

try

{

buffString = buffString.substr(1, buffString.size() - 2);

}

catch (std::exception ex)

{

continue;

}

std::vector <std::string> elems = split\_to\_elem(buffString, ';');

if (elems.size() >= 3)

{

isReadSuccessful = readDataAboutService(elems);

}

}

}

if (!isReadSuccessful)

{

MessageBox::Show("Ошибка открытия файла: " + msclr::interop::marshal\_as<System::String^>(path), "Ошибка!", MessageBoxButtons::OK, MessageBoxIcon::Error);

}

}

fin.close();

if (isReadSuccessful)

{

//MessageBox::Show("Данные успешно сохранены", "Сообщение!", MessageBoxButtons::OK, MessageBoxIcon::Information);

}

}

bool MockDatabase::readDataAboutService(std::vector<std::string> elems)

{

int ID = std::stoi(elems[0]);

std::string name = elems[1];

int price = std::stoi(elems[2]);

try

{

loadService(ID, name, price);

return true;

}

catch (std::exception ex)

{

return false;

}

}

void MockDatabase::clearListOfServices()

{

listOfService.clear();

}

void MockDatabase::clearListOfServicesOrder()

{

listOfServiceOrders.clear();

}

int MockDatabase::getMaxServiceID()

{

if (listOfService.empty()) return 0;

int maxID = -1;

for each (Service \* service in this->listOfService)

{

int userID = service->getID();

if (service->getID() > maxID) maxID = userID;

}

return maxID;

}

int MockDatabase::getMaxServiceOrderID()

{

if (listOfServiceOrders.empty()) return 0;

int maxID = -1;

for each (ServiceOrder \* serviceOrder in this->listOfServiceOrders)

{

int serviceOrderID = serviceOrder->getID();

if (serviceOrder->getID() > maxID) maxID = serviceOrderID;

}

return maxID;

}

std::string MockDatabase::getServices()

{

std::string info = "";

for each (Service \* service in this->listOfService)

{

info += service->getInfo();

}

return info;

}

int MockDatabase::getOrdersCount(std::string date)

{

int ordersInDay = 0;

for each (ServiceOrder \* serviceOrder in this->listOfServiceOrders)

{

if (serviceOrder->getDate() == date)

{

ordersInDay++;

}

}

return ordersInDay;

}

#pragma endregion

#pragma region Work with database

void MockDatabase::writeDatabaseToFile()

{

if (numberOfUsers().Equals(0))

{

MessageBox::Show("Данных нет!\nНечего записывать", "Внимание!", MessageBoxButtons::OK, MessageBoxIcon::Information);

return;

}

else

{

clearFilesbyPath(pathsArray);

bool isEntrySuccessful = false;

for each (User \* user in this->listOfUsers)

{

switch (user->status) {

case EMPLOYEE:

isEntrySuccessful = writeDataAboutUser("employeeData.txt", user);

break;

case ADMIN:

isEntrySuccessful = writeDataAboutUser("adminData.txt", user);

break;

case CLIENT:

isEntrySuccessful = writeDataAboutUser("clientData.txt", user);

break;

default:

isEntrySuccessful = writeDataAboutUser("none\_undefined.txt", user);

break;

}

}

if (isEntrySuccessful)

{

//MessageBox::Show("Данные успешно сохранены", "Сообщение!", MessageBoxButtons::OK, MessageBoxIcon::Information);

}

}

}

void MockDatabase::readDatabaseFromFile()

{

const char delim = '\n';

std::string path;

std::ifstream fin;

bool isReadSuccessful = false;

for each (std::string path in pathsArray)

{

fin.open(path);

if (!fin.is\_open())

{

MessageBox::Show("Ошибка открытия файла", "Объявление!", MessageBoxButtons::OK, MessageBoxIcon::Error);

return;

}

else

{

if (fin.peek() == EOF)

{

isReadSuccessful = true;

}

else

{

while (!fin.eof())

{

std::string buffString;

getline(fin, buffString, delim);

try

{

buffString = buffString.substr(1, buffString.size() - 2);

}

catch (std::exception ex)

{

continue;

}

std::vector <std::string> elems = split\_to\_elem(buffString, ';');

if (elems.size() >= 7)

{

if (path == "employeeData.txt")

isReadSuccessful = readDataAboutUser(EMPLOYEE, elems);

else if (path == "adminData.txt")

isReadSuccessful = readDataAboutUser(ADMIN, elems);

else if (path == "clientData.txt")

isReadSuccessful = readDataAboutUser(CLIENT, elems);

}

}

}

if (!isReadSuccessful)

{

MessageBox::Show("Ошибка открытия файла: " + msclr::interop::marshal\_as<System::String^>(path), "Ошибка!", MessageBoxButtons::OK, MessageBoxIcon::Error);

}

}

fin.close();

}

if (isReadSuccessful)

{

//MessageBox::Show("Данные успешно сохранены", "Сообщение!", MessageBoxButtons::OK, MessageBoxIcon::Information);

}

}

void MockDatabase::writeServicesOrdersToFile()

{

if (numberOfServicesOrder().Equals(0))

{

MessageBox::Show("Данных нет!\nНечего записывать", "Внимание!", MessageBoxButtons::OK, MessageBoxIcon::Information);

return;

}

else

{

clearFilesbyPath("services\_orders\_data.txt");

bool isEntrySuccessful = false;

for each (ServiceOrder \* serviceOrder in this->listOfServiceOrders)

{

isEntrySuccessful = writeDataAboutServicesOrder("services\_orders\_data.txt", serviceOrder);

}

if (isEntrySuccessful)

{

//MessageBox::Show("Данные успешно сохранены", "Сообщение!", MessageBoxButtons::OK, MessageBoxIcon::Information);

}

}

}

void MockDatabase::readServicesOrdersFromFile()

{

const char delim = '\n';

std::string path = "services\_orders\_data.txt";

std::ifstream fin;

bool isReadSuccessful = false;

fin.open(path);

if (!fin.is\_open())

{

MessageBox::Show("Ошибка открытия файла", "Объявление!", MessageBoxButtons::OK, MessageBoxIcon::Error);

return;

}

else

{

if (fin.peek() == EOF)

{

isReadSuccessful = true;

}

else

{

while (!fin.eof())

{

std::string buffString;

getline(fin, buffString, delim);

try

{

buffString = buffString.substr(1, buffString.size() - 2);

}

catch (std::exception ex)

{

continue;

}

std::vector <std::string> elems = split\_to\_elem(buffString, ';');

if (elems.size() >= 3)

{

isReadSuccessful = readDataAboutServiceOrder(elems);

}

}

}

if (!isReadSuccessful)

{

MessageBox::Show("Ошибка открытия файла: " + msclr::interop::marshal\_as<System::String^>(path), "Ошибка!", MessageBoxButtons::OK, MessageBoxIcon::Error);

}

}

fin.close();

if (isReadSuccessful)

{

//MessageBox::Show("Данные успешно сохранены", "Сообщение!", MessageBoxButtons::OK, MessageBoxIcon::Information);

}

}

#pragma endregion

-------------------

**mainForm.cpp**

-------------------

#include "menuForm.h"

#include <Windows.h>

#include "checkFunctions.h"

#include "MockDatabase.h"

#include "status.h"

#include <time.h>

#include <ctime>

#include <sstream>

#include <vector>

#include <fstream>

#include <iomanip>

MockDatabase\* db2 = MockDatabase::getInstance();

Employee\* currentEmployee;

std::list<ServiceOrder\*> curOrders;

static std::map<int, int> indexOfEmployeesMenu;

static std::map<int, int> indexOfClientsMenu;

System::Void bshop::menuForm::clearAllGrids()

{

usersDataGrid->Rows->Clear();

servicesDataGrid->Rows->Clear();

ordersDataGrid->Rows->Clear();

}

std::vector<std::string> split(const std::string& s, char delim) {

std::stringstream ss(s);

std::string item;

std::vector<std::string> elems;

while (std::getline(ss, item, delim)) {

elems.push\_back(item);

}

return elems;

}

System::Void bshop::menuForm::closeAllPanels(System::Object^ sender, System::EventArgs^ e)

{

reportPanel->Visible = false;

reportPanel->Enabled = false;

adminPanel->Visible = false;

adminPanel->Enabled = false;

employeePanel->Visible = false;

employeePanel->Enabled = false;

clientPanel->Visible = false;

clientPanel->Enabled = false;

changeUserPanel->Visible = false;

changeUserPanel->Enabled = false;

userRegistrationPanel->Visible = false;

userRegistrationPanel->Enabled = false;

changeServicePanel->Visible = false;

changeServicePanel->Enabled = false;

addServicePanel->Visible = false;

addServicePanel->Enabled = false;

}

System::Void bshop::menuForm::menuForm\_Load(System::Object^ sender, System::EventArgs^ e)

{

closeAllPanels(sender, e);

switch (userStatus) {

case ADMIN:

adminPanel->Visible = true;

adminPanel->Enabled = true;

//outputRTB->Text = "ADMIN";

break;

case CLIENT:

//outputRTB->Text = "CLIENT";

clientPanel->Visible = true;

clientPanel->Enabled = true;

Calendar->SetDate(employeeCalendar->TodayDate);

updateServiceOrderBtn\_Click(sender, e);

ClientNameLbl->Text = msclr::interop::marshal\_as<System::String^>(db2->getUser(ID)->getName());

break;

case EMPLOYEE:

//outputRTB->Text = "EMPLOYEE";

employeePanel->Visible = true;

employeePanel->Enabled = true;

employeeCalendar->SetDate(employeeCalendar->TodayDate);

updateTimetable(employeeCalendar, selectedDateEmployeeTB);

updateOrdersGrid(curOrders);

break;

default:

//outputRTB->Text = "NONE";

System::Windows::Forms::MessageBox::Show("Пользователь имеет неизвестный статус");

break;

}

}

System::Void bshop::menuForm::updateAvailableTimes()

{

availableTimeLB->Items->Clear();

std::string date = std::to\_string((Calendar->SelectionStart).Year) + "-" +

std::to\_string((Calendar->SelectionStart).Month) + "-" +

std::to\_string((Calendar->SelectionStart).Day);

std::list<std::string> times = db2->getOrdersTimes(date);

if (times.size() == 0)

{

timeStatusLbl->Text = "Всё забронировано!";

}

for each (std::string time in times)

{

availableTimeLB->Items->Add(msclr::interop::marshal\_as<System::String^>(time));

}

}

System::Void bshop::menuForm::updateBtn\_Click(System::Object^ sender, System::EventArgs^ e)

{

std::list<ServiceOrder\*> ServicesOrders = db2->getServiceOrdersList();

std::list<Service\*> Services = db2->getServiceList();

std::list<User\*> Users = db2->getUsersList();

for each (User \* user in Users)

{

std::string ID = std::to\_string(user->getID());

std::string Name = user->getName();

std::string status = statusToString(user->getStatus());

std::vector<std::string> v = { ID, Name, status };

usersDataGrid->Rows->Add();

int i = usersDataGrid->Rows->Count;

int k = usersDataGrid->ColumnCount;

for (int j = 1; j < k; j++)

{

usersDataGrid->Rows[i - 1]->Cells[j]->Value = msclr::interop::marshal\_as<System::String^>(v[j - 1]);

}

}

for each (Service \* service in Services)

{

std::string ID = std::to\_string(service->getID());

std::string Name = service->getName();

std::vector<std::string> v = { ID, Name };

servicesDataGrid->Rows->Add();

int i = servicesDataGrid->Rows->Count;

int k = servicesDataGrid->ColumnCount;

for (int j = 1; j < k; j++)

{

servicesDataGrid->Rows[i - 1]->Cells[j]->Value = msclr::interop::marshal\_as<System::String^>(v[j - 1]);

}

}

for each (ServiceOrder \* serviceOrder in ServicesOrders)

{

std::string ID = std::to\_string(serviceOrder->getID());

std::string employeeName = db2->getUser(serviceOrder->getEmployeeID())->getName();

std::string clientName = db2->getUser(serviceOrder->getClientID())->getName();

std::string status = serviceOrder->getStatus().Equals(true) ? "Проведён" : "Не проведён";

std::vector<std::string> v = { ID, employeeName, clientName, status };

ordersDataGrid->Rows->Add();

int i = ordersDataGrid->Rows->Count;

int k = ordersDataGrid->ColumnCount;

for (int j = 1; j < k; j++)

{

ordersDataGrid->Rows[i - 1]->Cells[j]->Value = msclr::interop::marshal\_as<System::String^>(v[j - 1]);

}

}

}

System::Int32 bshop::menuForm::findID(int Index, bool isEmployee)

{

int res = 0;

if (isEmployee) {

for (const auto& kv : indexOfEmployeesMenu) {

if (kv.first == Index)

return res = kv.second;

}

}

else

{

for (const auto& kv : indexOfClientsMenu) {

if (kv.first == Index)

return res = kv.second;

}

}

return -1;

}

System::Void bshop::menuForm::addUserBtn\_Click(System::Object^ sender, System::EventArgs^ e)

{

std::list<ServiceOrder\*> ServicesOrders = db2->getServiceOrdersList();

std::list<Service\*> Services = db2->getServiceList();

std::list<User\*> Users = db2->getUsersList();

for each (User \* user in Users)

{

std::string ID = std::to\_string(user->getID());

std::string Name = user->getName();

std::string status = statusToString(user->getStatus());

std::vector<std::string> v = { ID, Name, status };

usersDataGrid->Rows->Add();

int i = usersDataGrid->Rows->Count;

int k = usersDataGrid->ColumnCount;

for (int j = 1; j < k; j++)

{

usersDataGrid->Rows[i - 1]->Cells[j]->Value = msclr::interop::marshal\_as<System::String^>(v[j - 1]);

}

}

for each (Service \* service in Services)

{

std::string ID = std::to\_string(service->getID());

std::string Name = service->getName();

std::vector<std::string> v = { ID, Name };

usersDataGrid->Rows->Add();

int i = usersDataGrid->Rows->Count;

int k = usersDataGrid->ColumnCount;

for (int j = 1; j < k; j++)

{

usersDataGrid->Rows[i - 1]->Cells[j]->Value = msclr::interop::marshal\_as<System::String^>(v[j - 1]);

}

}

for each (ServiceOrder \* serviceOrder in ServicesOrders)

{

std::string ID = std::to\_string(serviceOrder->getID());

std::string employeeName = db2->getUser(serviceOrder->getEmployeeID())->getName();

std::string clientName = db2->getUser(serviceOrder->getClientID())->getName();

std::string status = serviceOrder->getStatus().Equals(true) ? "Проведён" : "Не проведён";

std::vector<std::string> v = { ID, employeeName, clientName, status };

usersDataGrid->Rows->Add();

int i = usersDataGrid->Rows->Count;

int k = usersDataGrid->ColumnCount;

for (int j = 1; j < k; j++)

{

usersDataGrid->Rows[i - 1]->Cells[j]->Value = msclr::interop::marshal\_as<System::String^>(v[j - 1]);

}

}

}

System::Void bshop::menuForm::menuForm\_VisibleChanged(System::Object^ sender, System::EventArgs^ e)

{

}

#pragma region Работа с панелью клиента

System::Void bshop::menuForm::clientPanel\_VisibleChanged(System::Object^ sender, System::EventArgs^ e)

{

if (clientPanel->Visible == true)

{

Calendar->MinDate = Calendar->TodayDate;

}

}

System::Void bshop::menuForm::acceptOrderBtn\_Click(System::Object^ sender, System::EventArgs^ e)

{

/\*std::string date = std::to\_string((Calendar->SelectionStart).Day) + "/" +

std::to\_string((Calendar->SelectionStart).Month) + "/" +

std::to\_string((Calendar->SelectionStart).Year);\*/

double cost;

int clientID = this->ID;

int employeeID;

int serviceID;

std::string time;

std::string date;

try

{

if (availableTimeLB->SelectedItem == nullptr ||

serviceIDCB->Text == "" ||

employeeIDCB->Text == "" ||

//clientIDCB->Text == "" ||

availableTimeLB->Items->Count == 0 ||

costTB->Text == "")

{

MessageBox::Show("Ошибка ввода!", "Ошибка!", MessageBoxButtons::OK, MessageBoxIcon::Error);

return;

}

date = std::to\_string((Calendar->SelectionStart).Year) + "-" +

std::to\_string((Calendar->SelectionStart).Month) + "-" +

std::to\_string((Calendar->SelectionStart).Day);

time = msclr::interop::marshal\_as<std::string>(availableTimeLB->SelectedItem->ToString());

serviceID = std::stoi(msclr::interop::marshal\_as<std::string>(serviceIDCB->Text));

int employeeIndex = employeeIDCB->SelectedIndex;

if (employeeIndex == -1)

{

employeeIndex = employeeIDCB->Items->IndexOf(employeeIDCB->Text);

}

employeeID = findID(employeeIndex, true);

cost = std::stod(msclr::interop::marshal\_as<std::string>(costTB->Text));

}

catch (std::exception ex)

{

MessageBox::Show("Ошибка ввода!", "Ошибка!", MessageBoxButtons::OK, MessageBoxIcon::Error);

}

DateTime dateDT((Calendar->SelectionStart).Year, (Calendar->SelectionStart).Month, (Calendar->SelectionStart).Day);

if (db2->getOrdersCount(date) >= 8)

{

MessageBox::Show("На данный день все места забронированы!", "Ошибка!", MessageBoxButtons::OK, MessageBoxIcon::Error);

return;

}

else

{

db2->addServiceOrder(date, time, serviceID, employeeID, clientID, cost, false);

updateAvailableTimes();

if (db2->getOrdersCount(date) >= 8)

{

Calendar->AddBoldedDate(dateDT);

Calendar->UpdateBoldedDates();

}

}

db2->writeServicesOrdersToFile();

}

System::Void bshop::menuForm::exitFromClientPanel\_Click(System::Object^ sender, System::EventArgs^ e)

{

db2->writeServicesOrdersToFile();

this->Close();

Application::Exit();

}

System::Void bshop::menuForm::updateAvailableTimes(System::Object^ sender, System::EventArgs^ e)

{

return System::Void();

}

System::Void bshop::menuForm::Calendar\_DateSelected(System::Object^ sender, System::Windows::Forms::DateRangeEventArgs^ e)

{

updateAvailableTimes();

}

System::Void bshop::menuForm::serviceIDCB\_SelectedValueChanged(System::Object^ sender, System::EventArgs^ e)

{

return System::Void();

}

System::Void bshop::menuForm::serviceIDCB\_SelectionChangeCommitted(System::Object^ sender, System::EventArgs^ e)

{

int serviceID = std::stoi(msclr::interop::marshal\_as<std::string>(serviceIDCB->SelectedItem->ToString()));

double cost = db2->getService(serviceID)->getPrice();

Client\* curClient = (Client\*)db2->getUser(this->ID);

if (this->userStatus == CLIENT)

{

cost = Math::Round((cost - ((cost / 100) \* curClient->getClientDiscount())), 2);

String^ costString = cost.ToString();

//costTB->Text = msclr::interop::marshal\_as<System::String^>(std::to\_string(cost));

costTB->Text = costString;

}

else

{

MessageBox::Show("Вы не клиент!", "Ошибка!", MessageBoxButtons::OK, MessageBoxIcon::Error);

return;

}

}

System::Void bshop::menuForm::updateServiceOrderBtn\_Click(System::Object^ sender, System::EventArgs^ e)

{

std::list<ServiceOrder\*> ServicesOrders = db2->getServiceOrdersList();

employeeIDCB->Items->Clear();

serviceIDCB->Items->Clear();

orderInitGrid->Rows->Clear();

int i = 0;

for each (ServiceOrder \* serviceOrder in ServicesOrders)

{

std::string ID = std::to\_string(serviceOrder->getID());

std::string employeeName = db2->getUser(serviceOrder->getEmployeeID())->getName();

std::string serviceName = db2->getService(serviceOrder->getServiceID())->getName();

std::string clientName = db2->getUser(serviceOrder->getClientID())->getName();

std::string cost = std::to\_string(serviceOrder->getCost());

std::string data = serviceOrder->getDate();

std::string status = serviceOrder->getStatus().Equals(true) ? "Проведён" : "Не проведён";

std::vector<std::string> v = { ID, serviceName, employeeName,clientName, cost, status, data };

orderInitGrid->Rows->Add();

int k = orderInitGrid->ColumnCount;

for (int j = 0; j < k; j++)

{

orderInitGrid->Rows[i]->Cells[j]->Value = msclr::interop::marshal\_as<System::String^>(v[j]);

}

i++;

}

std::list<Service\*> Services = db2->getServiceList();

std::list<User\*> Users = db2->getUsersList();

for each (Service \* service in Services)

{

serviceIDCB->Items->Add(service->getID());

}

for each (User \* user in Users)

{

if (user->getStatus() == EMPLOYEE)

{

String^ userName = msclr::interop::marshal\_as<System::String^>(user->getName());

employeeIDCB->Items->Add(userName);

indexOfEmployeesMenu[(employeeIDCB->Items->Count - 1)] = user->getID();

}

if (user->getStatus() == CLIENT)

{

indexOfClientsMenu[(employeeIDCB->Items->Count - 1)] = user->getID();

}

}

}

#pragma endregion

#pragma region Работа с панелью рабочего

System::Void bshop::menuForm::exitFromEmployeePanelBtn\_Click(System::Object^ sender, System::EventArgs^ e)

{

db2->writeServicesOrdersToFile();

this->Close();

Application::Exit();

}

System::Void bshop::menuForm::updateTimetable(MonthCalendar^ employeeCalendar, TextBox^ selectedDateEmployeeTB)

{

std::string time;

std::string date;

date = std::to\_string((employeeCalendar->SelectionStart).Year) + "-" +

std::to\_string((employeeCalendar->SelectionStart).Month) + "-" +

std::to\_string((employeeCalendar->SelectionStart).Day);

selectedDateEmployeeTB->Text = "";

selectedDateEmployeeTB->Text = msclr::interop::marshal\_as<System::String^>(date);

std::list<ServiceOrder\*> employeeServicesOrders;

std::list<int> employeeServicesOrdersID = db2->getOrdersID(currentEmployee);

for each (int ID in employeeServicesOrdersID)

{

ServiceOrder\* buffServiceOrder = db2->getServiceOrder(ID);

if (buffServiceOrder->getDate() == date)

{

employeeServicesOrders.push\_back(buffServiceOrder);

}

}

curOrders = employeeServicesOrders;

}

System::Void bshop::menuForm::employeeCalendar\_DateSelected(System::Object^ sender, System::Windows::Forms::DateRangeEventArgs^ e)

{

updateTimetable(employeeCalendar, selectedDateEmployeeTB);

}

System::Void bshop::menuForm::updateOrdersGrid(std::list<ServiceOrder\*> orders)

{

employeeDataGrid->Rows->Clear();

int i = 0;

for each (ServiceOrder \* serviceOrder in orders)

{

std::string ID = std::to\_string(serviceOrder->getID());

std::string data = serviceOrder->getDate();

std::string serviceName = db2->getService(serviceOrder->getServiceID())->getName();

std::string clientName = db2->getUser(serviceOrder->getClientID())->getName();

std::string cost = std::to\_string(serviceOrder->getCost());

std::string time = serviceOrder->getTime();

std::string status = serviceOrder->getStatus().Equals(true) ? "Выпонен" : "Не выполнен";

std::vector<std::string> v = { ID, data, serviceName, clientName, cost, time, status};

employeeDataGrid->Rows->Add();

int k = employeeDataGrid->ColumnCount;

for (int j = 0; j < k; j++)

{

employeeDataGrid->Rows[i]->Cells[j]->Value = msclr::interop::marshal\_as<System::String^>(v[j]);

}

i++;

}

}

System::Void bshop::menuForm::employeePanel\_VisibleChanged(System::Object^ sender, System::EventArgs^ e)

{

if (employeePanel->Visible == true)

{

orderEmployeeIdCB->Items->Clear();

currentEmployee = (Employee\*)db2->getUser(this->ID);

nameEmployeeTB->Text = msclr::interop::marshal\_as<System::String^>(currentEmployee->getName());

specialityEmployeeTB->Text = msclr::interop::marshal\_as<System::String^>(currentEmployee->getSpeciality());

achievementsEmployeeTB->Text = msclr::interop::marshal\_as<System::String^>(currentEmployee->getPersonalAchievements());

for each (int ID in db2->getOrdersID(currentEmployee))

{

orderEmployeeIdCB->Items->Add(ID);

}

}

}

System::Void bshop::menuForm::orderEmployeeIdCB\_SelectionChangeCommitted(System::Object^ sender, System::EventArgs^ e)

{

ServiceOrder\* buffOrder = db2->getServiceOrder((int)(orderEmployeeIdCB->SelectedItem));

orderEmployeeCheckCB->Checked = buffOrder->getStatus();

}

System::Void bshop::menuForm::orderEmployeeCheckCB\_CheckedChanged(System::Object^ sender, System::EventArgs^ e)

{

ServiceOrder\* buffOrder = db2->getServiceOrder((int)(orderEmployeeIdCB->SelectedItem));

buffOrder->setStatus(orderEmployeeCheckCB->Checked);

}

System::Void bshop::menuForm::updateOrdersGridBtn\_Click(System::Object^ sender, System::EventArgs^ e)

{

updateOrdersGrid(curOrders);

}

System::Void bshop::menuForm::employeeCalendar\_DateChanged(System::Object^ sender, System::Windows::Forms::DateRangeEventArgs^ e)

{

/\*std::string time;

std::string date;

date = std::to\_string((Calendar->SelectionStart).Year) + "-" +

std::to\_string((Calendar->SelectionStart).Month) + "-" +

std::to\_string((Calendar->SelectionStart).Day);

selectedDateEmployeeTB->Text = "";

selectedDateEmployeeTB->Text = msclr::interop::marshal\_as<System::String^>(date);

std::list<ServiceOrder\*> employeeServicesOrders;

std::list<int> employeeServicesOrdersID = db2->getOrdersID(currentEmployee);

for each (int ID in employeeServicesOrdersID)

{

ServiceOrder\* buffServiceOrder = db2->getServiceOrder(ID);

if (buffServiceOrder->getDate() == date)

{

employeeServicesOrders.push\_back(buffServiceOrder);

}

}

curOrders = employeeServicesOrders;\*/

}

#pragma endregion

#pragma region Работа с пользователями из панели админа

System::Void bshop::menuForm::addUserBtn\_Click\_1(System::Object^ sender, System::EventArgs^ e)

{

if (!userRegistrationPanel->Visible)

{

userRegistrationPanel->Visible = true;

userRegistrationPanel->Enabled = true;

}

else

{

userRegistrationPanel->Visible = false;

userRegistrationPanel->Enabled = false;

}

}

System::Void bshop::menuForm::addUserTypeCB\_SelectionChangeCommitted(System::Object^ sender, System::EventArgs^ e)

{

}

System::Void bshop::menuForm::addUserFormBtn\_Click(System::Object^ sender, System::EventArgs^ e)

{

std::string name = msclr::interop::marshal\_as<std::string>(addUserNameTB->Text);

bool sex;

if (addUserMaleRB->Checked)

{

sex = true;

}

else

{

sex = false;

}

double experience = NULL;

std::string personalAchievements = "";

std::string date = msclr::interop::marshal\_as<std::string>(addUserDateTB->Text);

std::string phonenumber = msclr::interop::marshal\_as<std::string>(addUserPhonenumberTB->Text);

std::string email = msclr::interop::marshal\_as<std::string>(addUserMailTB->Text);

std::string password = msclr::interop::marshal\_as<std::string>(addUserPasswordTB->Text);

std::string statusString = msclr::interop::marshal\_as<std::string>(addUserTypeCB->Text);

if (name == "" || email == "" || statusString == "" || password == "")

{

MessageBox::Show("Есть пустые поля!", "Ошибка!", MessageBoxButtons::OK, MessageBoxIcon::Error);

return;

}

std::string selectedSpeciality;

if (addUserTypeCB->Text == "EMPLOYEE")

{

std::string selectedSpeciality = msclr::interop::marshal\_as<std::string>(addUserSpecialityCB->Text);

}

else

{

specialityPanel->Visible = false;

}

userType status = checkSelectedStatus(statusString);

userSpeciality speciality = checkSelectedSpeciality(selectedSpeciality);

if (db2->emailIsUnique(email))

switch (status) {

case EMPLOYEE:

if (experience.Equals(NULL)) experience = DEF\_WORK\_EXPERIENCE;

if (personalAchievements.empty()) personalAchievements = DEF\_PERSONAL\_ACHIEVEMENTS;

db2->addUser(name, sex, date, phonenumber, email, password, status, experience,

speciality, personalAchievements);

db2->writeDatabaseToFile();

break;

case ADMIN:

db2->addUser(name, sex, date, phonenumber, email, password, status);

db2->writeDatabaseToFile();

break;

case CLIENT:

db2->addUser(name, sex, date, phonenumber, email, password, status);

db2->writeDatabaseToFile();

break;

default:

System::Windows::Forms::MessageBox::Show("There is no such status");

break;

}

else System::Windows::Forms::MessageBox::Show("Пользователь с таким e-mail уже существует");

}

System::Void bshop::menuForm::addUserTypeCB\_SelectedIndexChanged(System::Object^ sender, System::EventArgs^ e)

{

if (addUserTypeCB->Text == "EMPLOYEE")

{

specialityPanel->Visible = true;

specialityPanel->Enabled = true;

}

else

{

specialityPanel->Visible = false;

specialityPanel->Enabled = false;

}

}

System::Void bshop::menuForm::changeUserPanel\_VisibleChanged(System::Object^ sender, System::EventArgs^ e)

{

if (changeUserPanel->Visible == true)

{

for each (User \* user in db2->getUsersList())

{

changeUserIdCB->Items->Add(user->getID());

}

}

}

System::Void bshop::menuForm::changeUserIdCB\_SelectionChangeCommitted(System::Object^ sender, System::EventArgs^ e)

{

int selectID = std::stoi(msclr::interop::marshal\_as<std::string>(changeUserIdCB->SelectedItem->ToString()));

User\* selectUser = db2->getUser(selectID);

changeUserNameTB->Text = msclr::interop::marshal\_as<System::String^>(selectUser->getName());

changeUserMaleRB->Checked = (selectUser->getSex()) == ("true") ? true : false;

if (!changeUserMaleRB->Checked) changeUserFemaleRB->Checked = true;

changeUserDateTB->Text = msclr::interop::marshal\_as<System::String^>(selectUser->getDate());

changeUserPhonenumberTB->Text = msclr::interop::marshal\_as<System::String^>(selectUser->getPhonenumber());

changeUserMailTB->Text = msclr::interop::marshal\_as<System::String^>(selectUser->getEmail());

changeUserPasswordTB->Text = "Введите новый пароль!";

userType status = selectUser->getStatus();

changeUserTypeCB->Text = msclr::interop::marshal\_as<System::String^>(statusToString(status));

if (status == EMPLOYEE)

{

changeUserSpecialityPanel->Visible = true;

Employee\* selectEmployee = (Employee\*)selectUser;

changeUserSpecialityCB->Text = msclr::interop::marshal\_as<System::String^>(selectEmployee->getSpeciality());

changeUserExperienceTB->Text = msclr::interop::marshal\_as<System::String^>(std::to\_string(selectEmployee->getExperience()));

changePersonalAchievementsTB->Text = msclr::interop::marshal\_as<System::String^>(selectEmployee->getPersonalAchievements());

}

else

{

changeUserSpecialityPanel->Visible = false;

changeUserSpecialityPanel->Enabled = false;

}

}

System::Void bshop::menuForm::changeUserFormBtn\_Click(System::Object^ sender, System::EventArgs^ e)

{

if (changeUserNameTB->Text == "" ||

changeUserMailTB->Text == "" ||

changeUserPasswordTB->Text == "" ||

changeUserSpecialityCB->Text == "" ||

changeUserTypeCB->Text == "")

{

MessageBox::Show("Есть пустые поля!", "Ошибка!", MessageBoxButtons::OK, MessageBoxIcon::Error);

return;

}

int selectID = std::stoi(msclr::interop::marshal\_as<std::string>(changeUserIdCB->Text));

User\* selectUser = db2->getUser(selectID);

selectUser->setName(msclr::interop::marshal\_as<std::string>(changeUserNameTB->Text));

selectUser->setSex(changeUserMaleRB->Checked);

selectUser->setDate(msclr::interop::marshal\_as<std::string>(changeUserDateTB->Text));

selectUser->setPhonenumber(msclr::interop::marshal\_as<std::string>(changeUserPhonenumberTB->Text));

selectUser->setEmail(msclr::interop::marshal\_as<std::string>(changeUserMailTB->Text));

//size\_t passwordHash = NULL;

//#pragma warning(suppress : 4996)

//sscanf(msclr::interop::marshal\_as<std::string>(changeUserPasswordTB->Text).c\_str(), "%zu", &passwordHash);

std::hash<std::string> hasher;

size\_t passwordHash = hasher(msclr::interop::marshal\_as<std::string>(changeUserPasswordTB->Text));

selectUser->setPassword(passwordHash);

userType status = checkSelectedStatus(msclr::interop::marshal\_as<std::string>(changeUserTypeCB->Text));

selectUser->setStatus(status);

if (status == EMPLOYEE)

{

Employee\* selectEmployee = (Employee\*)selectUser;

selectEmployee->setSpeciality(checkSelectedSpeciality(msclr::interop::marshal\_as<std::string>(changeUserSpecialityCB->Text)));

selectEmployee->setExperience(std::stoi(msclr::interop::marshal\_as<std::string>(changeUserExperienceTB->Text)));

selectEmployee->setPersonalAchievements(msclr::interop::marshal\_as<std::string>(changePersonalAchievementsTB->Text));

db2->writeDatabaseToFile();

}

}

System::Void bshop::menuForm::changeUserTypeCB\_SelectedIndexChanged(System::Object^ sender, System::EventArgs^ e)

{

if (changeUserTypeCB->Text == "EMPLOYEE")

{

changeUserSpecialityPanel->Visible = true;

changeUserSpecialityPanel->Enabled = true;

}

else

{

changeUserSpecialityPanel->Visible = false;

changeUserSpecialityPanel->Enabled = false;

}

}

System::Void bshop::menuForm::changeUserBtn\_Click(System::Object^ sender, System::EventArgs^ e)

{

if (!changeUserPanel->Visible)

{

changeUserPanel->Visible = true;

changeUserPanel->Enabled = true;

}

else

{

changeUserPanel->Visible = false;

changeUserPanel->Enabled = false;

}

}

System::Void bshop::menuForm::deleteSpecificUser\_Click(System::Object^ sender, System::EventArgs^ e)

{

bool isDel = false;

for (int i = 0; i < usersDataGrid->Rows->Count; i++)

{

if (usersDataGrid->Rows[i]->Selected == true)

{

int userID = std::stoi(msclr::interop::marshal\_as<std::string>(usersDataGrid->Rows[i]->Cells[0]->Value->ToString()));

try

{

isDel = db2->deleteSpecificUser(userID);

}

catch (std::exception ex)

{

String^ exception = msclr::interop::marshal\_as<System::String^>(ex.what());

System::Windows::Forms::MessageBox::Show(exception);

}

}

}

if (isDel)

{

System::Windows::Forms::MessageBox::Show("Удаление прошло успешно!");

db2->writeDatabaseToFile();

}

}

#pragma endregion

#pragma region Работа с услугами из панели админа

System::Void bshop::menuForm::addServiceBtn\_Click(System::Object^ sender, System::EventArgs^ e)

{

std::string name = msclr::interop::marshal\_as<std::string>(serviceNameTB->Text);

int price = std::stoi(msclr::interop::marshal\_as<std::string>(servicePriceTB->Text));

try

{

db2->addService(name, price);

db2->writeServicesToFile();

}

catch (std::exception ex)

{

String^ exception = msclr::interop::marshal\_as<System::String^>(ex.what());

System::Windows::Forms::MessageBox::Show(exception);

}

}

System::Void bshop::menuForm::saveServicesBtn\_Click(System::Object^ sender, System::EventArgs^ e)

{

db2->writeServicesToFile();

}

System::Void bshop::menuForm::loadServicesBtn\_Click(System::Object^ sender, System::EventArgs^ e)

{

db2->clearListOfServices();

db2->readServicesFromFile();

}

System::Void bshop::menuForm::updateServicesBtn\_Click(System::Object^ sender, System::EventArgs^ e)

{

servicesRTB->Clear();

try

{

String^ info = msclr::interop::marshal\_as<System::String^>(db2->getServices());

servicesRTB->AppendText(info);

}

catch (std::exception ex)

{

String^ exception = msclr::interop::marshal\_as<System::String^>(ex.what());

System::Windows::Forms::MessageBox::Show(exception);

}

}

System::Void bshop::menuForm::changeServiceBtn\_Click(System::Object^ sender, System::EventArgs^ e)

{

if (!changeServicePanel->Visible)

{

changeServicePanel->Visible = true;

changeServicePanel->Enabled = true;

}

else

{

changeServicePanel->Visible = false;

changeServicePanel->Enabled = false;

}

}

System::Void bshop::menuForm::changeServiceFormBtn\_Click(System::Object^ sender, System::EventArgs^ e)

{

for (int i = 0; i < servicesDataGrid->Rows->Count; i++)

{

if (servicesDataGrid->Rows[i]->Selected == true)

{

std::string name = msclr::interop::marshal\_as<std::string>(changeServiceNameTB->Text);

int price = std::stoi(msclr::interop::marshal\_as<std::string>(changeServicePriceTB->Text));

int serviseID = std::stoi(msclr::interop::marshal\_as<std::string>(servicesDataGrid->Rows[i]->Cells[0]->Value->ToString()));

try

{

Service\* buffService = db2->getService(serviseID);

buffService->setName(name);

buffService->setPrice(price);

db2->writeServicesToFile();

}

catch (std::exception ex)

{

String^ exception = msclr::interop::marshal\_as<System::String^>(ex.what());

System::Windows::Forms::MessageBox::Show(exception);

}

}

}

changeServicePanel->Visible = false;

changeServicePanel->Enabled = false;

}

System::Void bshop::menuForm::delOrderBtn\_Click(System::Object^ sender, System::EventArgs^ e)

{

bool isDel = false;

for (int i = 0; i < servicesDataGrid->Rows->Count; i++)

{

if (servicesDataGrid->Rows[i]->Selected == true)

{

int serviseID = std::stoi(msclr::interop::marshal\_as<std::string>(servicesDataGrid->Rows[i]->Cells[0]->Value->ToString()));

try

{

isDel = db2->deleteSpecificService(serviseID);

}

catch (std::exception ex)

{

String^ exception = msclr::interop::marshal\_as<System::String^>(ex.what());

System::Windows::Forms::MessageBox::Show(exception);

}

}

}

if (isDel)

{

System::Windows::Forms::MessageBox::Show("Удаление прошло успешно!");

db2->writeServicesToFile();

}

}

System::Void bshop::menuForm::addServiceBtnFromAdminPanel\_Click(System::Object^ sender, System::EventArgs^ e)

{

if (!addServicePanel->Visible)

{

addServicePanel->Visible = true;

addServicePanel->Enabled = true;

}

else

{

addServicePanel->Visible = false;

addServicePanel->Enabled = false;

}

}

#pragma endregion

System::Void bshop::menuForm::updateGridsBtn\_Click(System::Object^ sender, System::EventArgs^ e)

{

clearAllGrids();

std::list<ServiceOrder\*> ServicesOrders = db2->getServiceOrdersList();

std::list<Service\*> Services = db2->getServiceList();

std::list<User\*> Users = db2->getUsersList();

for each (User \* user in Users)

{

std::string ID = std::to\_string(user->getID());

std::string Name = user->getName();

std::string status = statusToString(user->getStatus());

std::vector<std::string> v = { ID, Name, status };

usersDataGrid->Rows->Add();

int i = usersDataGrid->Rows->Count;

int k = usersDataGrid->ColumnCount;

for (int j = 0; j < k; j++)

{

usersDataGrid->Rows[i - 1]->Cells[j]->Value = msclr::interop::marshal\_as<System::String^>(v[j]);

}

}

for each (Service \* service in Services)

{

std::string ID = std::to\_string(service->getID());

std::string Name = service->getName();

std::vector<std::string> v = { ID, Name };

servicesDataGrid->Rows->Add();

int i = servicesDataGrid->Rows->Count;

int k = servicesDataGrid->ColumnCount;

for (int j = 0; j < k; j++)

{

servicesDataGrid->Rows[i - 1]->Cells[j]->Value = msclr::interop::marshal\_as<System::String^>(v[j]);

}

}

for each (ServiceOrder \* serviceOrder in ServicesOrders)

{

std::string ID = std::to\_string(serviceOrder->getID());

std::string employeeName = db2->getUser(serviceOrder->getEmployeeID())->getName();

std::string clientName = db2->getUser(serviceOrder->getClientID())->getName();

std::string status = serviceOrder->getStatus().Equals(true) ? "Проведён" : "Не проведён";

std::vector<std::string> v = { ID, employeeName, clientName, status };

ordersDataGrid->Rows->Add();

int i = ordersDataGrid->Rows->Count;

int k = ordersDataGrid->ColumnCount;

for (int j = 0; j < k; j++)

{

ordersDataGrid->Rows[i - 1]->Cells[j]->Value = msclr::interop::marshal\_as<System::String^>(v[j]);

}

}

}

System::Void bshop::menuForm::deleteSpecidicServiceOrderBtn\_Click(System::Object^ sender, System::EventArgs^ e)

{

bool isDel = false;

for (int i = 0; i < ordersDataGrid->Rows->Count; i++)

{

if (ordersDataGrid->Rows[i]->Selected == true)

{

int serviseOrderID = std::stoi(msclr::interop::marshal\_as<std::string>(ordersDataGrid->Rows[i]->Cells[0]->Value->ToString()));

try

{

isDel = db2->deleteSpecificServiceOrder(serviseOrderID);

}

catch (std::exception ex)

{

String^ exception = msclr::interop::marshal\_as<System::String^>(ex.what());

System::Windows::Forms::MessageBox::Show(exception);

}

}

}

if (isDel)

{

System::Windows::Forms::MessageBox::Show("Удаление прошло успешно!");

db2->writeServicesOrdersToFile();

}

}

System::Void bshop::menuForm::showReportBtn\_Click(System::Object^ sender, System::EventArgs^ e)

{

if (!reportPanel->Visible)

{

reportPanel->Visible = true;

reportPanel->Enabled= true;

}

else

{

reportPanel->Visible = false;

reportPanel->Enabled = false;

}

}

System::Void bshop::menuForm::makeReportBtn\_Click(System::Object^ sender, System::EventArgs^ e)

{

tm Tm;

time\_t t = std::time(nullptr);

errno\_t err = localtime\_s(&Tm, &t);

int curYear = Tm.tm\_year + 1900;

int curMonth = Tm.tm\_mon+1;

int curDay = Tm.tm\_mday;

int clientsNumber = 0;

double salary = 0;

double proceeds = 0;

double Profit = 0;

double averageСheck = 0;

if (reportPeriodCB->Text == "Год")

{

for each (ServiceOrder\* serviceOrder in db2->getServiceOrdersList())

{

std::string date = serviceOrder->getDate();

std::vector <std::string> elems = split(date, '-');

int buffYear = std::stoi(elems[0]);

int buffMonth = std::stoi(elems[1]);

int buffDay = std::stoi(elems[2]);

if ((buffYear == curYear && buffMonth <= curMonth && buffDay <= curDay) ||

(buffYear == curYear - 1 && buffMonth >= curMonth && buffDay >= buffDay))

{

clientsNumber++;

proceeds += serviceOrder->getCost();

}

}

for each (User \* user in db2->getUsersList())

{

if (user->getStatus() == EMPLOYEE)

{

Employee\* buffEmployee = (Employee\*)user;

salary += buffEmployee->getSalary() \* 12;

}

}

Profit = proceeds - salary;

averageСheck = proceeds / clientsNumber;

String^ yearInfo = "";

if (ExpensesCB->Checked == true)

{

yearInfo += "Затраты за год составили: " + salary.ToString() + "\n";

};

if (proceedsCB->Checked == true)

{

yearInfo += "Выручка за год составила: " + proceeds.ToString() + "\n";

};

if (profitCB->Checked == true)

{

yearInfo += "Прибыль за год составила: " + Profit.ToString() + "\n";

};

if (clientsNumberCB->Checked == true)

{

yearInfo += "Количество клиентов за год составило: " + clientsNumber.ToString() + "\n";

};

if (averageСheckCB->Checked == true)

{

yearInfo += "Средний чек за год составил: " + averageСheck.ToString() + "\n";

};

makeReportRTB->Text = yearInfo;

}

if (reportPeriodCB->Text == "Месяц")

{

String^ monthInfo = "";

for each (ServiceOrder \* serviceOrder in db2->getServiceOrdersList())

{

std::string date = serviceOrder->getDate();

std::vector <std::string> elems = split(date, '-');

int buffYear = std::stoi(elems[0]);

int buffMonth = std::stoi(elems[1]);

int buffDay = std::stoi(elems[2]);

if ((buffYear == curYear && buffMonth == curMonth-1 && buffDay >= curDay) ||

(buffYear == curYear && buffMonth == curMonth && buffDay <= curDay))

{

clientsNumber++;

proceeds += serviceOrder->getCost();

}

}

for each (User \* user in db2->getUsersList())

{

if (user->getStatus() == EMPLOYEE)

{

Employee\* buffEmployee = (Employee\*)user;

salary += buffEmployee->getSalary()\*1;

}

}

Profit = proceeds - salary;

averageСheck = proceeds / clientsNumber;

if (ExpensesCB->Checked == true)

{

monthInfo += "Затраты за месяц составили: " + salary.ToString() + "\n";

};

if (proceedsCB->Checked == true)

{

monthInfo += "Выручка за месяц составила: " + proceeds.ToString() + "\n";

};

if (profitCB->Checked == true)

{

monthInfo += "Прибыль за месяц составила: " + Profit.ToString() + "\n";

};

if (clientsNumberCB->Checked == true)

{

monthInfo += "Количество клиентов за месяц составило: " + clientsNumber.ToString() + "\n";

};

if (averageСheckCB->Checked == true)

{

monthInfo += "Средний чек за месяц составил: " + averageСheck.ToString() + "\n";

};

makeReportRTB->Text = monthInfo;

}

if (reportPeriodCB->Text == "День")

{

String^ dayInfo = "";

for each (ServiceOrder \* serviceOrder in db2->getServiceOrdersList())

{

std::string date = serviceOrder->getDate();

std::vector <std::string> elems = split(date, '-');

int buffYear = std::stoi(elems[0]);

int buffMonth = std::stoi(elems[1]);

int buffDay = std::stoi(elems[2]);

if (buffDay == curDay)

{

clientsNumber++;

proceeds += serviceOrder->getCost();

}

}

for each (User \* user in db2->getUsersList())

{

if (user->getStatus() == EMPLOYEE)

{

Employee\* buffEmployee = (Employee\*)user;

salary += buffEmployee->getSalary()/12;

}

}

Profit = proceeds - salary;

averageСheck = proceeds / clientsNumber;

if (ExpensesCB->Checked == true)

{

dayInfo += "Затраты за текущий день составили: неизвестно""\n";

};

if (proceedsCB->Checked == true)

{

dayInfo += "Выручка за текущий день составила: " + proceeds.ToString() + "\n";

};

if (profitCB->Checked == true)

{

dayInfo += "Прибыль за текущий день составила: неизвестно" + "\n";

};

if (clientsNumberCB->Checked == true)

{

dayInfo += "Количество клиентов за текущий день составило: " + clientsNumber.ToString() + "\n";

};

if (averageСheckCB->Checked == true)

{

dayInfo += "Средний чек за текущий день составил: " + averageСheck.ToString() + "\n";

};

makeReportRTB->Text = dayInfo;

}

}

-------------------

**mainForm.cpp**

-------------------

//Aphonin, Ankudinov, Timurziev, Dmitriev

#include "mainForm.h"

#include <Windows.h>

#include <math.h>

using namespace bshop; // Название проекта

using namespace System;

using namespace System::Windows::Forms;

static std::map<int, int> indexOfEmployees;

static std::map<int, int> indexOfClients;

//MockDatabase db3;

int curID;

MockDatabase\* db3 = MockDatabase::getInstance();

[STAThreadAttribute]

void main(array<String^>^ args) {

std::setlocale(LC\_ALL, "Russian\_Russia.1251");

Application::EnableVisualStyles();

Application::SetCompatibleTextRenderingDefault(false);

bshop::mainForm form;

Application::Run(% form);

}

System::Void bshop::mainForm::mainForm\_Load(System::Object^ sender, System::EventArgs^ e)

{

userRegistrationPanel->Visible = false;

userRegistrationPanel->Enabled = false;

lgnPanel->Visible = true;

lgnPanel->Enabled = true;

db3->readDatabaseFromFile();

db3->readServicesFromFile();

db3->readServicesOrdersFromFile();

/\*users.push\_back(user1);

users.push\_back(user2);\*/

}

System::Void bshop::mainForm::lgnAcceptBtn\_Click(System::Object^ sender, System::EventArgs^ e)

{

std::hash<std::string> hasher;

std::string inputEmail= msclr::interop::marshal\_as<std::string>(lgnEmailTB->Text);

std::string inputPassword = msclr::interop::marshal\_as<std::string>(lgnPasswordTB->Text);

size\_t passwordHash = hasher(inputPassword);

int userID = db3->findUser(inputEmail, passwordHash);

if (userID !=NOT\_FOUND)

{

adminForm^ adminNewForm;

clientForm^ clientNewForm;

employeeForm^ employeeNewForm;

int userStatus = db3->getUser(userID)->getStatus();

switch (userStatus) {

case ADMIN:

adminNewForm = gcnew adminForm(userID, ADMIN);

adminNewForm->Show();

this->Hide();

/\*curID = userID;

adminPanel->Visible = true;\*/

break;

case CLIENT:

clientNewForm = gcnew clientForm(userID, CLIENT);

clientNewForm->Show();

this->Hide();

/\*curID = userID;

clientPanel->Visible = true;\*/

break;

case EMPLOYEE:

employeeNewForm = gcnew employeeForm(userID, EMPLOYEE);

employeeNewForm->Show();

this->Hide();

/\*curID = userID;

employeePanel->Visible = true;\*/

break;

default:

System::Windows::Forms::MessageBox::Show("Пользователь имеет неизвестный статус");

break;

}

}

}

System::Void bshop::mainForm::ragBtn\_Click(System::Object^ sender, System::EventArgs^ e)

{

lgnPanel->Visible = false;

lgnPanel->Enabled = false;

userRegistrationPanel->Visible = true;

userRegistrationPanel->Enabled = true;

}

System::Void bshop::mainForm::addUserFormBtn\_Click(System::Object^ sender, System::EventArgs^ e)

{

std::string name = msclr::interop::marshal\_as<std::string>(addUserNameTB->Text);

bool sex;

if (addUserMaleRB->Checked)

{

sex = true;

}

else

{

sex = false;

}

double experience = NULL;

std::string personalAchievements = "";

std::string date = msclr::interop::marshal\_as<std::string>(addUserDateTB->Text);

std::string phonenumber = msclr::interop::marshal\_as<std::string>(addUserPhonenumberTB->Text);

std::string email = msclr::interop::marshal\_as<std::string>(addUserMailTB->Text);

std::string password = msclr::interop::marshal\_as<std::string>(addUserPasswordTB->Text);

//std::string statusString = msclr::interop::marshal\_as<std::string>(addUserTypeCB->Text);

if (name == "" || email == "" || password == "")

{

MessageBox::Show("Есть пустые поля!", "Ошибка!", MessageBoxButtons::OK, MessageBoxIcon::Error);

return;

}

userType status = CLIENT;

if (db3->emailIsUnique(email))

{

db3->addUser(name, sex, date, phonenumber, email, password, status);

db3->writeDatabaseToFile();

}

else System::Windows::Forms::MessageBox::Show("Пользователь с таким e-mail уже существует");

lgnPanel->Visible = true;

lgnPanel->Enabled = true;

userRegistrationPanel->Visible = false;

userRegistrationPanel->Enabled = false;

}

System::Void bshop::mainForm::mainForm\_FormClosing(System::Object^ sender, System::Windows::Forms::FormClosingEventArgs^ e)

{

Application::Exit();

}

-------------------

**employeeForm.cpp**

-------------------

//Aphonin, Ankudinov, Timurziev, Dmitriev

#include "employeeForm.h"

MockDatabase\* DBEmployeeForm = MockDatabase::getInstance();

Employee\* currentEmployeeEmployeeForm;

std::list<ServiceOrder\*> currentOrdersEmployeeForm;

System::Void bshop::employeeForm::employeeForm\_Load(System::Object^ sender, System::EventArgs^ e)

{

currentEmployeeEmployeeForm = (Employee\*)DBEmployeeForm->getUser(this->ID);

employeeCalendar->SetDate(employeeCalendar->TodayDate);

updateTimetable(employeeCalendar, selectedDateEmployeeTB);

updateOrdersGrid(currentOrdersEmployeeForm);

orderEmployeeIdCB->Items->Clear();

orderEmployeeIdCB->Items->Clear();

nameEmployeeTB->Text = msclr::interop::marshal\_as<System::String^>(currentEmployeeEmployeeForm->getName());

specialityEmployeeTB->Text = msclr::interop::marshal\_as<System::String^>(currentEmployeeEmployeeForm->getSpeciality());

achievementsEmployeeTB->Text = msclr::interop::marshal\_as<System::String^>(currentEmployeeEmployeeForm->getPersonalAchievements());

for each (int ID in DBEmployeeForm->getOrdersID(currentEmployeeEmployeeForm))

{

orderEmployeeIdCB->Items->Add(ID);

}

}

System::Void bshop::employeeForm::updateOrdersGrid(std::list<ServiceOrder\*> orders)

{

employeeDataGrid->Rows->Clear();

int i = 0;

for each (ServiceOrder \* serviceOrder in orders)

{

std::string ID = std::to\_string(serviceOrder->getID());

std::string data = serviceOrder->getDate();

Service\* selectedService = (Service\*)DBEmployeeForm->getService(serviceOrder->getServiceID());

std::string serviceName = selectedService != nullptr ? selectedService->getName() :

"УСЛУГА УДАЛЕНА, ID:" + std::to\_string(serviceOrder->getServiceID());

Client\* selectedClient = (Client\*)DBEmployeeForm->getUser(serviceOrder->getClientID());

std::string clientName = selectedClient != nullptr ? selectedClient->getName() :

"ПОЛЬЗОВАТЕЛЬ УДАЛЁН, ID:" + std::to\_string(serviceOrder->getClientID());

std::string cost = std::to\_string(serviceOrder->getCost());

std::string time = serviceOrder->getTime();

std::string status = serviceOrder->getStatus().Equals(true) ? "Выпонен" : "Не выполнен";

std::vector<std::string> v = { ID, data, serviceName, clientName, cost, time, status };

employeeDataGrid->Rows->Add();

int k = employeeDataGrid->ColumnCount;

for (int j = 0; j < k; j++)

{

employeeDataGrid->Rows[i]->Cells[j]->Value = msclr::interop::marshal\_as<System::String^>(v[j]);

}

i++;

}

}

System::Void bshop::employeeForm::employeeForm\_FormClosing(System::Object^ sender, System::Windows::Forms::FormClosingEventArgs^ e)

{

Application::Exit();

}

#pragma region Работа с панелью рабочего

System::Void bshop::employeeForm::exitFromEmployeePanelBtn\_Click(System::Object^ sender, System::EventArgs^ e)

{

DBEmployeeForm->writeServicesOrdersToFile();

this->Close();

Application::Exit();

}

System::Void bshop::employeeForm::updateTimetable(MonthCalendar^ employeeCalendar, TextBox^ selectedDateEmployeeTB)

{

std::string time;

std::string date;

date = std::to\_string((employeeCalendar->SelectionStart).Year) + "-" +

std::to\_string((employeeCalendar->SelectionStart).Month) + "-" +

std::to\_string((employeeCalendar->SelectionStart).Day);

selectedDateEmployeeTB->Text = "";

selectedDateEmployeeTB->Text = msclr::interop::marshal\_as<System::String^>(date);

std::list<ServiceOrder\*> employeeServicesOrders;

std::list<int> employeeServicesOrdersID = DBEmployeeForm->getOrdersID(currentEmployeeEmployeeForm);

for each (int ID in employeeServicesOrdersID)

{

ServiceOrder\* buffServiceOrder = DBEmployeeForm->getServiceOrder(ID);

if (buffServiceOrder->getDate() == date)

{

employeeServicesOrders.push\_back(buffServiceOrder);

}

}

currentOrdersEmployeeForm = employeeServicesOrders;

}

System::Void bshop::employeeForm::employeeCalendar\_DateSelected(System::Object^ sender, System::Windows::Forms::DateRangeEventArgs^ e)

{

updateTimetable(employeeCalendar, selectedDateEmployeeTB);

}

System::Void bshop::employeeForm::orderEmployeeIdCB\_SelectionChangeCommitted(System::Object^ sender, System::EventArgs^ e)

{

ServiceOrder\* buffOrder = DBEmployeeForm->getServiceOrder((int)(orderEmployeeIdCB->SelectedItem));

orderEmployeeCheckCB->Checked = buffOrder->getStatus();

}

System::Void bshop::employeeForm::orderEmployeeCheckCB\_CheckedChanged(System::Object^ sender, System::EventArgs^ e)

{

ServiceOrder\* buffOrder = DBEmployeeForm->getServiceOrder((int)(orderEmployeeIdCB->SelectedItem));

buffOrder->setStatus(orderEmployeeCheckCB->Checked);

}

System::Void bshop::employeeForm::updateOrdersGridBtn\_Click(System::Object^ sender, System::EventArgs^ e)

{

updateOrdersGrid(currentOrdersEmployeeForm);

}

System::Void bshop::employeeForm::employeeCalendar\_DateChanged(System::Object^ sender, System::Windows::Forms::DateRangeEventArgs^ e)

{

/\*std::string time;

std::string date;

date = std::to\_string((Calendar->SelectionStart).Year) + "-" +

std::to\_string((Calendar->SelectionStart).Month) + "-" +

std::to\_string((Calendar->SelectionStart).Day);

selectedDateEmployeeTB->Text = "";

selectedDateEmployeeTB->Text = msclr::interop::marshal\_as<System::String^>(date);

std::list<ServiceOrder\*> employeeServicesOrders;

std::list<int> employeeServicesOrdersID = DBEmployeeForm->getOrdersID(currentEmployeeEmployeeForm);

for each (int ID in employeeServicesOrdersID)

{

ServiceOrder\* buffServiceOrder = DBEmployeeForm->getServiceOrder(ID);

if (buffServiceOrder->getDate() == date)

{

employeeServicesOrders.push\_back(buffServiceOrder);

}

}

currentOrdersEmployeeForm = employeeServicesOrders;\*/

}

#pragma endregion

-------------------

**Employee.cpp**

-------------------

#include "Employee.h"

std::string Employee::getInfo()

{

std::string info = "{" + std::to\_string(getID()) + ";" + getName() + ";" + getSex() + ";" + getDate() + ";" + getPhonenumber() + ";" +

getEmail() + ";" + std::to\_string(getPassword()) + ";" + getStatusString() + ";" + getSpeciality() + ";" +

std::to\_string(getExperience()) + ";" + getPersonalAchievements() + ";" + std::to\_string(getSalary()) + ";" +

std::to\_string(getWorkingHours()) + "}" + "\n";

return info;

}

void Employee::setExperience(double experience)

{

this->experience = experience;

}

void Employee::setSpeciality(userSpeciality speciality)

{

this->speciality = speciality;

}

std::string Employee::getSpeciality()

{

switch (this -> speciality) {

case BARBER:

return "barber";

break;

case ADMINISTRATOR:

return "administrator";

break;

case CLEANER:

return "cleaner";

break;

case DIRECTOR:

return "director";

break;

default:

return "none";

break;

}

}

void Employee::setPersonalAchievements(std::string personalAchievements)

{

}

double Employee::getExperience()

{

return this->experience;

}

std::string Employee::getPersonalAchievements()

{

return this->personalAchievements;

}

double Employee::getSalary()

{

return this->salary;

}

double Employee::getWorkingHours()

{

return this->workingHours;

}

-------------------

**Руководство пользователя.**

1. При запуске программы открывается окно авторизации (рис. 1).
2. Пользователь может ввести логин и пароль, если он уже зарегистрирован в системе, или зарегистрироваться в системе, нажав кнопку «Регистрация».

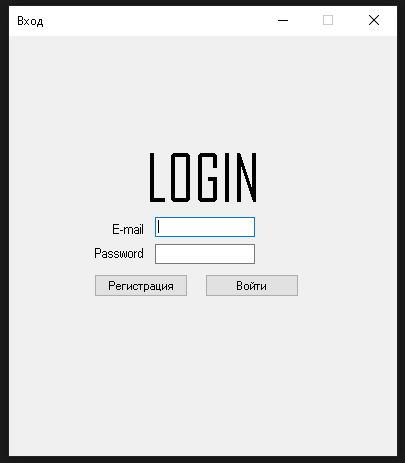


Рис. 1 Панель авторизации.

1. Если пользователь решил зарегистрироваться и нажал кнопку «регистрация», появится окно регистрации (Рис. 2).

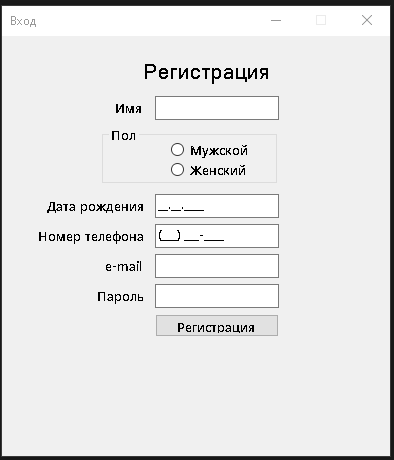


Рис. 2 Форма регистрации

1. Пользователю необходимо ввести данные в соответствующие поля.

(Зарегистрировавшись, пользователь получает статус - клиент)

1. После ввода данных и нажатии кнопки «Регистрация», если пользователь заполнил не все поля, появится соответствующая ошибка (Рис. 3)

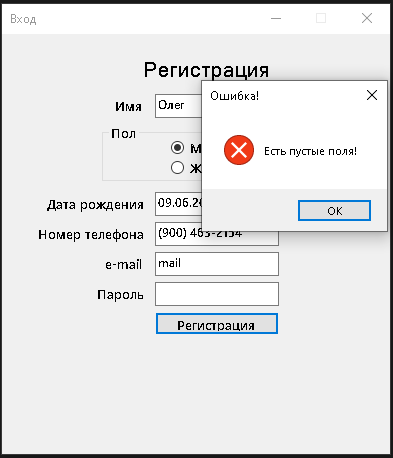


Рис. 3 Сообщение об ошибке при неполном заполнении полей

1. Если пользователь ввел e-mail, а такой e-mail уже существует в системе, то пользователю выдаст соответствующую ошибку.

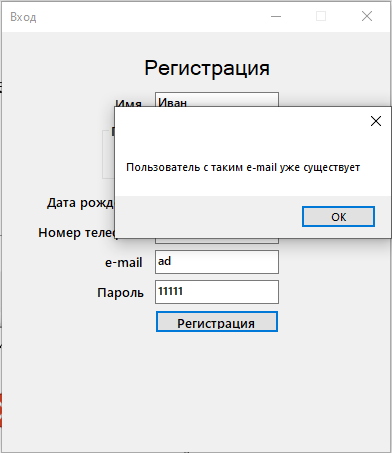


Рис. 4 Сообщение о том, что пользователь с таким e-mail уже существует.

1. При правильном вводе данных и нажатии кнопки «Регистрация» (Рис. 5),   
   пользователь вернется на окно авторизации (Рис. 1).

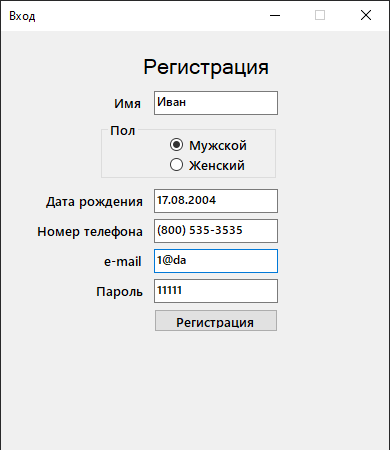


Рис. 5 Регистрация без ошибок

1. При авторизации, если пользователь ввел ошибочные данные: несуществующий e-mail, или неправильный пароль, система выдаст ошибку авторизации (Рис. 6).

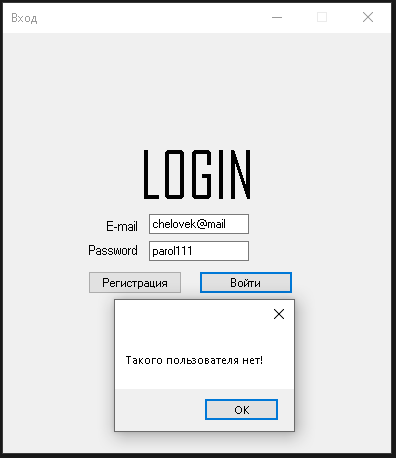


Рис. 6 Ошибка авторизации

1. Если пользователь ввел правильные данные при авторизации, система выдаст сообщение об успешном входе

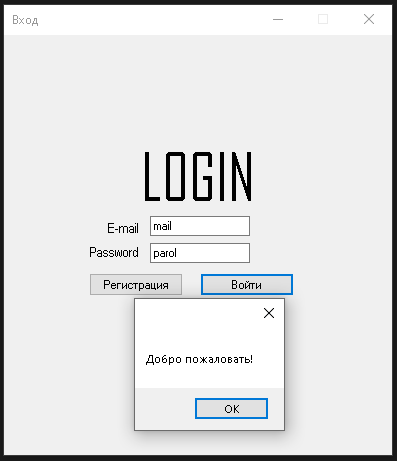


Рисунок 7 Успешная авторизация

**Далее описана инструкция для пользователя типа «клиент».**

1. Войдя в систему, у пользователя типа «клиент» откроется форма клиента (Рис. 8)

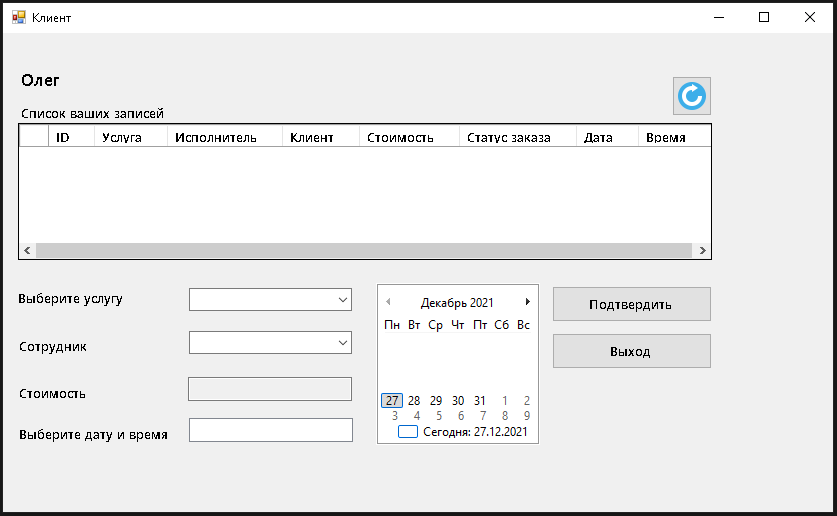


Рис. 8 Окно клиента

1. Посередине сверху клиент может посмотреть свои записи на услуги,

Так же клиент может записаться на услугу.

1. Для записи на услугу клиенту нужно ввести данные в соответствующие поля, выбрать дату на календаре, и выбрать время. Если клиент хочет выбрать день, а он выделен жирным, значит у выбранного барбера все записи заняты.
2. Если клиент заполнил не все поля и нажал кнопку «Подтвердить» система выдаст соответствующее сообщение (Рис. 9)

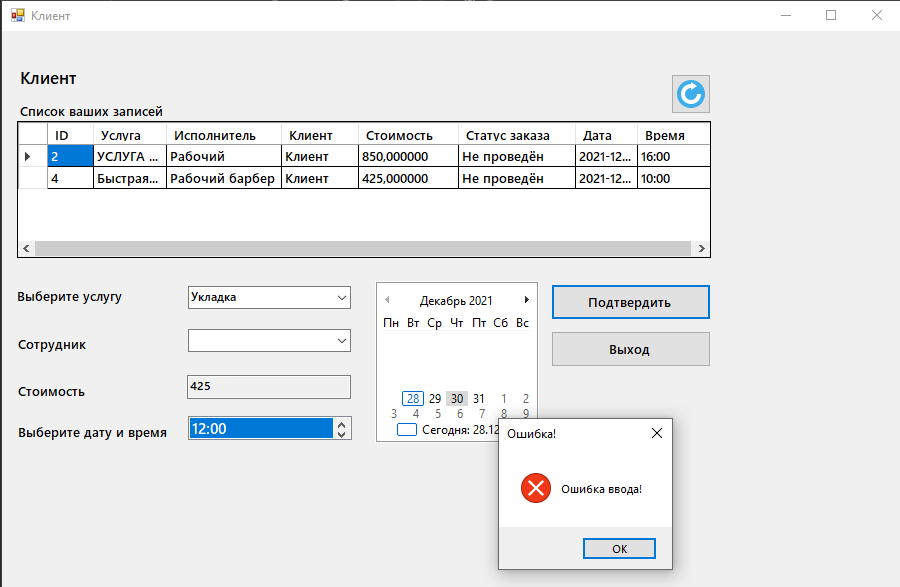


Рис. 9 Ошибка ввода при записи на услугу

1. Если клиент все ввел верно (Рис. 10) и нажал кнопку «Подтвердить», в поле вывода записей клиента появится соответствующая запись на услугу (для вывода записи нужно нажать кнопку «обновление», которая находится справа над списком записей).

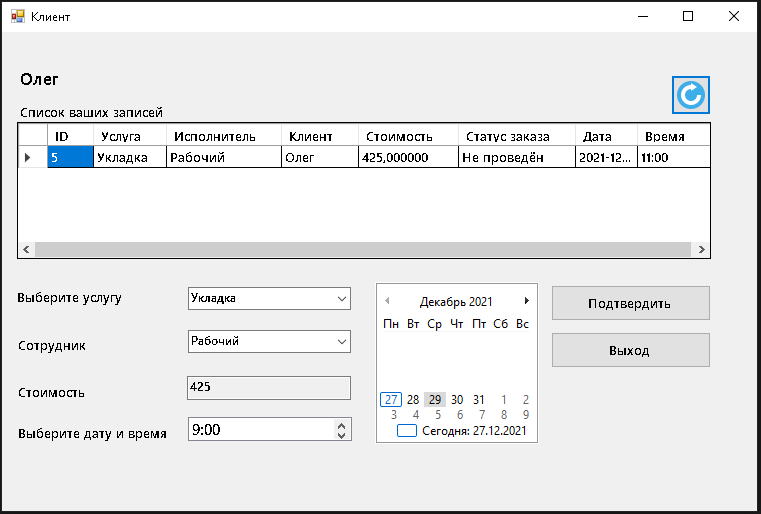


Рис. 10 Пример записи клиента на услугу

**Далее описана инструкция для пользователя типа «админ».**

1. Войдя в систему, у пользователя типа «администратор» откроется форма администратора (Рис. 11)

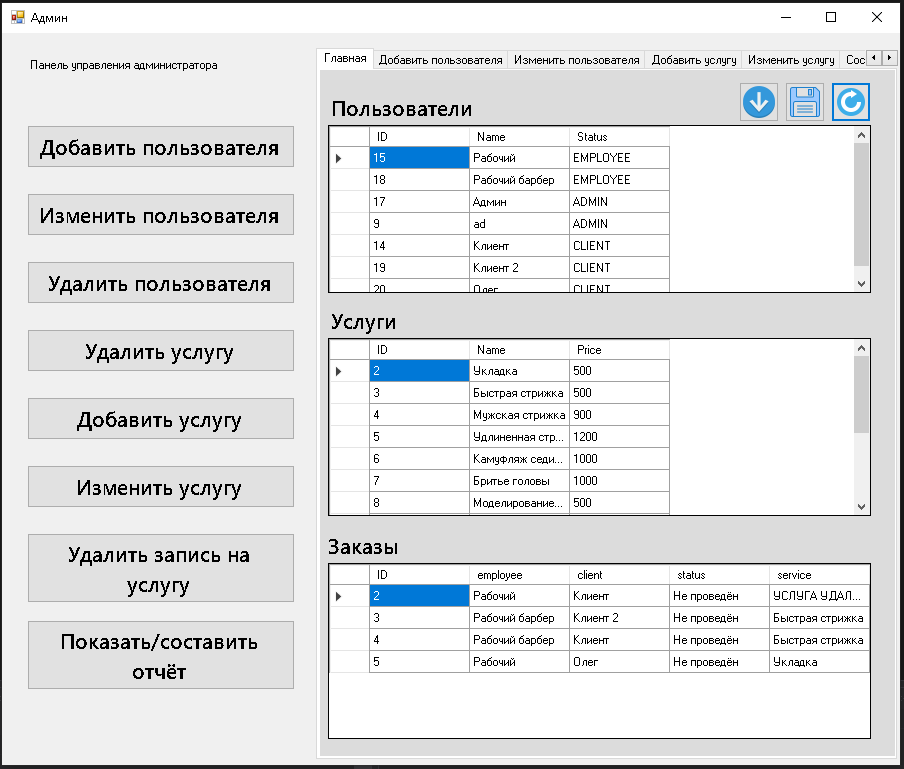


Рис. 11 Форма администратора

1. Слева на форме расположены кнопки: «Добавить пользователя», «Изменить пользователя», «Удалить пользователя», «Добавить услугу», «Изменить услугу», «Удалить услугу». Справа на форме располагаются три таблицы, для вывода списка текущих заказов, услуг и пользователей системы, с соответствующими названиями. Над таблицей «Пользователи» находятся три кнопки: загрузить данные из файла в базу данных, сохранить изменения, обновить данные таблиц.   
   Так же над таблицей «Пользователи» и кнопками работы с данными находится панель вкладок.
2. Если администратор нажал кнопку «Добавить пользователя», откроется вкладка «Добавить пользователя» (Рис. 12).

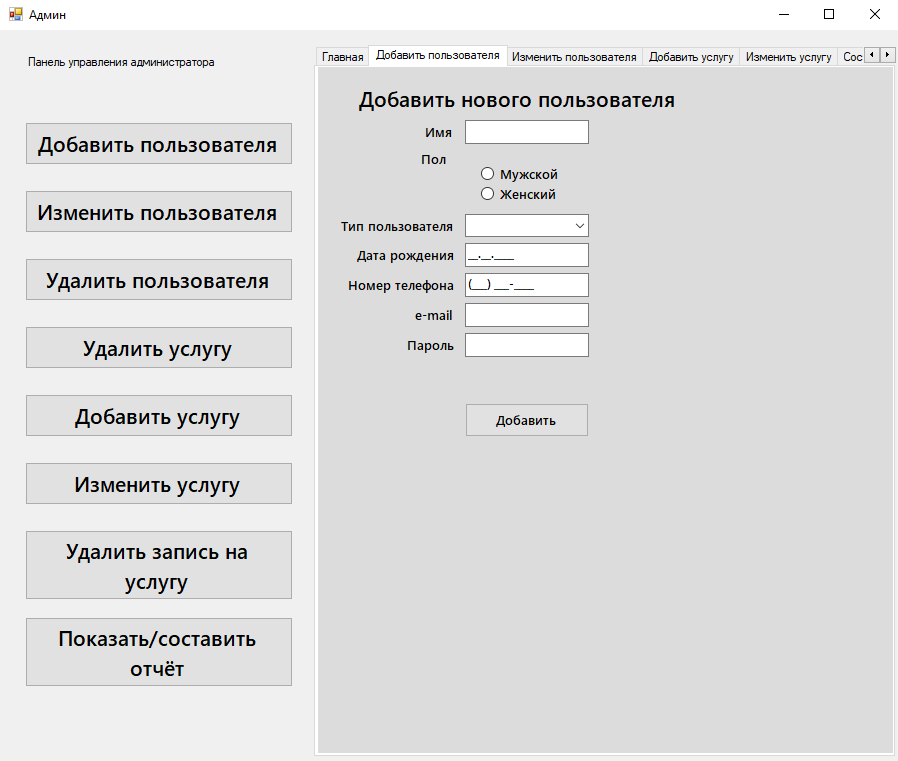


Рис. 12 Добавление нового пользователя

1. Если администратор ввел e-mail и нажал кнопку «Добавить», а такой e-mail уже существует в системе, администратор получит соответствующую ошибку.

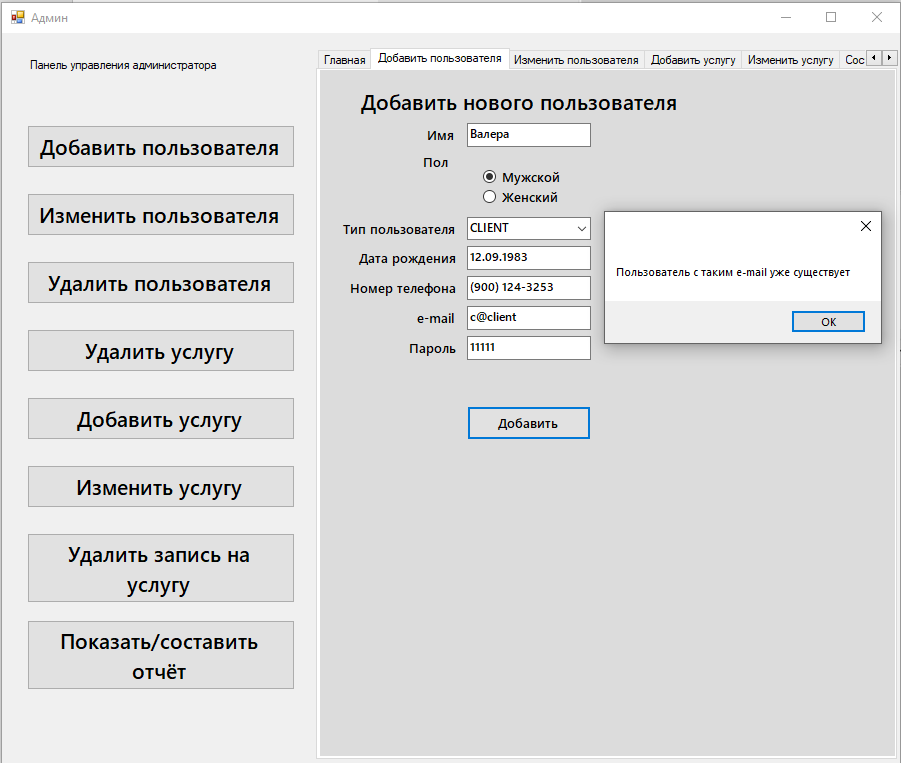


Рис. 13 Ввод существующего e-mail

1. При правильном вводе данных и нажатии кнопки «Добавить» (Рис. 14),   
   пользователь добавляется в базу данных.

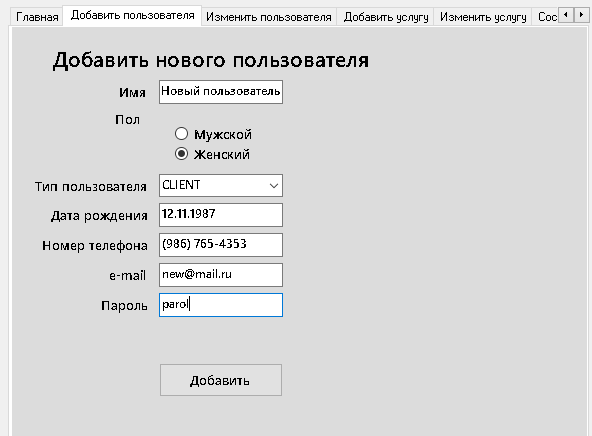


Рис. 14 Добавление нового пользователя без ошибок.

1. Чтобы посмотреть добавленного пользователя, нужно перейти на вкладку «Главная», нажать на кнопку обновления таблиц.

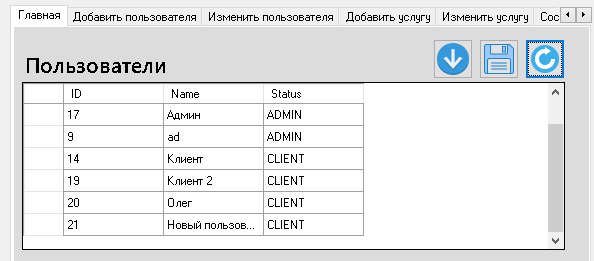


Рис. 15 Вкладка "Главная".

1. Чтобы изменить данные существующего пользователя, нужно выбрать вкладку «Изменить пользователя», откроется панель для редактирования информации о пользователе. Далее нужно изменить нужные поля и нажать кнопку «Изменить».

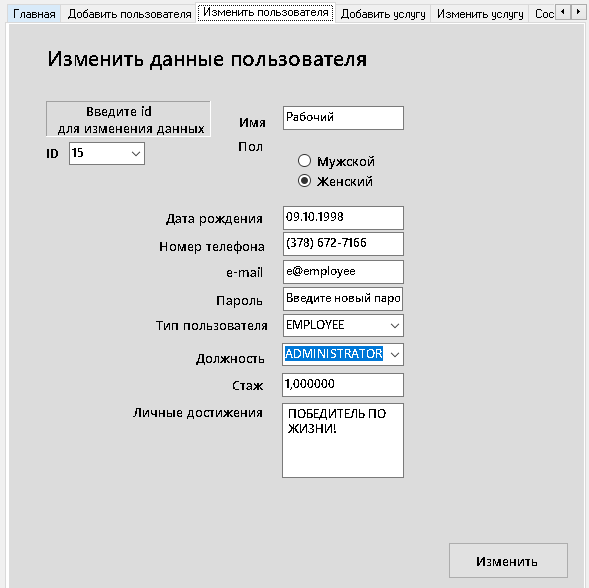


Рис. 16 Панель изменения данных пользователя.

1. Чтобы удалить пользователя, нужно зайти на вкладку «Главная», в таблицы «Пользователи» нужно выбрать пользователя. Выбор пользователя осуществляется нажатием на крайнюю левую ячейку нужного пользователя (Рис. 17).

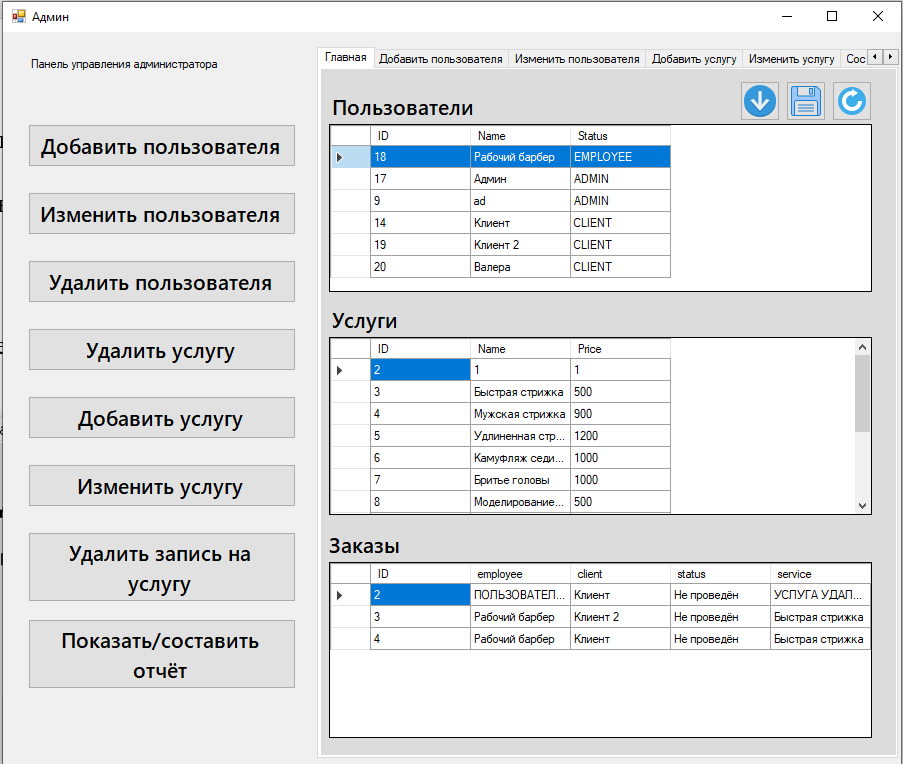


Рис. 17 Удаление пользователя.

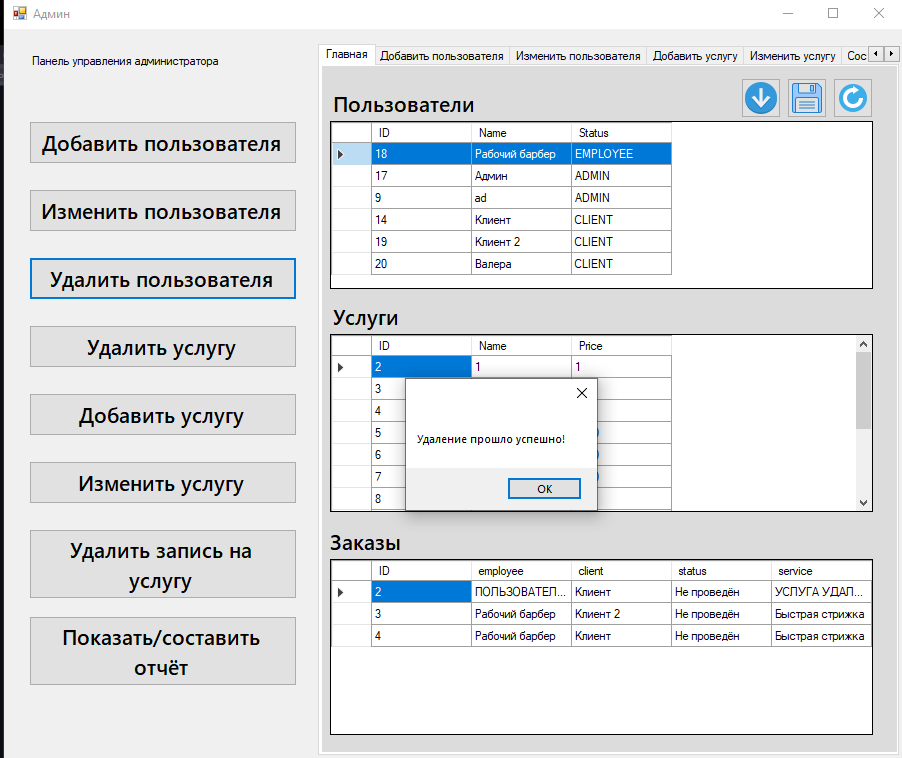
1. Далее нужно нажать кнопку «Удалить пользователя», если удаление прошло успешно, выведится соответствующее сообщение (Рис. 18).

Рис. 18 Успешное удаление.

1. Чтобы посмотреть удалился ли пользователь, нужно нажать на кнопку обновления таблиц (Рис. 19).

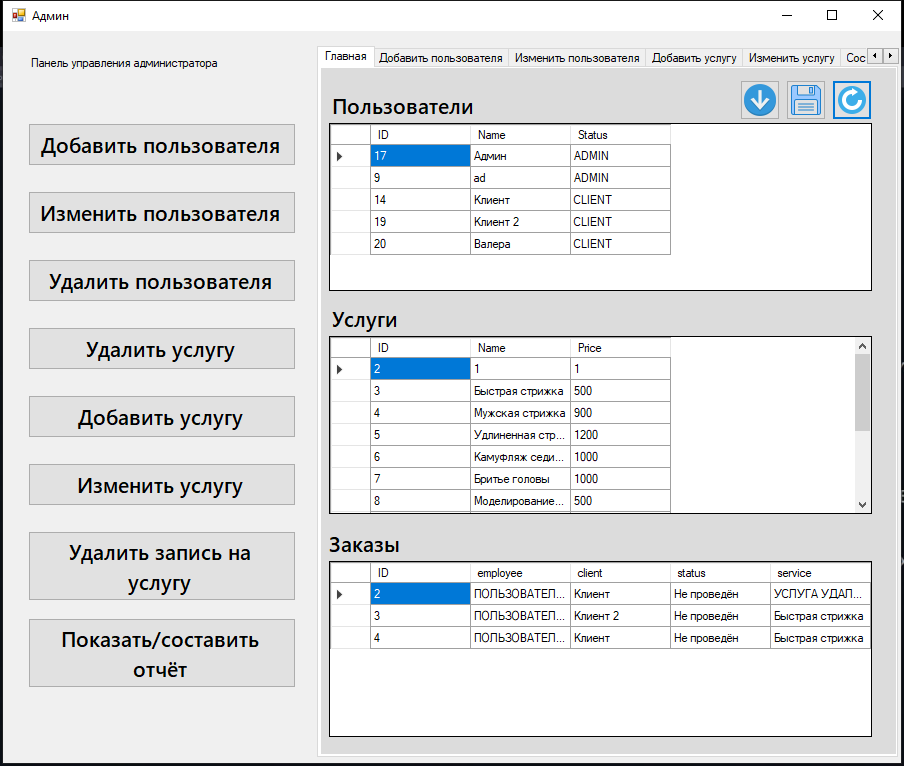


Рис. 19 Просмотр удаленного пользователя.

1. Чтобы удалить услугу, нужно зайти на вкладку «Главная», в таблице «Услуги» нужно выбрать услугу. Выбор услуги осуществляется нажатием на крайнюю левую ячейку нужной услуги (Рис. 20).

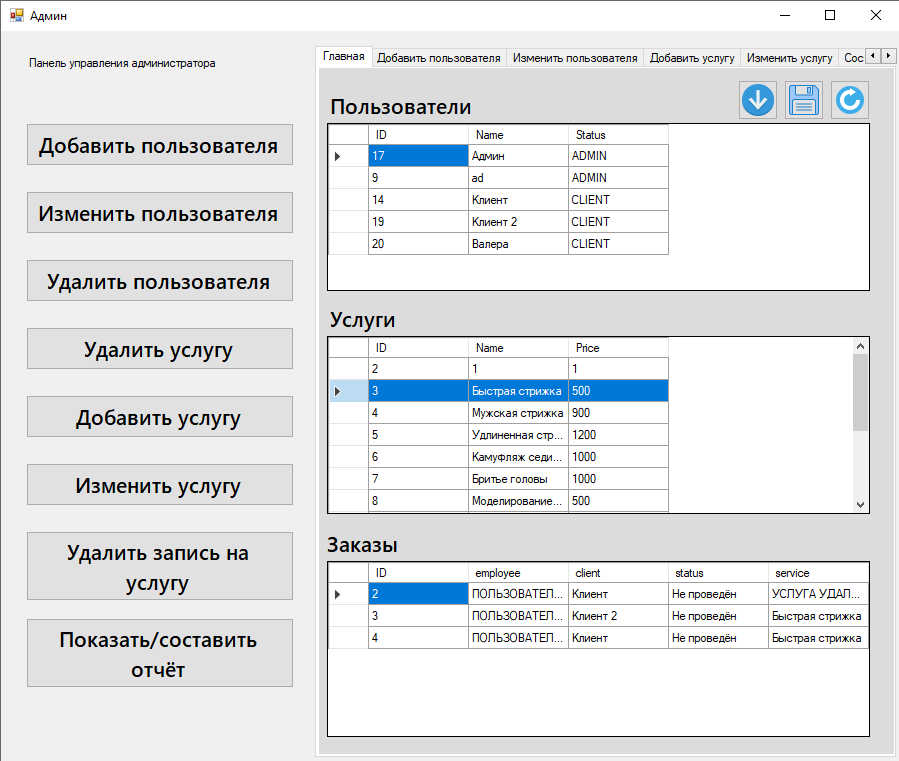


Рис. 20 Удаление пользователя.

1. Далее нужно нажать кнопку «Удалить услугу», если удаление прошло успешно, будет отображено соответствующее сообщение (Рис. 21).

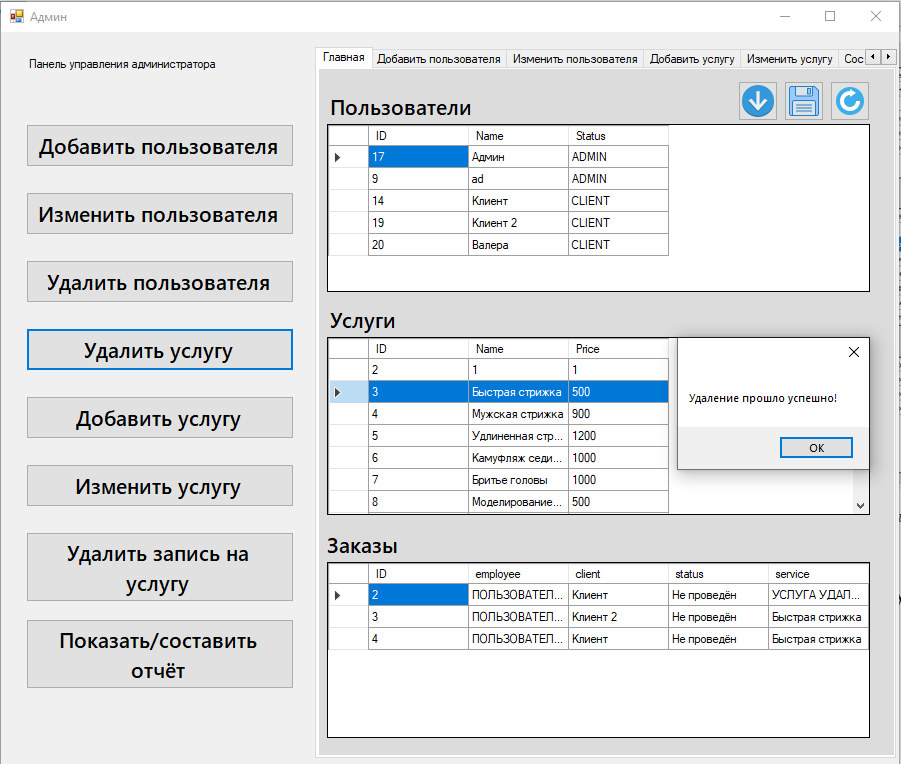


Рис. 21 Успешное удаление.

1. Чтобы посмотреть удалилась ли услуга, нужно нажать на кнопку обновления таблиц (Рис. 22).

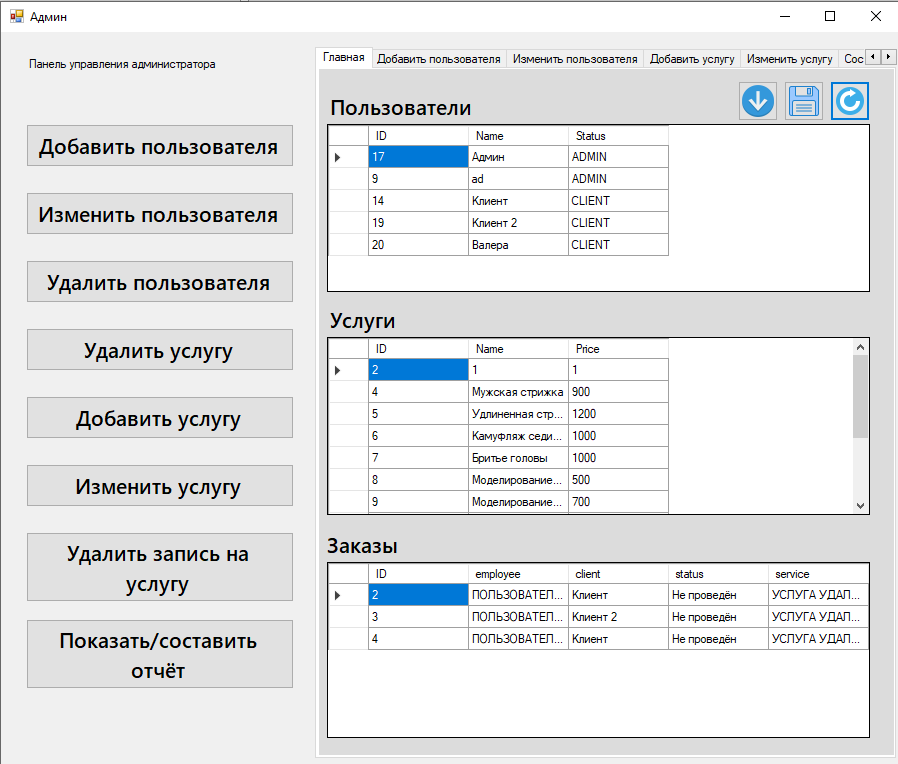


Рис. 22 Просмотр удаленной услуги.

1. Чтобы добавить услугу, нужно нажать на кнопку «Добавить услугу», программа перейдет на вкладку «Добавить услугу», далее нужно ввести название и цену в соответствующие поля и нажать кнопку «Добавить»  
   (Рис. 23). Снизу можно посмотреть существующие услуги.

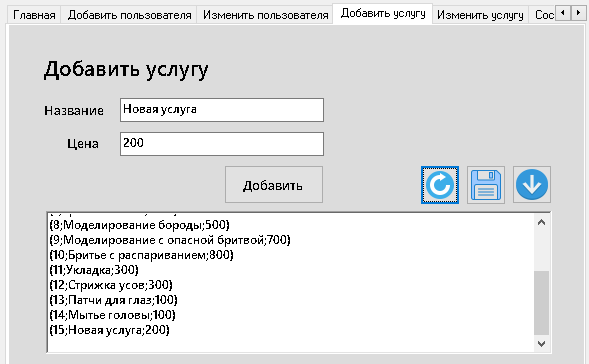


Рис. 23 Добавление услуги.

1. Чтобы изменить услугу, нужно зайти на вкладку «Главная», в таблице «Услуги» нужно выбрать услугу. Выбор услуги осуществляется нажатием на крайнюю левую ячейку нужной услуги (Рис. 24). Далее нажимаем на кнопку «Изменить услугу».

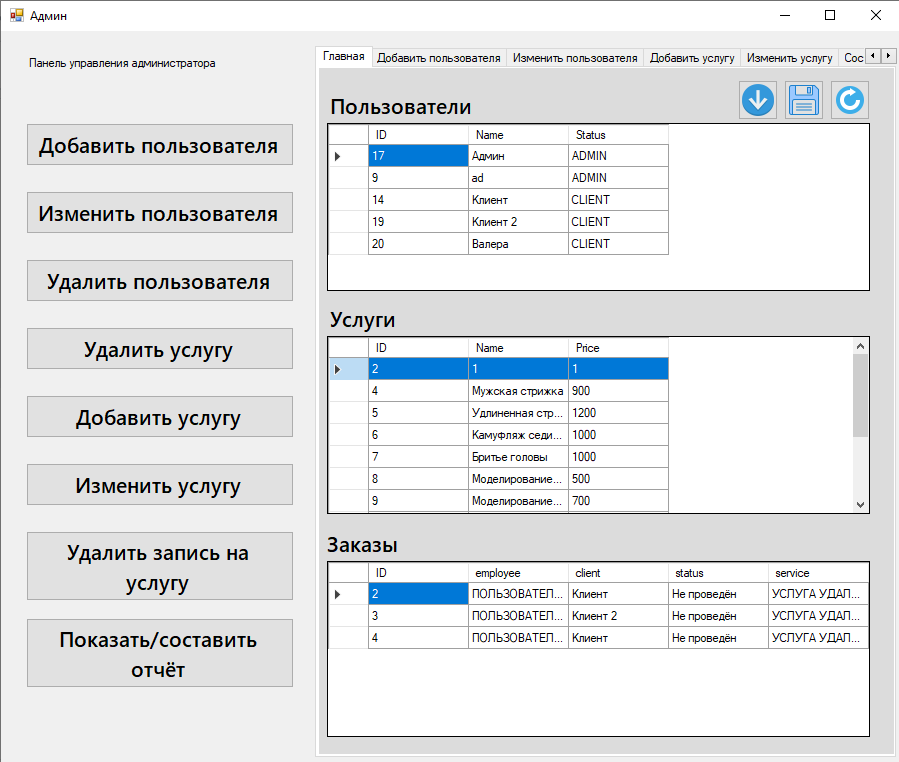


Рис. 24 Выбор услуги для изменения.

1. После нажатия на кнопку «Изменить услугу» откроется вкладка «Изменение услуги» (Рис. 25).

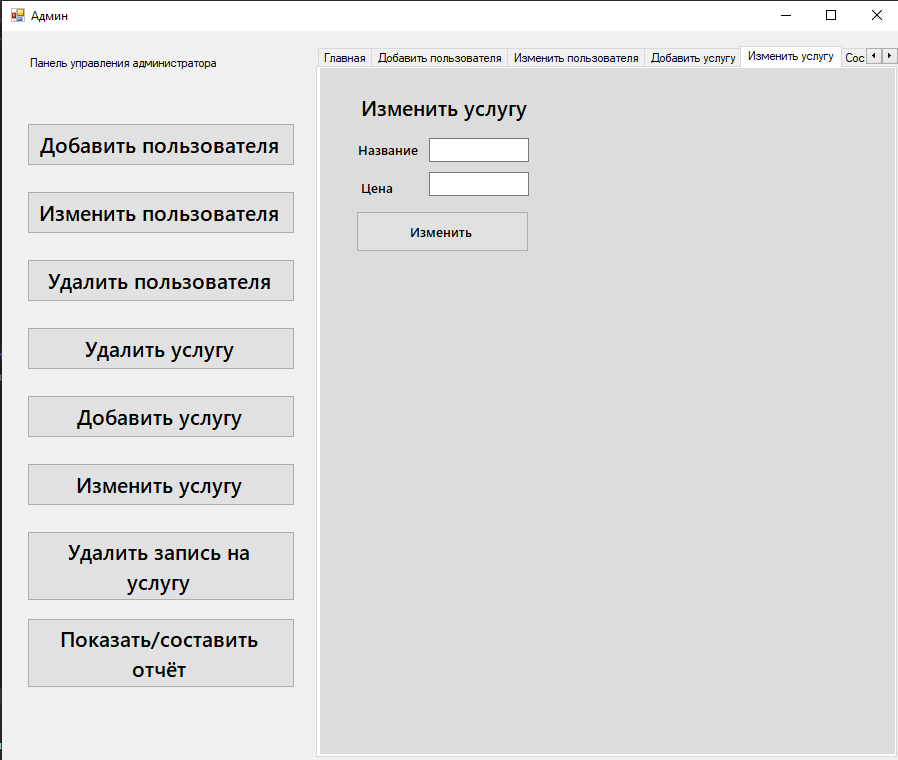


Рис. 25 Вкладка "Изменение услуги".

1. Администратору нужно ввести новые название и цену услуги, после нажать кнопку «Изменить» (Рис. 26).

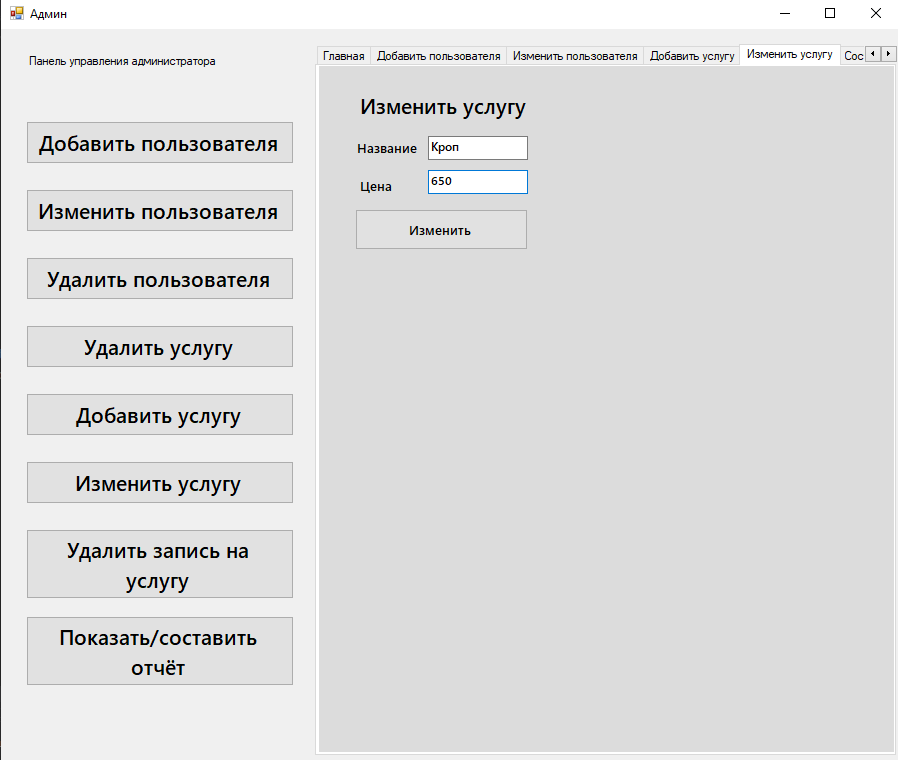


Рис. 26 Изменение услуги.

1. Чтобы удалить запись, нужно зайти на вкладку «Главная», в таблице «Заказы» нужно выбрать Заказ. Выбор заказа осуществляется нажатием на крайнюю левую ячейку нужной услуги (Рис. 27). Далее нажимаем на кнопку «Удалить запись на услугу», программа выведет сообщение об успешном удалении (Рис. 28).

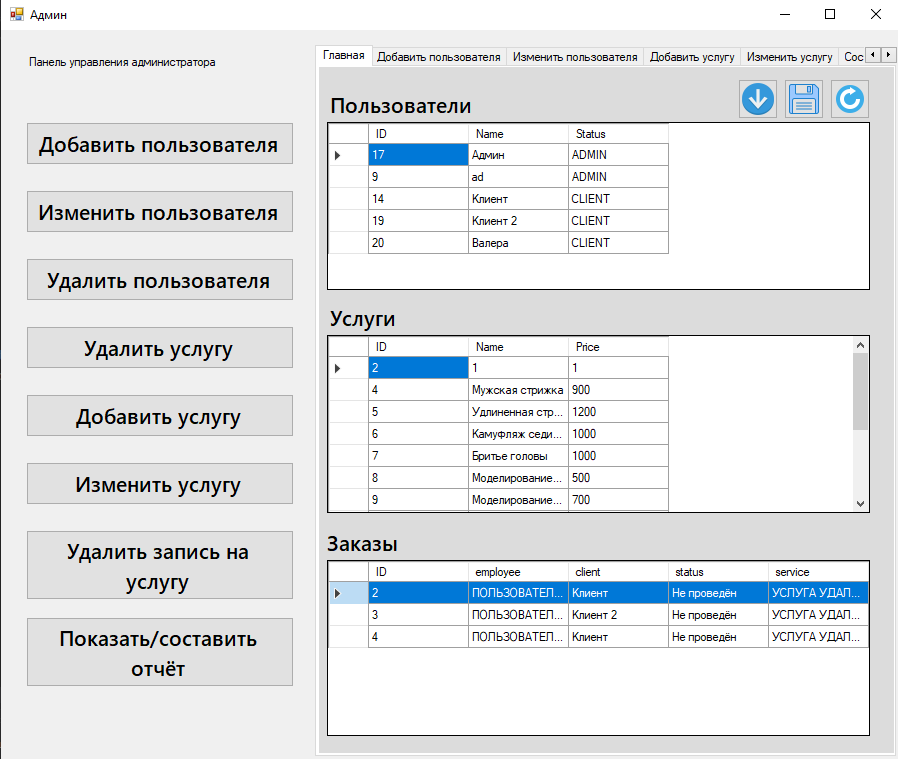


Рис. 27 Удаление записи.

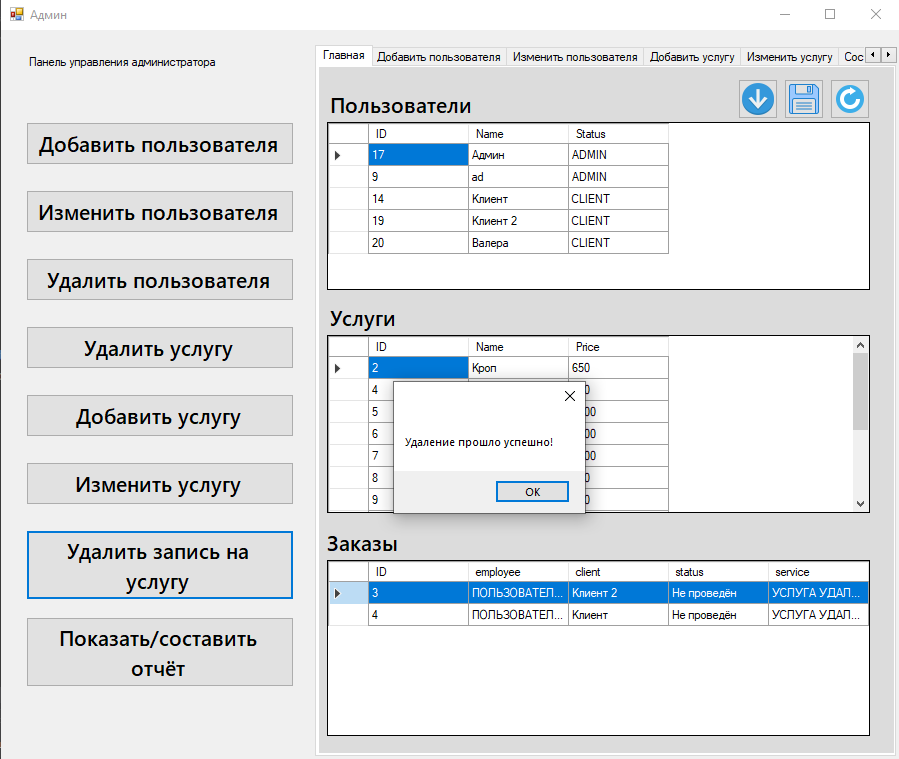


Рис. 28 Успешное удаление.

1. Чтобы составить или просмотреть отчёт, нужно нажать на кнопку «Показать/составить отчёт». Откроется вкладка «Составить отчет»   
   (Рис. 29).

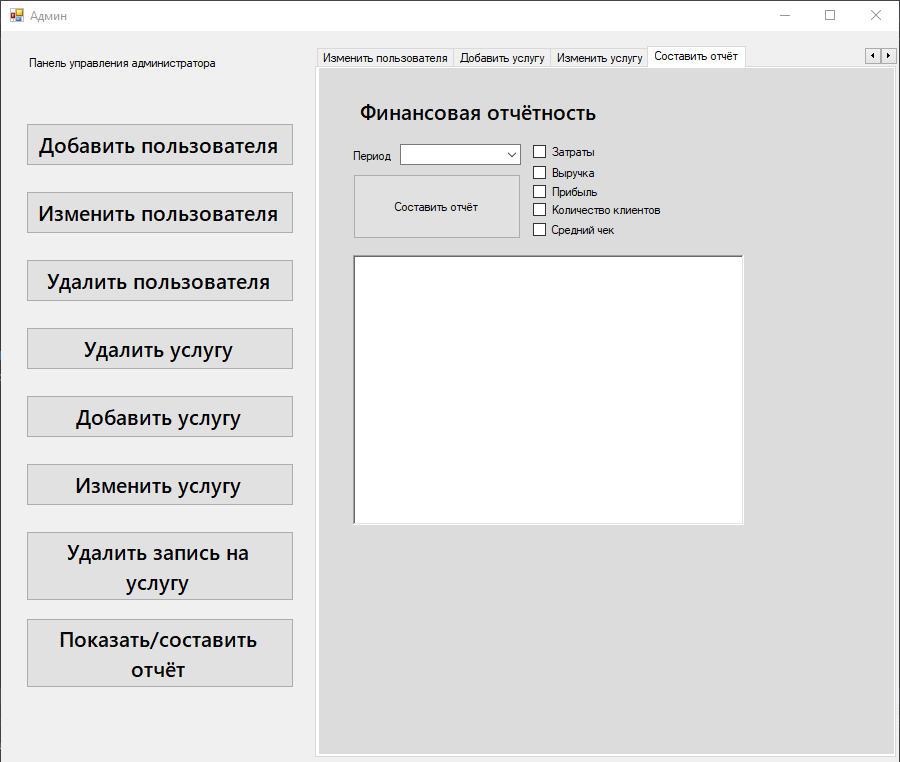


Рис. 29 Составление отчета.

1. Далее можно выбрать интересующий период отчета (день, месяц, год) и параметры, который администратор хочет вывести (Рис. 30).

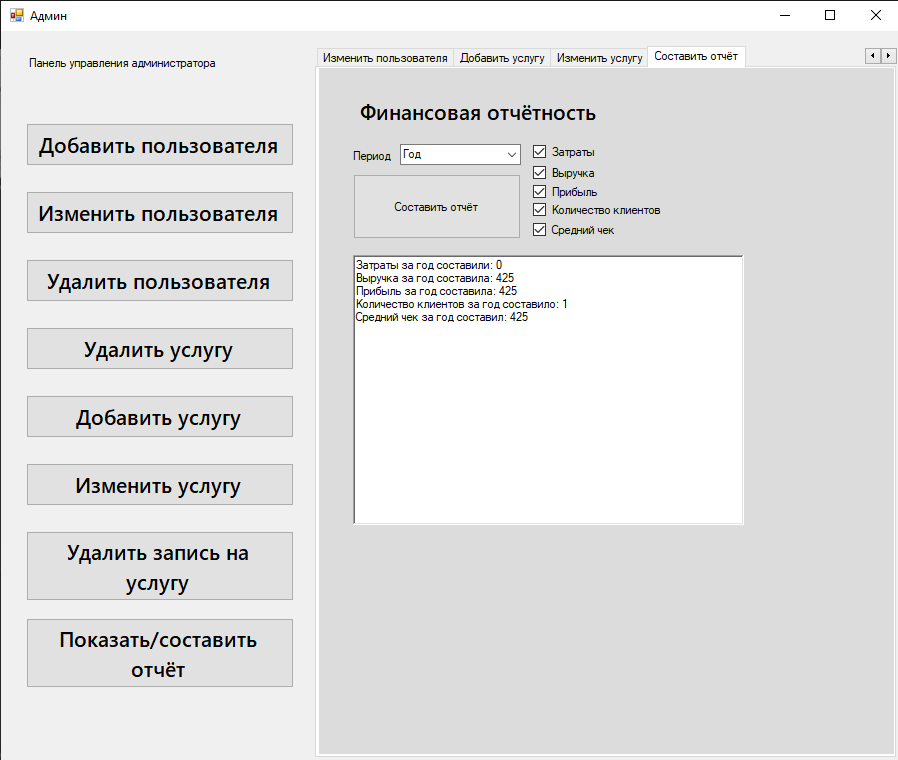


Рис. 30 Вывод отчетности.

**Далее описана инструкция для пользователя типа работник.**

1. Если пользователь зашел в систему как «Работник», открывается форма работника (Рис. 31). В верхней части форму находится таблица, в которой показаны заказы работника за выбранный в календаре день. Так же на форме показаны – имя, должность и личные достижения работника.

На этой форме работник, указывает заказы, которые он выполнил.(Выбрать ID заказа и поставить галочку о его выполнении).

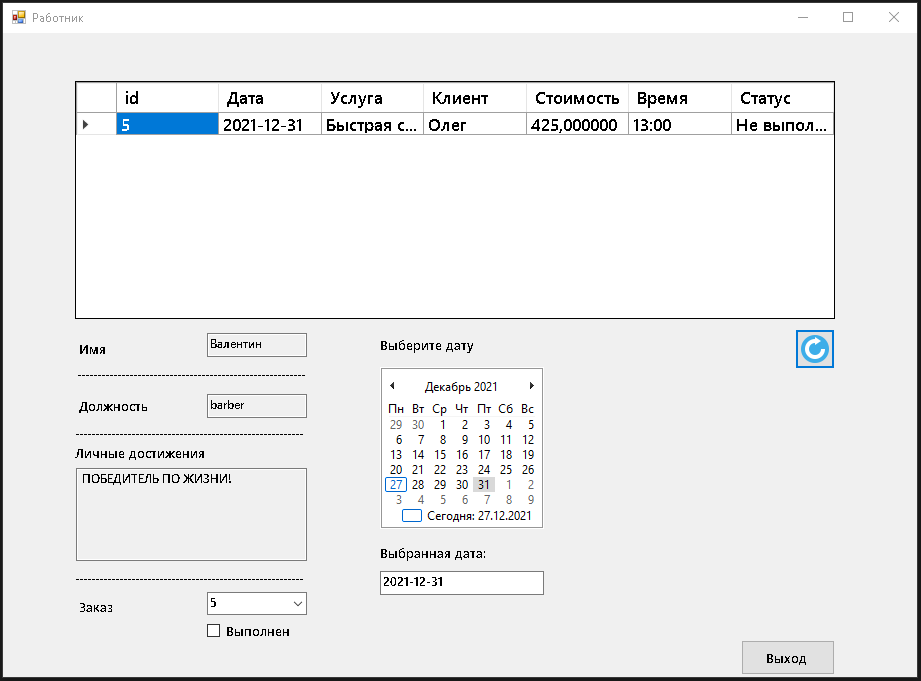


Рис. 31Форма работника.

**История проекта на GitHub**

URL: github.com/Aphonin/barbershop\_project.git



**fix: plural defenition**

Комментарий: Удаление множественного определения

@Aphonin

**Merge branch 'release' into main …**

@Aphonin

**code markup**

@Aphonin

**Merge branch 'release' into main …**

@Aphonin

**Merge branch 'aphonin\_feature' into dev …**

@Aphonin

**add: rename some panels**

Комментарий: Переименование панелей

@Aphonin

**Merge branch 'timurziev\_feature' into dev …**

@Aphonin

**Fix: bug with loading db from file**

Комментарий: Исправление ошибки с загрузкой db из файла

@Timurziev

**Merge branch 'aphonin\_feature' into dev …**

@Aphonin

**fix: err with displey the services in CB, prohibition of viewing orders of other users**

Комментарий: Ошибка с отображением сервисов в комбобоксе.

Запрет просмотра заказов от других пользователей.

@Aphonin

**fix: bugs with password hash**

Комментарий: Ошибка с отображением сервисов в комбобоксе.

Запрет просмотра заказов от других пользователей.

@Aphonin

**Merge branch 'timurziev\_feature' into dev** …

@Aphonin

**add: Employee Form. Fix warnings**

Комментарий: Новый вид рабочей формы. Исправление багов, исправление ошибок типов

@Timurziev

**add: Client Form, Admin Form**

Комментарий: Новый вид формы клиента и формы админа

@Timurziev

**fix: messages in a comment**

Комментарий: Изменение метода setPrice. Скрытие оповещения об успешном

сохранении данных

@Timurziev

**fix: bugs**

Комментарий: Исправление элементов форм

@Timurziev

**fix: otimize forms**

Комментарий: Оптимизация форм.

Исправление имен объектов WinForms.

Удаление лишних объектов.

Удаление избыточных проверок.

@Aphonin

**merge with ankudinov\_feature**

@Aphonin

**merge with dmitriev\_feature**

@Aphonin

**Merge branch 'main' of https://github.com/names1Ended/barbershop\_project…**

@Ankudinov

**add: add panels**

Комментарий: Исправление пользовательских панелей. Группировка панелей по смыслу.

@Ankudinov

**fix: code markup, issues**

Комментарий: Исправление мелких ошибок и улучшение разметки кода.

@Dmitriev

**add: add User, change User**

Комментарий: Разработка методов изменения и добавления пользователей  
@Dmitriev  
  
**Merge branch 'release' into main …**  
@Aphonin  
  
**Merge branch 'dmitriev\_feature' of https://github.com/names1Ended/barb… …**  
@Dmitriev  
  
**add: User setters, complement Employee setters,changeUserBTN**

Комментарий: Разработка действия при нажатии на кнопку changeUserBTN.

Добавление дополнительных сеттеров в классе User и Employee.  
@Dmitriev  
 **add: work with admins panel**

Комментарий: Разработка панели для работы с классом Admin.

Добавление новых методов.  
@Ankudinov  
  
Merge branch 'singleton\_feature' into dev …  
@Aphonin  
  
**fix: singleton errors, add two panels (employee panel, client panel)**

Комментарий: Исправление работы СинглТона. Добавление новых пользовательских панелей. Панель для

работы с Employee и панель для работы с Client

@Aphonin  
 **add: simple singleton**

Комментарий: Добавление метода позволяющего передавать ссылку на объект базы, не создавая его при

каждом вызове. Если при вызове метода такого объекта нет - объект создается  
@Aphonin  
  
**merge dev**  
@Aphonin  
  
**add: methods data read and load about ServiceOrder**

Комментарий: Добавление методов для записи ServiceOrder и для его чтения из файла  
@Dmitriev  
  
**add: new form and users panels, fix data transfer between forms**

Комментарий: Добавление новой WinForm с новыми пользовательскими панелями.

Исправление ошибок передачи данных между формами. Улучшение разметки кода,

группировка кода  
@Aphonin  
  
**Merge remote-tracking branch 'origin/dmitriev\_feature' into dev …**

Комментарий: Проведение изменение в классе Employe и Client  
@Aphonin  
 **add: changes in Employee and Client classes**

Комментарий: Проведение изменение в классе Employe и Client  
@Dmitriev

**add: methods to work with MockDataBase, fix(ServiceOrder and Admin)**

Комментарий: Добавление методов для записи ServiceOrder и для его чтения из файла

@Ankudinov

**Merge branch 'dmitriev\_feature' into dev …**

Комментарий: Добавление новой WinForm с новыми пользовательскими панелями.

Исправление ошибок передачи данных между формами.  
 Улучшение разметки кода, группировка кода

@Aphonin

**add: methods to work with MockDatabase and Service**

Комментарий: Проведение изменение в классе Employe и Client

@Dmitriev

**add: methods to work with MockDatabase and Users**

Комментарий: Исправление ошибок в методах для работы с классами ServiceOrder и Admin

@Aphonin

**add: db sumulator, new funcs for mainForm**

Комментарий: Добавление методов для работы с MockDatabase and Service.  
 Определение и объявление

@Aphonin

**merge: ankudinov\_feature/dmitriev\_feature**

Комментарий: Создание класса MockDatabase.  
 Объявление и опредления методов.  
 Добавление функциональных кнопок в форму.  
 Первые попытки управления эмулятором базы данных.

@Aphonin

**Merge remote-tracking branch 'origin/ankudinov\_feature' into dev …**

Комментарий: Создание классов User и ServiceOrder.  
 Исправление ссылок на файлы в проекте.  
 Объявление и определение их методов.

@Aphonin

**fix: ref to files**

Комментарий: Исправление ссылок на файлы в проекте после слияния

@Ankudinov

**add: ServiceOrder class**

Комментарий: Добавление класса ServiceOrder  
 Объявление и определение методов класса ServiceOrder

@Ankudinov

**add: Admin class**

Комментарий: Добавление класса Admin.  
 Объявление методов класса Admin

@Ankudinov

**add: Client, Employee classes**

Комментарий: Создание класса Client и Employee  
 Объявление и определение их методов.

@Dmitriev

**init project**

Комментарий: Предоставляет основу - скелет для начала разработки программы.

@Aphonin

**Initial commit**

@Aphonin

**Заключение:**

Среда разработки – Microsoft Visual Studio 2019  
  
Язык разработки – C++  
  
В данном проекте применялась система контроля версий Git. У каждого участника проекта имеется более трёх коммитов. Все прецеденты были реализованы. Сбоев и зависаний не наблюдается.  
Использовались принципы раздельной компиляции. Весь проект разбит на модули, состоящие из двух файлов: заголовочного и файла реализации.  
В заголовочных файлах у нас содержатся только объявления. В файлах реализации – определения и объявления. В файле реализации подключена директива включения соответствующего заголовочного файла.  
Неизбыточности алгоритмов или неиспользуемых объектов - не присутствует.  
  
В отчёте приложены диаграммы деятельностей, вариантов использования, классов и диаграмма последовательностей.