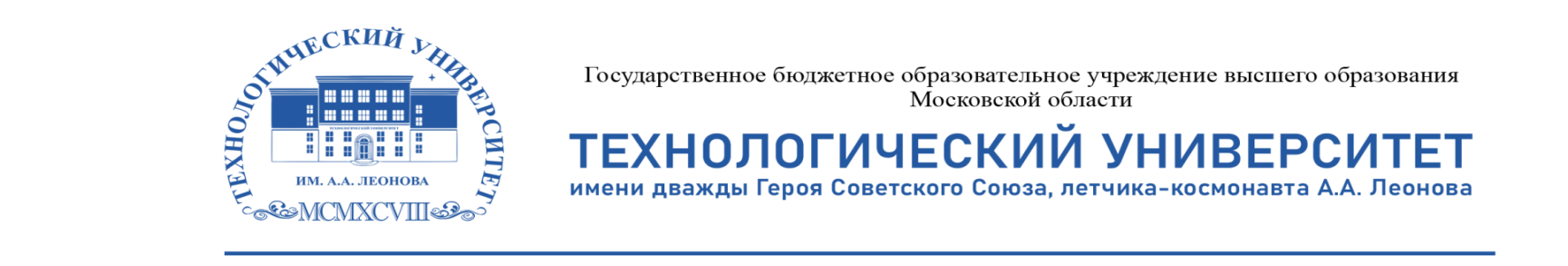
****

Колледж космического машиностроения и технологии

**Курсовой проект**

**Разработка модуля графического интерфейса для настольного приложения «Журнал успеваемости»**

Пояснительная записка

КП.09.02.03.22.18ПЗ

Обучающийся группы П2-19 Ратников В.Н.

Руководитель курсового проекта Гусятинер Л.Б.

Результат защиты\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Королев, 2022 г.

**Оглавление**

[**Введение** 3](#_Toc106713478)

[1. **Теоретическая часть** 4](#_Toc106713479)

[**1.1 Описание предметной области** 4](#_Toc106713480)

[**1.2.** **Описание существующих разработок** 5](#_Toc106713481)

[**1.2.1 «Сотрудники предприятия» от AraxGroup** 5](#_Toc106713482)

[**1.2.2 «Кадры Плюс» от** «**AnDeeSoft**» 7](#_Toc106713483)

[**2. Проектная часть** 11](#_Toc106713484)

[**2.1 Построение диаграммы прецедентов** 11](#_Toc106713485)

[**2.2. Выбор инструментов** 12](#_Toc106713486)

[**2.3. Проектирование сценария** 13](#_Toc106713487)

[**2.4. Построение диаграммы классов** 14](#_Toc106713488)

[**2.5. Описание стартового модуля** 15](#_Toc106713489)

[**2.6. Описание спецификаций к модулям** 17](#_Toc106713490)

[**2.7. Описание модулей** 19](#_Toc106713491)

[**2.7.1 BaseUser.cs; Department.cs; User.cs; UserServices.cs** 19](#_Toc106713492)

[**2.7.2 Модули Razor pages** 21](#_Toc106713493)

[**2.8. Описание тестовых наборов модулей** 29](#_Toc106713494)

[**2.9. Описание применения средств отладки** 31](#_Toc106713495)

[**2.** **Эксплуатационная часть** 32](#_Toc106713496)

[**2.1.** **Руководство оператора** 32](#_Toc106713497)

[**2.1.** **To-Do лист** 36](#_Toc106713498)

[**Заключение** 37](#_Toc106713499)

[**Список использованной литературы и интернет-ресурсов** 38](#_Toc106713500)

[**Приложение 1. Интерфейс IUserService** 39](#_Toc106713501)

[**Приложение 2. Функции Counter.razor** 40](#_Toc106713502)

# **Введение**

Целью данного курсового проекта является написание программы для учета сотрудников производственного предприятия. Тема «Разработка приложения для учета сотрудников производственного предприятия» была выбрана, поскольку любому предприятию необходимо вести учёт сотрудников для удобной классификации и сортировке сотрудников. Данный курсовой проект позволит создавать и редактировать базу данных, в которой хранятся персональные данные каждого сотрудника.

Курсовой проект представляет из себя приложение Blazor Server, который задействует файлы с кодами на языке программирования C# и компонентами Razor pages. В программе задействована база данных SQL.

В первой части (теоретической) будет рассмотрена предметная область данной темы, а также несколько аналогичных продуктов, с их плюсами и минусами.

В второй части (проектной) будет рассмотрен выбор инструментов и сценарий программы. Будет рассмотрен главный модуль, спецификации к модулям, дополнительные модули и применения средств откладки.

В третьей части (эксплуатационной) будет рассмотрено руководство оператора и To-Do лист.

В заключительной части будет представлен список используемой литературы и интернет ресурсов, а также сделаны общие выводы о получившемся проекте.

# 1. **Теоретическая часть**

## **1.1 Описание предметной области**

Предметной областью разрабатываемой программы является автоматизация учета кадров на любом производственном предприятии.

Кадровик (специалист отдела кадров, менеджер по персоналу или инспектор по кадрам) – это сотрудник компании, который занимается ведением кадров и несет ответственность за формирование кадровой политики предприятия.

Слово «кадровик» широко использовалось и впервые было применено ещё в СССР. Сотрудник отдела кадров заполнял трудовые книги, ввёл личные дела сотрудников и записывал должностные инструкции. Работа была рутинной.

Задача кадровика — оформлять и вносить вправки в документы кадровой службы, проверять на соответствие актуальному законодательству.

Сотрудник отдела кадров несёт большую ответственность, именно поэтому в больших компаниях работают несколько специалистов, каждый из которых отвечает за своё подразделение. На каждом предприятии должен работать начальник отдела кадров, который руководит процессом и принимает вопросы о стратегическом развитии компании.

В обязанности начальника отдела кадров входит учёт работников предприятия; работа по комплектованию предприятия кадрами рабочих требуемых профессий; прием, размещение и расстановка новых специалистов; анализ кадровой работы на предприятии и разработка предложений по ее улучшению.

## **1.2. Описание существующих разработок**

В качестве примера будут рассмотрены уже имеющиеся программы для автоматизации работы работников кадрового отдела.

Интерфейс в таких программах должен быть удобным, понятным и многофункциональным, для удовлетворения всех потребностей начальника отдела кадров

Необходимые функции программы:

1. Создание базы данных содержащую список всех сотрудников
2. Добавление, редактирование и удаление сотрудников предприятия
3. Хранение сведений о сотруднике (Номер телефона, Электронная почта и т.п.)

### **1.2.1 «Сотрудники предприятия» от AraxGroup**

**«**Сотрудники предприятия**»** [**[1]**](https://araxgroup.ru/sotrud/ustanovka.htm) — Программа для автоматизации работы кадровых отделов организаций. В этом приложении можно создать базу данных с карточками каждого сотрудника предприятия и грамотно структурировать ее.

В качестве примера был добавлен сотрудник и внесены некоторые персональные данные. На рисунке 1 изображена карточка Ратникова В.Н. Стоит отметить, что программа позволяет хранить большое количество информации о сотруднике, вроде паспортных данных, сведений о образовании и воинском учете, а также трудовую книжку.

Все необходимые поля покрашены зелёным цветом. Всю нужную информацию можно будет узнать в несколько кликов, а также вывести её в Word или Excel файлы.

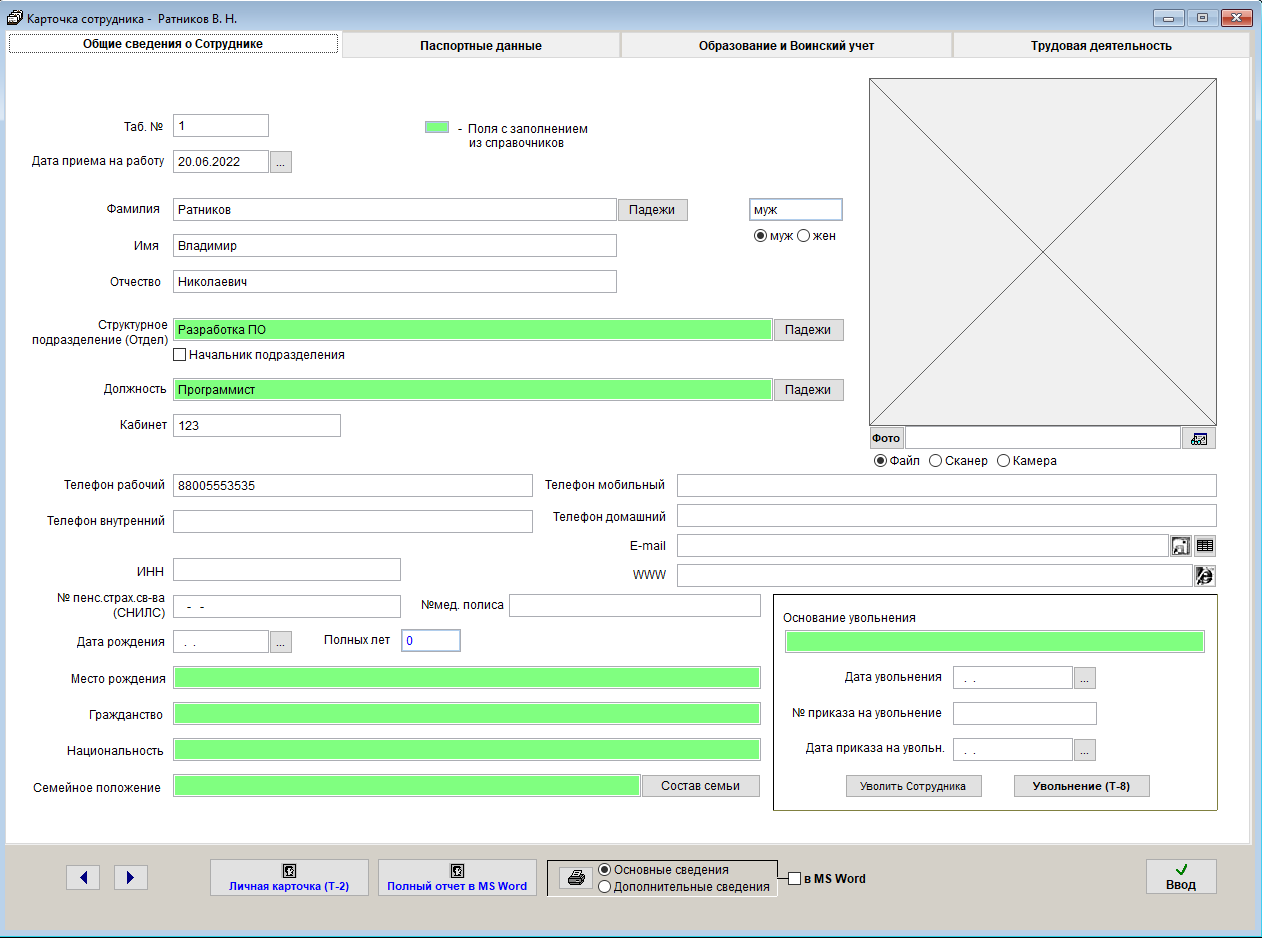


Рисунок 1. Карточка сотрудника

Главные преимущества данной программы:

1. Все сведения о сотрудниках хранятся в одном месте. Это удобно, ведь так можно легко найти нужного сотрудника. На втором рисунке показан Журнал сотрудников.
2. Интуитивный и простой интерфейс, в котором не запутаешься.
3. На каждого работника заведена своя Карточка, содержащая всю информацию о нём. В Карточку также можно загрузить фотографию сотрудника
4. Экспорт информации с Карточки в Word файл.
5. Несколько прав доступа для разных сотрудников и разных целей. Возможна детальная настройка для каждого уровня доступа

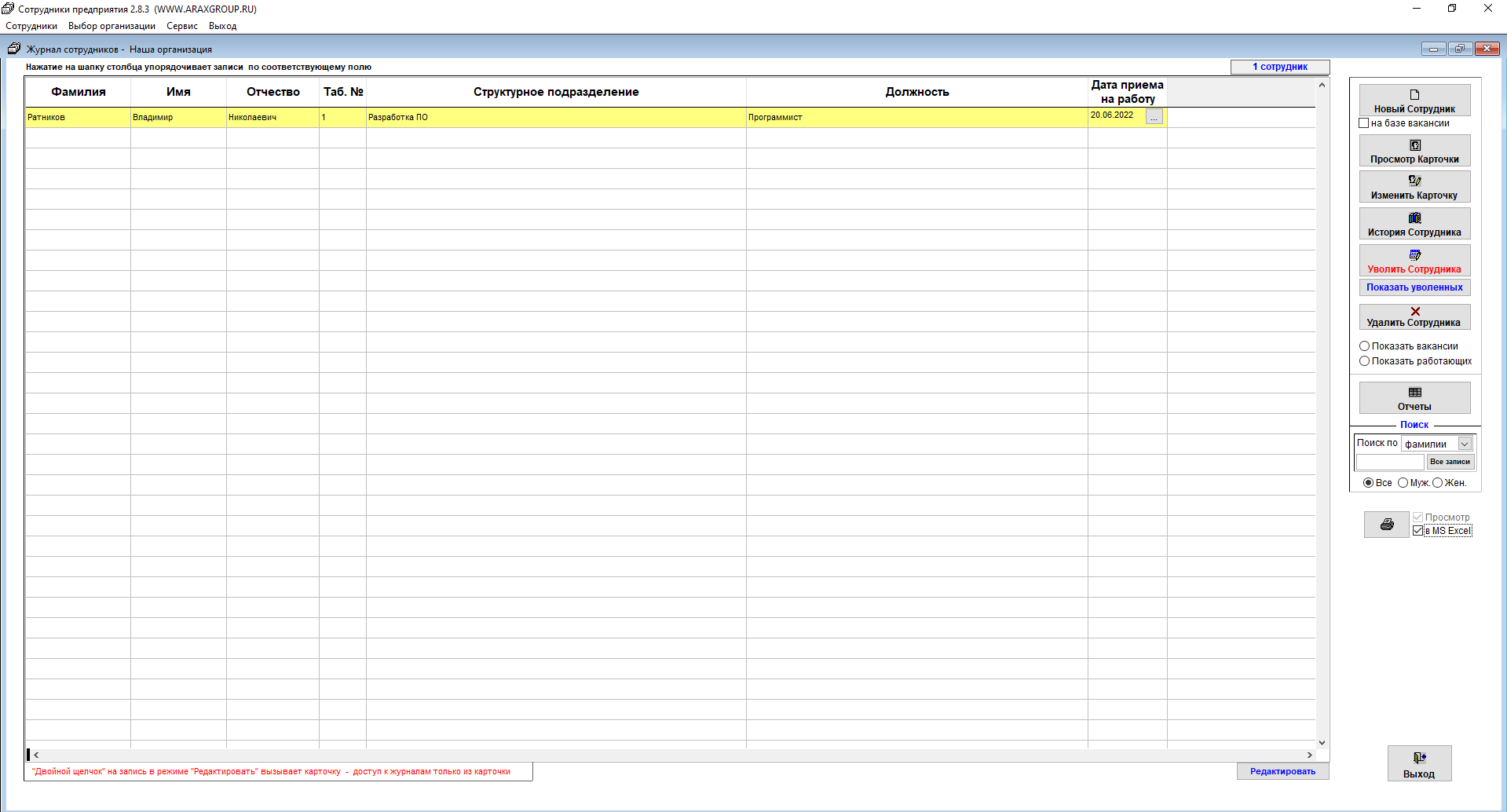


Рисунок 2. Журнал сотрудников

«Сотрудники предприятия» простая и удобная, которая содержит всё необходимое для работника отдела кадров в небольшой компании. Программа содержит множество полей для хранения данных. Присутствует вывод полного отчёта сотрудника в Word файл.

### **1.2.2 «Кадры Плюс» от** «**AnDeeSoft**»

«Кадры Плюс» [**[2]**](https://soft.mydiv.net/win/download-Kadry-Plyus.html) — это программа от компании «AnDeeSoft» для автоматизации кадрового учёта. Она позволяет создавать любые необходимые заявления, приказы и отчёты, а также осуществляет учет рабочего времени сотрудников во время работы.

При установке программы будет предоставлен выбор режима работы (Отображено на Рисунок 3. Выбор режима). Можно выбрать режим сервера, клиента и обычный. При одиночном режиме IP-адрес не нужный, он необходим только в режиме сервера и клиента. Выбираем одиночный и жмём Далее. Выбираем Табель сотрудников и сохраняем.

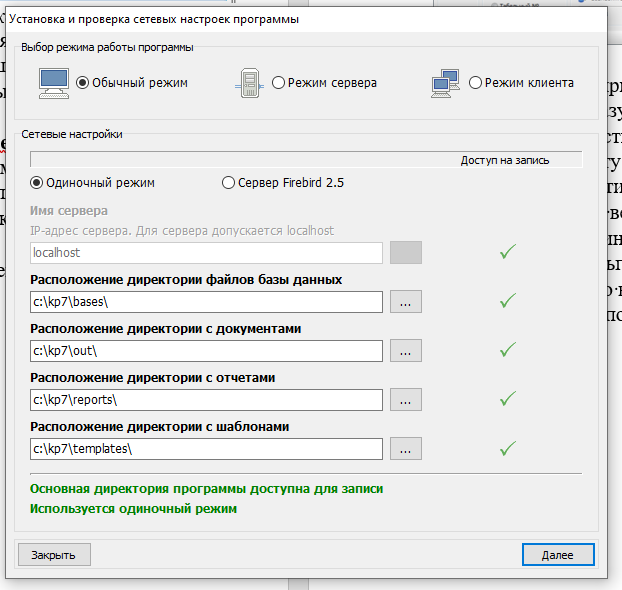


Рисунок 3. Выбор режима

Примечательно, что для базы данных нужно установить пароль и уже можно приступать к работе. Далее можно установить Общие сведения, Место расположения и другие параметры.

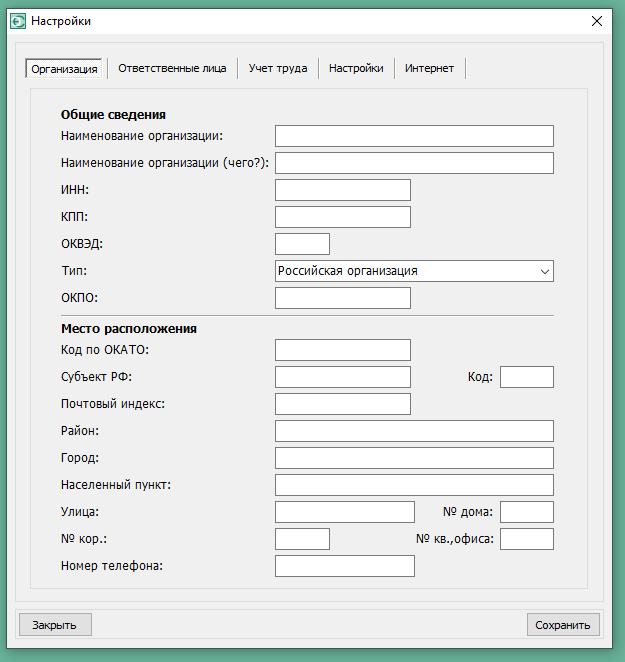


Рисунок 4. Настройка

После настройки, открываем список сотрудников. Видим пустой список. Открываем структуру организации и для тестовой группы добавляем должность программист. Возвращаемся к списку сотрудников и добавляем тестового пользователя. На Рисунок 5 отображена личная карточка сотрудника.

Интерфейс программы сложен и неудобен, но зато присутствует фильтрация и поиск сотрудников. «Кадры Плюс» годиться для работы в крупных компаниях с огромным количеством сотрудников и отделов.

Главные особенности программы:

1. Возможность ведение нескольких организаций
2. Подробная информация о сотруднике: Стаж, дети сотрудника, социальные льготы
3. Учет рабочего времени персонала
4. Экспорт и импорт данных в Word и Excel файлы

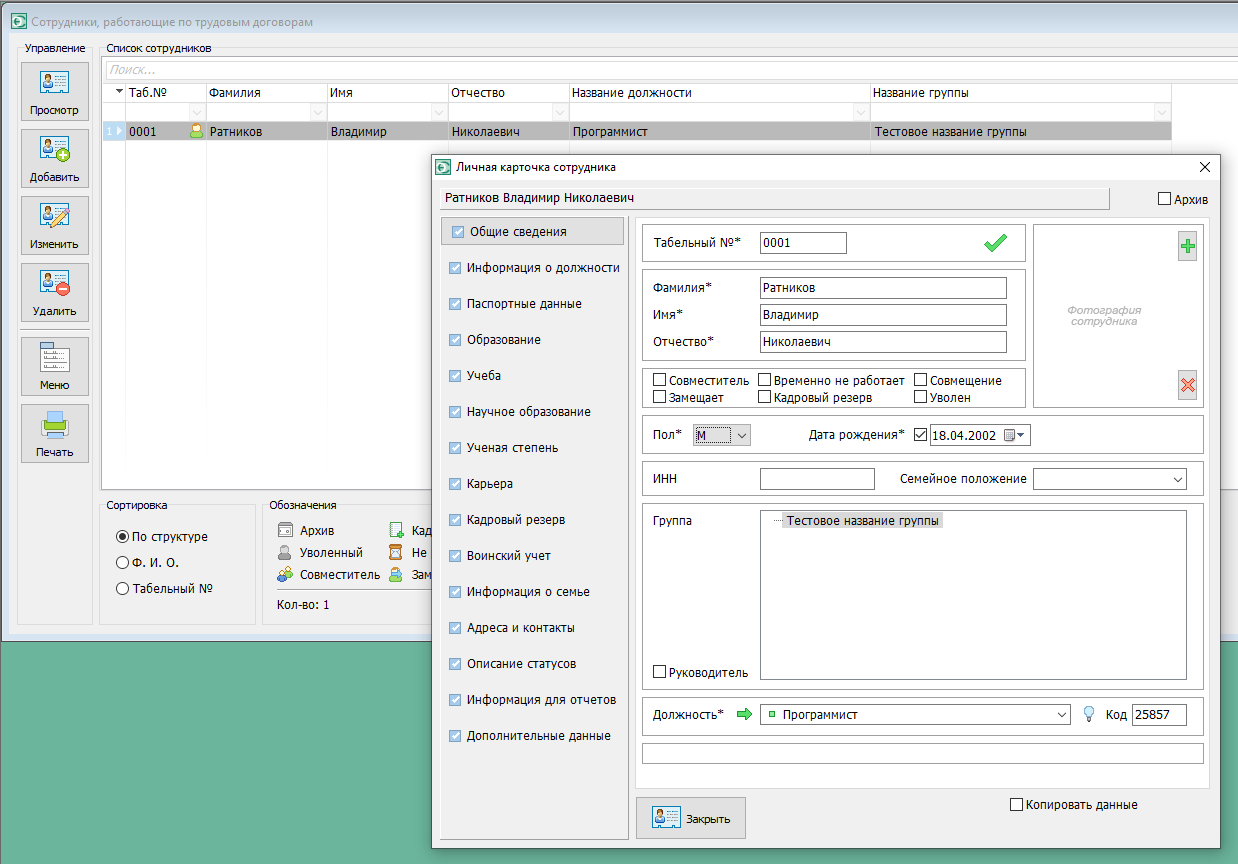


Рисунок 5. Личная карточка сотрудника

«Кадры Плюс» - профессиональное приложение для автоматизации работы отдела кадров, предназначенное для работы в крупных организациях. Программа имеет широкий спектр возможностей, которые пригодятся при учёте и поиске сотрудников. Из недостатков стоит отметить неудобный интерфейс.

# **2. Проектная часть**

## **2.1 Построение диаграммы прецедентов**

На Рисунке 6 представлена диаграмма прецедентов. На ней показаны все возможные действия Сотрудника отдела кадров.

Сотрудник отдела кадров при работе с приложением постоянно взаимодействует с базой данных. Через неё он может уволить сотрудника или добавить нового. Также, он может получить данные о уже записанных сотрудниках и отредактировать.

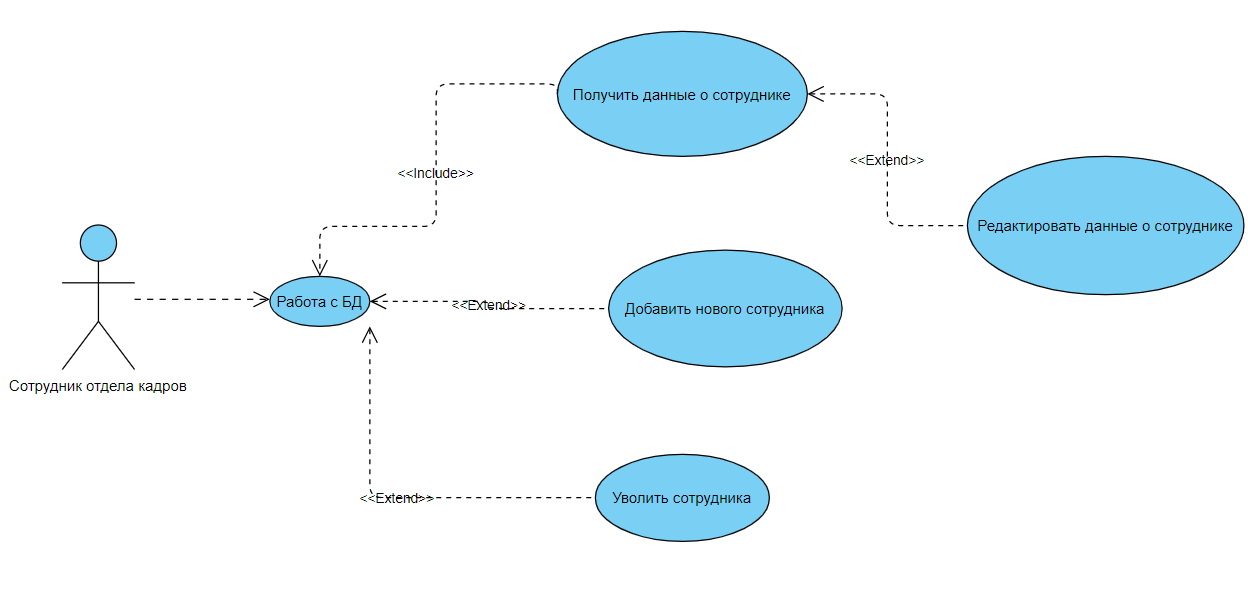


Рисунок 6. Диаграмма прецедентов сотрудника отдела кадров

## **2.2. Выбор инструментов**

При выборе инструментов сравнивались все достоинства и недостатки каждого инструмента, знания и время на обучение.

В качестве языка программирования был выбран C#. Он имеет схожий синтаксис как у Python и C++, а также он объектно-ориентированный, что подходит для разработки графических приложений.

В качестве фреймворка был выбран Blazor Server. Он позволяет создавать серверные приложения с использованием языка программирования C# и компонентов Razor Pages.

В качестве среды разработки была выбрана Microsoft Visual Studio 2022, так как только на ней можно писать приложения на платформе Blazor Server. На Рисунке 7 показано окно разработки в Visual Studio.

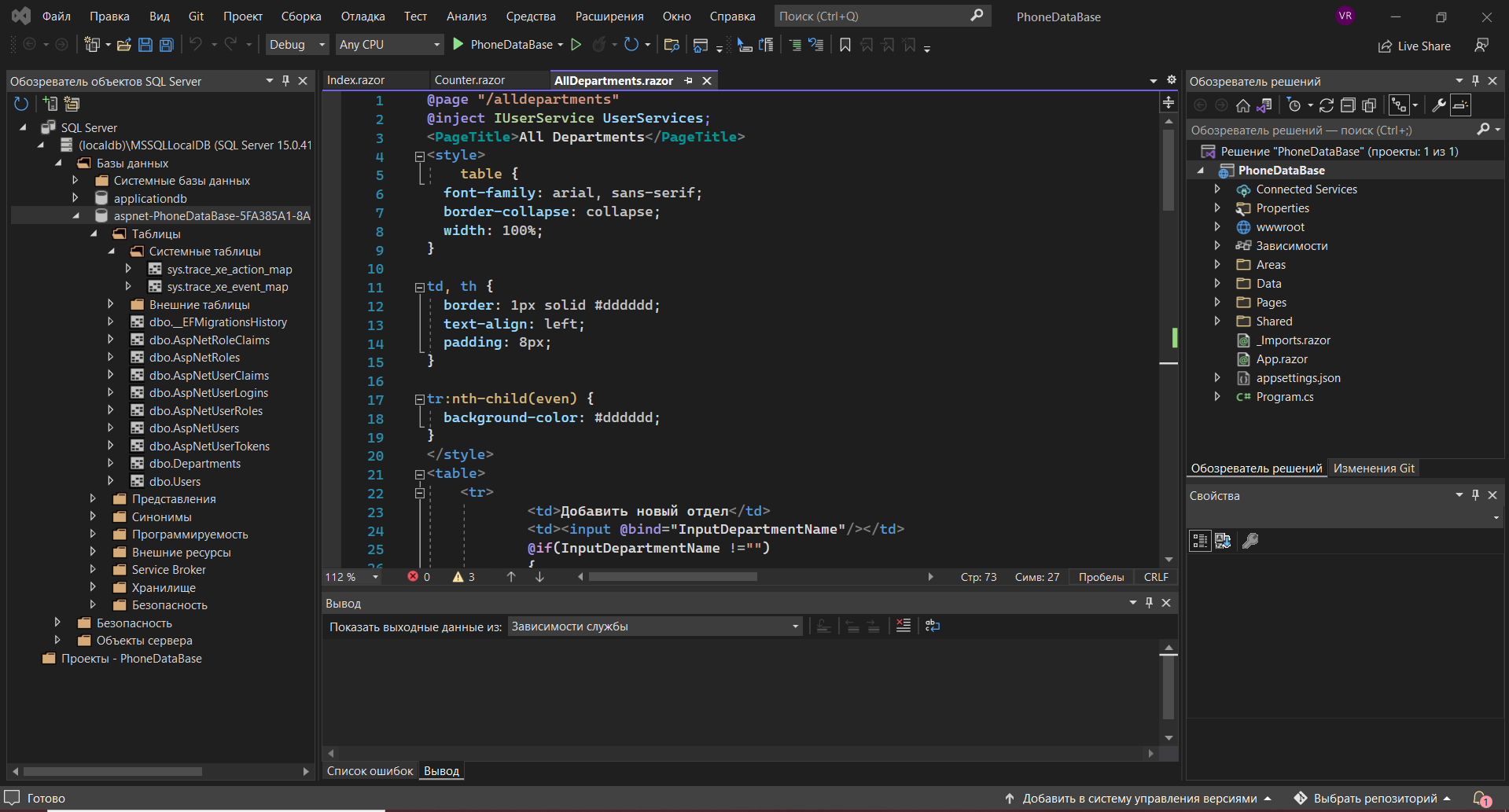


Рисунок 7. Разработка в Visual Studio

## **2.3. Проектирование сценария**

На Рисунке 7 приведён сценарий использования программы

Сотрудник отдела кадров запускает программу, открывает браузер и вводит в адресную строку адрес хоста. Обычно это <https://localhost:5001>.

Первоначально открывается страница со всеми, кроме удаленных, сотрудниками. Их можно редактировать, а также удалять.

При нажатии на Deleted, или же при добавлении в ссылку /counter, можно переключиться на страницу с удаленными пользователями. Точно также можно переключиться на страницу с отделами, нажав на Departments или добавив в адрес страницы /alldepartments.

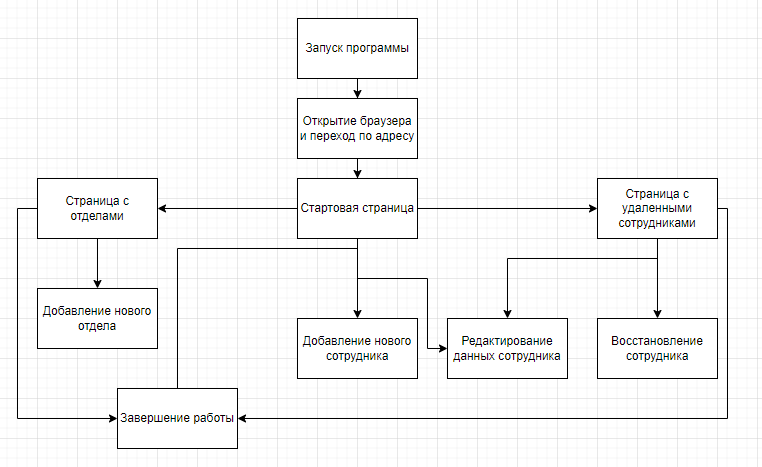


Рисунок 8. Сценарий использования

## **2.4. Построение диаграммы классов**

На Рисунке 8. Диаграмма классов приведена диаграмма всех классов, их методов и взаимодействие между собой

Класс «Base User» является базовым и хранит в себе только Id и Status Id. Он нужен чтобы в дальнейшем наследовать от него другие классы

Класс «Department» является наследником класса «Base User» и содержит DepartmentName

Класс «User» также является наследником класса «Base User», и содержит в себе поля Fio, Email, NumberPhone, Department, Department Position и DepartmentNumber. Department — одноименный объект класса Department.

Класс «UserServices» содержит методы класса «User».

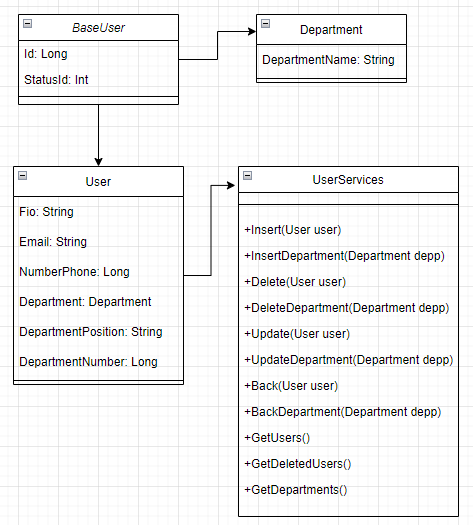


Рисунок 9. Диаграмма классов

## **2.5. Описание стартового модуля**

В стартовый модуль входит файл program.cs который предоставляет точку входа в приложение, в котором настраивается конвейер обработки запроса, внедряются сервисы, осуществляется конфигурация приложения, запускает и конфигурирует хост.

Program.cs является шаблонным файлом, который автоматически конфигурируется при создании проекта. В файле можно ничего не изменять, все страницы и классы хранятся в других модулях. Он нужен только для запуска приложения. Для работы с базой данных понадобится её подключить в этом файле.

**Листинг 1. Создание контейнера**

var builder = WebApplication.CreateBuilder(args);

Приложение на Blazor Server всегда начинается с класса WebApplicationBuilder. Для работы приложения нам понадобится метод WebApplication.CreateBuilder() для создания объекта этого класса.

**Листинг 2. Регистрация групп служб в контейнере**

// Add services to the container.

var connectionString = builder.Configuration.GetConnectionString("DefaultConnection");

builder.Services.AddDbContext<ApplicationDbContext>(options =>

options.UseSqlServer(connectionString));

builder.Services.AddDatabaseDeveloperPageExceptionFilter();

builder.Services.AddDefaultIdentity<IdentityUser>(options => options.SignIn.RequireConfirmedAccount = true)

.AddEntityFrameworkStores<ApplicationDbContext>();

builder.Services.AddRazorPages();

builder.Services.AddServerSideBlazor();

builder.Services.AddScoped<AuthenticationStateProvider, RevalidatingIdentityAuthenticationStateProvider<IdentityUser>>();

builder.Services.AddScoped<IUserService, UserServices>();

**Листинг 3. Создание приложения**

var app = builder.Build();

После того как зарегестрировали все необходимые службы, создаем приложение на основе контейнера с помощью метода builder.Build();

**Листинг 4. Проверка, на какой стадии находится приложение**

if (app.Environment.IsDevelopment())

{

app.UseMigrationsEndPoint();

}

else

{

app.UseExceptionHandler("/Error");

app.UseHsts();

}

Если приложение всё ещё находится на стадии разработки, проверяем на наличие ожидающих миграций. В противном случае выкидываем страницу ошибки

**Листинг 5. Добавление переадресации**

app.UseHttpsRedirection();

Теперь помимо собственного адреса, проект имеет переадресацию на такой же ресурс, но с ссылкой на htttps:

**Листинг 6. Настройка статистических файлов**

app.UseStaticFiles();

Добавление в проект возможности отправки статистических файлов по умолчанию, не обращаясь к ним по полному пути

**Листинг 7. Добавление конечных точек**

app.UseRouting();

Добавляет конечные точки для обработки запросов по определенным маршрутам.

**Листинг 8. Авторизация**

app.UseAuthentication();

app.UseAuthorization();

Добавление возможности авторизации в проекте. Сама авторизация не реализована и не задействована в проекте

**Листинг 9. Завершающая настройка**

app.MapControllers();

app.MapBlazorHub();

app.MapFallbackToPage("/\_Host");

Метод MapControllers позволяет в проекте не определять определенных маршрутов. Метод MapBlazorHub() организует связь клиента с сервером через SignalR соединение. Метод MapFallbackToPage("/\_Host") устанавливает Razor pages страницы по умолчанию.

**Листинг 10. Запуск приложения**

app.Run();

## **2.6. Описание спецификаций к модулям**

В данном разделе будут описаны спецификации в модулях курсового проекта.

**Program.cs**

В стартовом модуле (Program.cs) содержатся методы класса WebApplicationBuilder, которые используются для настройки и запуска приложения. В этом модуле задействовано несколько объектов:

1. builder ­­ — Объект класса WebApplicationBuilder. Нужен для установки конфигурации приложения, добавления сервисов, настройки логгирования в приложении и установки окружения приложения.
2. app — Объект класса WebApplication, объявляется методом build() и объектом builder. Применяется для установки маршрутов, управления обработкой запроса и получением сервисов.

В дополнительных модулях, которые реализовывают страницы проекта, состоят из объектов для реализации графического интерфейса и объектов, хранящих данные из базы данных. Объекты дополнительных модулей:

1. <table> — Таблица, состоящая из строк (<tr> <tr\>) и столбцов (<td> </td>). Необходима для вывода данных из базы данных и занесения в таблицу.
2. InputFio — окно для ввода ФИО нового пользователя.
3. InputEmail — окно для ввода почты нового пользователя.
4. InputNumber — окно для ввода номера телефона нового пользователя.
5. ListId — выпадающий список для выбора отдела новому пользователю.
6. InputPosition — окно для ввода позиции нового пользователя. Если он не причастен ни к одному отделу, позиция не должна вводится.
7. AddUser — кнопка для добавления нового пользователя. Не активна пока не будут введены все данные.
8. EditUser — кнопка для изменения данных уже существующего пользователя.
9. DeleteUser — кнопка для удаления пользователя.
10. EditFio — кнопка для изменения ФИО конкретного пользователя.
11. EditEmail — кнопка для изменения электронной почты конкретного пользователя.
12. EditNumberPhone — кнопка для изменения номера телефона конкретного пользователя.
13. EditId — кнопка для изменения отдела конкретного пользователя
14. EditDepartmentPosition — кнопка для изменения отдела
15. SaveUser — кнопка для сохранения отредактированного пользователя
16. CancelUser — кнопка для отката изменений

**BaseUser.cs**

Модуль BaseUser содержит класс BaseUser.

1. Id — поле для персонального Id пользователя или отдела
2. StatusId — поле для интерфиксации статуса пользователя

**Department.cs**

Department.cs содержит класс Department, который наследует класс BaseUser.

1. DepartmentName — поле для названия отдела

**User.cs**

User.cs содержит класс User, который наследует класс BaseUser. Этот класс хранит персональные данные пользователя.

1. Fio — поле для Фамилии и инициалов
2. Email — поле для электронной почты
3. NumberPhone — поле для номера телефона
4. Department — объект класса Department.
5. DepartmentPosition — поле для названия позиции в отделе
6. DepartmentNumber — поле для номера отдела

**UserServices.cs**

Класс UserServices наследует интерфейс IUserService. UserServices используется для работы с базой данных и хранит в себе методы и функции класса.

1. GetUsers() — метод для получения списка всех пользователей, кроме удаленных.
2. GetDeletedUsers() — метод для получения списка удаленных пользователей.
3. GetDepartments() — метод для получения списка всех отделов
4. Insert() — метод для добавления нового пользователя в базу данных.
5. InsertDepartment() — метод для добавления нового отдела в базу данных.
6. Delete() — метод для удаления пользователя из базы данных.
7. DeleteDepartment() — метод для удаления отдела из базы данных.
8. Update() — метод для обновление пользователя в базе данных.
9. UpdateDepartment() — метод для обновления отдела в базе данных.
10. Back() — метод для смены статуса пользователя.
11. BackDepartment() — метод для смены статуса отдела.

## **2.7. Описание модулей**

Помимо стартового модуля, проект содержит модули классов и Razor pages.

### **2.7.1 BaseUser.cs; Department.cs; User.cs; UserServices.cs**

**Листинг 11. Модуль BaseUser.cs**

public class BaseUser

{

public long Id { get; set; }

public int StatusId { get; set; }

}

public enum Statuses

{

Deleted = 0,

Working = 1,

Edit = 2

}

Класс BaseUser хранит Id пользователя, а также его StatusId. Для удобства использован enum. При StatusId = 0 пользователь заносится в список удаленных, а при StatusId = 2 его данные редактируется в текущий момент.

**Листинг 12. Класс Department**

public class Department: BaseUser

{

public string DepartmentName { get; set; } //Название подразделения

}

Класс Department наследует класс BaseUser и используется для хранения названия Отдела.

**Листинг 13. Класс Department**

public class User: BaseUser

{

public string Fio { get; set; }

public string Email { get; set; }

public long NumberPhone { get; set; }

public virtual Department? Department { get; set; }

public string? DepartmentPosition { get; set; } //Название должности

public long? DepartmentNumber { get; set; } //Номер отдела

}

Объекты класса User наследуют все переменные из класса BaseUser, а также хранят персональные данные пользователей (ФИО, Эл.Почта, Номер телефона, Отдел, Название должности и Номер отдела).

**Листинг 14. Класс ApplicationDbContext**

public class ApplicationDbContext : IdentityDbContext

{

public DbSet<User> Users { get; set; }

public DbSet<Department> Departments { get; set; }

public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options)

: base(options)

{

}

}

ApplicationDbContext необходим для взаимодействия с базой данных.

**Листинг 15. Класс UserServices**

public class UserServices: IUserService

{

private readonly ApplicationDbContext \_applicationDbContext;

public UserServices(ApplicationDbContext applicationDbContext)

{

this.\_applicationDbContext = applicationDbContext;

}

public void Insert(User user)

{

\_applicationDbContext.Users.Add(user);

\_applicationDbContext.SaveChanges();

}

public void InsertDepartment(Department depp)

{

\_applicationDbContext.Departments.Add(depp);

\_applicationDbContext.SaveChanges();

}

public void Delete(User user)

{

user.StatusId = (int)Statuses.Deleted;

\_applicationDbContext.SaveChanges();

}

public void DeleteDepartment(Department depp)

{

depp.StatusId = (int)Statuses.Deleted;

\_applicationDbContext.SaveChanges();

}

public void Update(User user)

{

\_applicationDbContext.SaveChanges();

}

public void UpdateDepartment(Department depp)

{

\_applicationDbContext.SaveChanges();

}

public void Back(User user)

{

user.StatusId = (int)Statuses.Working;

\_applicationDbContext.SaveChanges();

}

public void BackDepartment(Department depp)

{

depp.StatusId = (int)Statuses.Working;

\_applicationDbContext.SaveChanges();

}

public async Task<List<User>> GetUsers() => await \_applicationDbContext.Users.Where(t => t.StatusId != (int)Statuses.Deleted).ToListAsync();

public async Task<List<User>> GetDeletedUsers() => await \_applicationDbContext.Users.Where(t => t.StatusId == (int)Statuses.Deleted).ToListAsync();

public async Task<List<Department>> GetDepartments()

{

//return await \_applicationDbContext.Departments.Where(t => t.StatusId != (long)Statuses.Deleted).ToListAsync();

return await \_applicationDbContext.Departments.Where(t => t.StatusId != (long)Statuses.Deleted).ToListAsync();

}

}

UserService реализовывает методы классов User и Department. Добавление, удаление, обновление и изменение статуса пользователя и отдела. UserService наследует интерфейс IUserService, код которого находится в приложении 1.

### **2.7.2 Модули Razor pages**

Файлы с расширение .razor служат для реализации страниц в проекте. Razor pages состоят из html-кода и языка программирования C#.

#### **Index.razor**

Index.razor имеет адрес “/” и открывается по умолчанию. На этой странице выводится список всех пользователей, за исключением удалённых.

**Листинг 16. Название страницы**

<**PageTitle**>Data Base</**PageTitle**>

**Листинг 17. Стиль для таблицы**

<style> @\*Стиль для создания красивой таблицы\*@

table {

font-family: arial, sans-serif;

border-collapse: collapse;

width: 100%;

}

td, th {

border: 1px solid #dddddd;

text-align: left;

padding: 8px;

}

tr:nth-child(even) {

background-color: #dddddd;

}

</style>

**Листинг 18. Таблица ввода**

<table>

<tr>

<td>ФИО</td>

<td><input @bind="InputFio"/></td>

<td>Email</td>

<td><input @bind="InputEmail"/></td>

<td>Номер телефона</td>

<td><input @bind="InputNumber"/></td>

<td>Отдел</td>

<td>

@if (Listdepartments != null && Listdepartments.Count > 0) @\*Вывод списка отделов, если их количество больше нуля\*@

{

<select @bind="ListId">

@foreach (var uom in Listdepartments)

{

<option value="@uom.Id">@uom.DepartmentName</option>

}

</select>

}</td>

<td>Позиция</td>

<td><input @bind="InputPosition"/></td>

@if (InputFio != "" && InputEmail != "" && InputNumber != 0) @\*Если ячейки не пустые, разрешаем добавить пользователя\*@

{

<td><button @onclick="AddUser" class="btn btn-success">+ Добавить</button></td>

}

</tr>

</table>

Данная таблица служит для ввода новых пользователей. На Рисунке 10 приведён её внешний вид.



Рисунок 10. Внешний вид таблицы

**Листинг 19. Таблица пользователей**

@if(users == null)

{

<p>Идёт загрузка</p>

}

else{

<table>

<tr>

<th>Id</th>

<th>Фио</th>

<th>Почта</th>

<th>Номер телефона</th>

<th>Отдел</th>

<th>Позиция</th>

</tr>

@foreach(var user in users){ @\*Вывод пользователей\*@

@if(user.StatusId == (int)Statuses.Working){

<tr>

<th> @user.Id</th>

<th> @user.Fio</th>

<th> @user.Email</th>

<th> @user.NumberPhone</th>

<th> @user.Department?.DepartmentName</th>

<th> @user.DepartmentPosition</th>

<td>

<button @onclick="() => EditUser(user)" class="btn btn-warning"><i class="oi oi-pencil"></i> Редактировать</button>

<button @onclick="() => DeleteUser(user)" class="btn btn-danger">- Удалить</button></td>

</tr>

}

else if(user.StatusId == (int)Statuses.Edit) @\*Вывод редактируемого пользвателя\*@

{

<tr>

<td>@user.Id</td>

<td><input @bind="EditFio"/></td>

<td><input @bind="EditEmail"/></td>

<td><input @bind="EditNumberPhone"/></td>

<td>

@if (Listdepartments != null && Listdepartments.Count > 0)

{

<select @bind="EditId">

@foreach (var dep in Listdepartments)

{

<option value="@dep.Id">@dep.DepartmentName</option>

}

</select>

}

</td>

<td><input @bind="EditDepartmentPosition"/></td>

<td>

<button @onclick="() => SaveUser(user)" class="btn btn-success"><i class="oi oi-flash"></i> Сохранить</button>

<button @onclick="() => CancelUser(user)" class="btn btn-danger">Отмена</button>

</td>

</tr>

}

else

{

<tr><td>Error</td></tr>

}

}

</table>

If(users == null) проверяет, пустой лист список пользователей. Если пользователей нет, выводится сообщение о загрузке. В ином случае он выводит пользователей из переменной класса users. При этом, он выводит только тех пользователей, статус которых равен 1 (Working).

**Листинг 20. Переменные**

public List<User> users;

public List<Department> Listdepartments;

public Department EditDepartment = null;

@\*Переменные для ввода данных для новых пользвателей\*@

public string InputFio{ get; set; }

public string InputEmail{ get; set; }

public long InputNumber{ get; set; }

public string InputPosition{ get; set; }

public long ListId { get; set; }

@\*Переменные для редактирования\*@

public string EditFio { get; set; }

public string EditEmail { get; set; }

public long EditNumberPhone { get; set; }

public string EditDepartmentPosition { get; set; }

public long EditId { get; set; }

Переменные, используемые на этой странице. users хранит в себе список всех пользователей из базы данных. Listdepartments хранит в себе все отделы. EditDepartment нужен был для редактирования отдела, однако не используется.

**Листинг 21. Инициализация**

protected override async Task OnInitializedAsync() @\*Инициализация\*@

{

users = await UserServices.GetUsers();

Listdepartments = await UserServices.GetDepartments();

}

Асинхронное считывание списка пользователей и отделов из базы данных.

**Листинг 22. Добавление пользователя**

public void AddUser() @\*Добавление нового пользователя\*@

{

var department = Listdepartments.FirstOrDefault(t => t.Id == ListId);

var user = new User

{

Fio = InputFio,

Email = InputEmail,

NumberPhone = InputNumber,

DepartmentPosition = InputPosition,

Department = department,

StatusId = (int)Statuses.Working,

};

UserServices.Insert(user);

users.Add(user);

InputFio = ""; InputEmail = ""; InputNumber = 0; InputPosition = "";

}

Первоначально новый пользователь заносится в базу данных с помощью метода UserServices.Insert а затем в переменную класса users.

**Листинг 23. Удаление пользователя**

public void DeleteUser(User user) @\*Удаление пользователя\*@

{

UserServices.Delete(user);

users.Remove(user);

}

**Листинг 24. Сохранение**

public void SaveUser(User user) @\*Сохранение данных после редактирования\*@

{

user.Fio = EditFio;

user.Email = EditEmail;

user.NumberPhone = EditNumberPhone;

user.DepartmentPosition = EditDepartmentPosition;

//user.Department.Id = EditId;

user.StatusId = (int)Statuses.Working;

UserServices.Update(user);

}

**Листинг 25. Редактирование**

public void EditUser(User user) @\*Редактирование пользователя\*@

{

users.ForEach(t => t.StatusId = (int)Statuses.Working);

user.StatusId = (int)Statuses.Edit;

EditFio = user.Fio;

EditEmail = user.Email;

EditNumberPhone = user.NumberPhone;

EditDepartmentPosition = user.DepartmentPosition;

}

**Листинг 26. Отмена**

public void CancelUser(User user) @\*Отмена редактирования\*@

{

user.StatusId = (int)Statuses.Working;

}

#### **Counter.razor**

Counter.razor почти в точности идентичен Index.razor. Он необходим для редактирования и восстановления удалённых пользователей. Адрес страницы — "/counter"

Отличия заключается в том, что в этом файле нет таблицы для добавления новых пользователей, а также вместо существующих пользователей выводятся удалённые.

**Листинг 27. Вывод удалённых пользователей**

@if(user.StatusId == (int)Statuses.Deleted){

<tr>

<th> @user.Id</th>

<th> @user.Fio</th>

<th> @user.Email</th>

<th> @user.NumberPhone</th>

<th> @user.Department?.DepartmentName</th>

<th> @user.DepartmentPosition</th>

<td>

<button @onclick="() => EditUser(user)" class="btn btn-warning"><i class="oi oi-pencil"></i> Редактировать</button>

<button @onclick="() => BackUser(user)" class="btn btn-success">- Вернуть</button></td>

</tr>

}

**Листинг 28. Восстановление**

public void BackUser(User user) @\*Вернуть удалённого пользователя\*@

{

UserServices.Back(user);

}

В приложении 2 приведены все функции файла Counter.razor

#### **AllDepartments.razor**

AllDepartments — страница, имеющая адрес "/alldepartments". Она выводит названия всех отделов. AllDepartments позволяет редактировать и удалять отделы.

**Листинг 29. Добавление**

<table>

<tr>

<td>Добавить новый отдел</td>

<td><input @bind="InputDepartmentName"/></td>

@if(InputDepartmentName !="")

{

<td><button @onclick="AddDepartments" class="btn btn-success">+</button></td>

}

</tr>

</table>

Таблица с кнопкой для добавления нового отдела.

**Листинг 30. Таблица с отделами**

<table>

@if(Listdepartments != null && Listdepartments.Count > 0)

@foreach(var depp in Listdepartments)

{

@if (depp.StatusId == (int)Statuses.Working)

{

<tr>

<td>@depp.DepartmentName</td>

<td>

<button @onclick="() => EditDepartments(depp)" class="btn btn-warning"><i class="oi oi-pencil"></i> Редактировать</button>

<button @onclick="() => DeleteDepartments(depp)" class="btn btn-danger">- Удалить</button>

</td>

</tr>

}

else if(depp.StatusId == (int)Statuses.Edit)

{

<tr>

<td><input @bind="EditDepartmentName"></td>

<td>

<button @onclick="() => SaveDepartments(depp)" class="btn btn-success"><i class="oi oi-flash"></i> Сохранить</button>

<button @onclick="() => CancelDepartments(depp)" class="btn btn-danger">Отмена</button>

</td>

</tr>

}

else

{

<tr><td>Error</td></tr>

}

}

</table>

Эта таблица почти идентична таблицам из других модулей. Выводится название отдела, а справа от него две кнопки для редактирования и удаления.

**Листинг 31. Таблица с отделами**

public List<Department> Listdepartments;

public string InputDepartmentName {get;set;}

public string EditDepartmentName {get;set;}

Listdepartments — список всех отделов. InputDepartmentName — переменная для ввода названия нового отдела. EditDepartmentName — переменная для редактирования названия отдела.

**Листинг 32. Инициализация отделов**

protected override async Task OnInitializedAsync() @\*Инициализация\*@

{

Listdepartments = await UserServices.GetDepartments();

}

**Листинг 33. Добавление отдела**

public void AddDepartments() @\*Добавление нового отдела\*@

{

var newdepp = new Department

{

DepartmentName = InputDepartmentName,

StatusId = (int)Statuses.Working,

};

UserServices.InsertDepartment(newdepp);

Listdepartments.Add(newdepp);

InputDepartmentName = "";

}

**Листинг 34. Удаление отдела**

public void DeleteDepartments(Department depp) @\*Удаление отдела\*@

{

UserServices.DeleteDepartment(depp);

Listdepartments.Remove(depp);

}

**Листинг 35. Редактирование отдела**

public void EditDepartments(Department depp) @\*Редактирование\*@

{

Listdepartments.ForEach(t => t.StatusId = (int)Statuses.Working);

depp.StatusId = (int)Statuses.Edit;

EditDepartmentName = depp.DepartmentName;

}

**Листинг 36. Сохранение**

public void SaveDepartments(Department depp) @\*Сохранение данных после редактирования\*@

{

depp.DepartmentName = EditDepartmentName;

depp.StatusId = (int)Statuses.Working;

UserServices.UpdateDepartment(depp);

}

**Листинг 37. Отмена редактирования**

public void CancelDepartments(Department depp) @\*Отмена редактирования\*@

{

depp.StatusId = (int)Statuses.Working;

}

## **2.8. Описание тестовых наборов модулей**

Проведём тестирование. Добавим нового пользователя, создадим новый отдел, добавим ещё одного пользователя с недавно созданным отделом, удалим его и восстановим.

1. Добавим нового пользователя

На главной странице заполняем верхние ячейки данными. Нажимаем на кнопку “Добавить” и проверяем. На 11 рисунке показаны заполненные поля для ввода данных. На 12 рисунке удостоверяемся в этом.

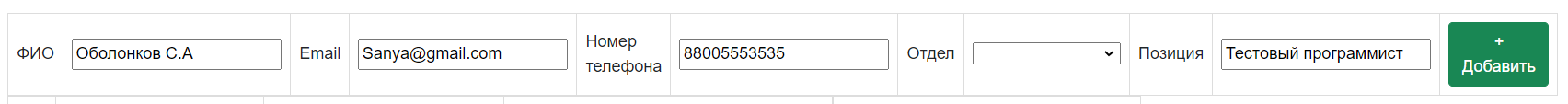


Рисунок 11. Заполнение данных

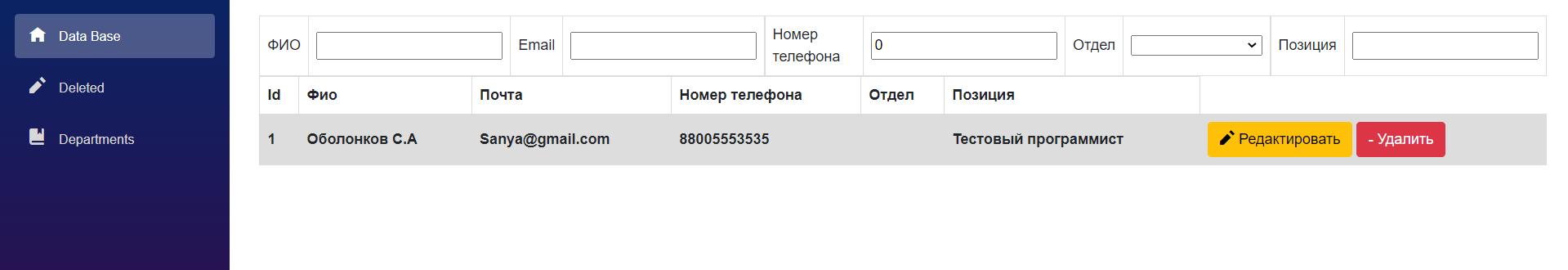


Рисунок 12. Проверка

Пользователь Оболонков С.А. был добавлен. Все данные соответствуют своим значениям. Данные пользователя можно редактировать, а пользователя удалить.

1. Добавляем новый отдел. Даём ему название “Тестовый отдел”. На рисунке 13 проиллюстрирован процесс ввода. Результат показан на 14 рисунке.

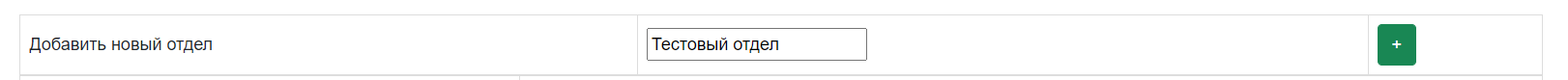


Рисунок 13. Тестовый отдел

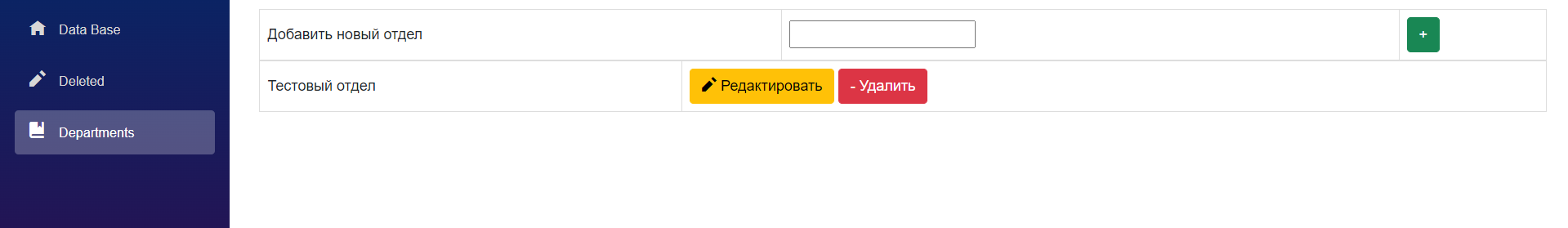


Рисунок 14. Проверка отдела

1. Добавляем нового пользователя и присваиваем ему новый отдел. На 15 рисунке показан результат добавления нового сотрудника.

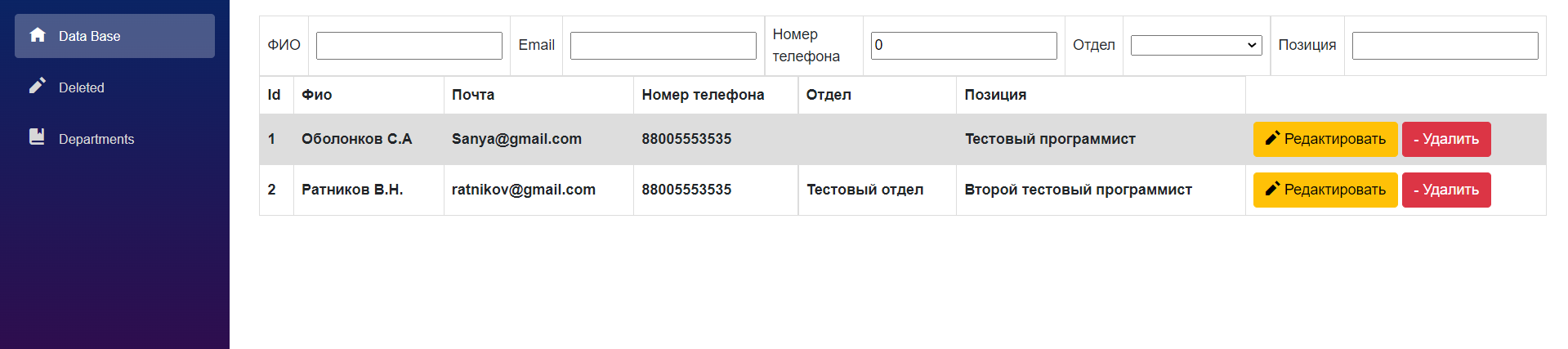


Рисунок 15. Новый сотрудник с отделом

1. Удаляем пользователя. Проверяем его наличие в списке удаленных. На 16 рисунке проиллюстрирован результат удаления пользователя.

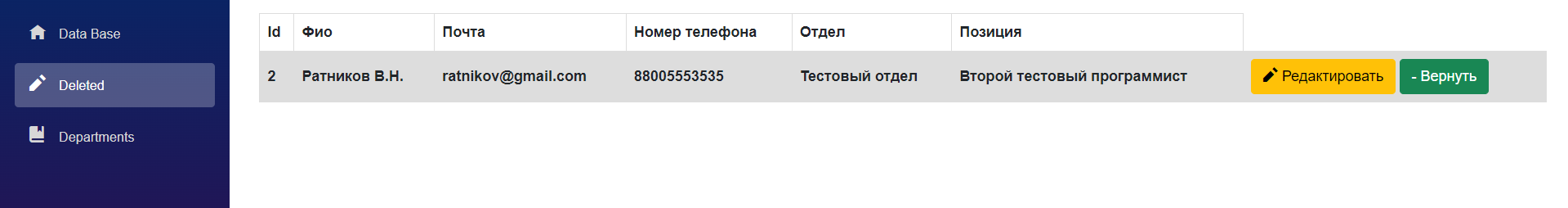


Рисунок 16. Проверка на наличие в списке удаленных

Восстанавливаем пользователя и убеждаемся, что проект прошел все тесты. Были созданы два пользователя и отдел. Данные пользователей изменяются, сами пользователи легко удаляются и восстанавливаются.

## **2.9. Описание применения средств отладки**

В этом разделе будет показано как проводится откладка приложение и как находятся ошибки в коде.

Откладка — это процесс поиска и исправления ошибок в коде программы. Откладка и сборка приложения в среде Microsoft Visual Studio проходят автоматически при её запуске. отладка приложения обычно означает запуск приложения с подключенным отладчиком (то есть в режиме отладчика)

В режиме отладчика можно просматривать значения из переменных и задавать контрольные значения для переменных, чтобы отслеживать изменение значений. Также можно устанавливать точки остановки и запуска отладчика. Точки остановки указывают, где следует остановить программу.

Для запуска приложения с откладкой следует нажать на клавишу F5. Задаем 25 строчку Program.cs файла как точку остановки. Запускаем программу.

Программа проработала пару секунд и остановилась, так как дошла до точки остановки. Точка остановки отображена на 17 рисунке.

В окне Видимые отображены переменные, используемые в текущей строке. Окно отображены на 18 рисунке.

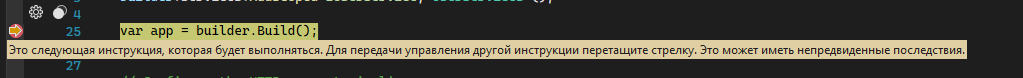


Рисунок 17. Точка остановки

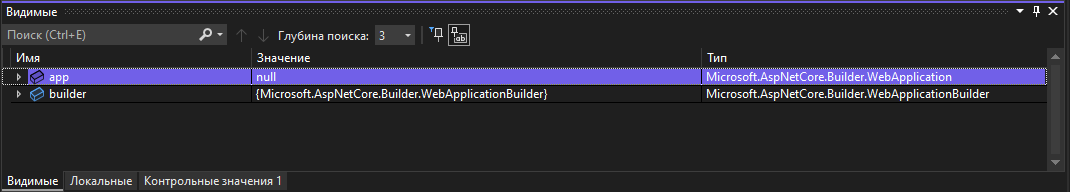


Рисунок 18. Окно видимые

# **Эксплуатационная часть**

## **Руководство оператора**

**Аннотация**

В данном руководстве приведены инструкции по применению PhoneDataBase, которая служит для автоматизации учета кадров на предприятии. Программа предназначена для сотрудников отдела кадров.

1. **Назначение программы**

Программа PhoneDataBase предназначена для автоматизации учета кадров на производственном предприятии. Добавление, редактирование и удаление сотрудников из базы данных.

* 1. **Функциональное назначение**

Программа предоставляет возможность хранить, записывать, перезаписывать и удалять данные о пользователях в базе данных SQL. Все действия совершаются во вкладке браузера.

* 1. **Эксплуатационное назначение**

PhoneDataBase может эксплуатироваться на малых-средних предприятиях сотрудниками отдела кадров.

1. **Состав функций**
   1. **Функция смены страницы**

Данная функция позволяет перейти с одной страницы на другую, не изменяя адрес вручную.

* 1. **Функция добавления сотрудника**

Данная функция позволяет внести в базу данных нового сотрудника.

* 1. **Функция редактирования сотрудника**

Данная функция позволяет редактировать данные уже существующего сотрудника.

* 1. **Функция отмены редактирования**

Данная функция позволяет отменить все изменения и вернуть изначальные значения сотрудника.

* 1. **Функция сохранения изменений**

Данная функция позволяет применить все изменения после редактирования.

* 1. **Функция удаления сотрудника**

Данная функция позволяет удалить выбранного сотрудника. Удаленный сотрудник переносится на другую страницу.

* 1. **Функция восстановления сотрудника**

Данная функция восстанавливает удаленного сотрудника.

* 1. **Функция добавления отдела**

Данная функция добавляет новый отдел.

* 1. **Функция редактирования отдела**

Данная функция позволяет изменить название отдела.

* 1. **Функция удаления отдела**

Данная функция позволяет удалить отдел.

1. **Условия выполнения программы**
   1. **Минимальные системные требования**

Операционная система: не ранее Windows 7.

ОЗУ: 1 ГБ.

Место на диске: 1 ГБ.

Разрешение экрана: не ниже 1024 на 768.

* 1. **Минимальный состав программных средств**

Наличие установленной Microsoft Visual Studio 2022 с необходимыми рабочими нагрузками (ASP.NET и разработка веб-приложений; Разработка классических приложений .NET), а также с несколькими отдельными компонентами (.NET Framework 4.8; SQL Server Data Tools; SQL Server Express 2019 LocalDB; SQL Server ODBC Driver).

Инструкцию по установке и настройке Visual Studio можно прочитать на официальном сайте [Microsoft](https://docs.microsoft.com/ru-ru/visualstudio/install/install-visual-studio?view=vs-2019).

* 1. **Требования к персоналу**

Для использование этого приложения понадобятся сотрудники из отдела кадров, с уверенным уровнем владения компьютера.

1. **Выполнение программы**
   1. **Открытие и установка миграции**

Открываем Microsoft Visual Studio. Выбираем “Открыть проект или решение”. В появившемся проводнике выбираем папку PhoneDataBase и жмём кнопку Открыть, или же в этой папке открываем решение PhoneDataBase.sln

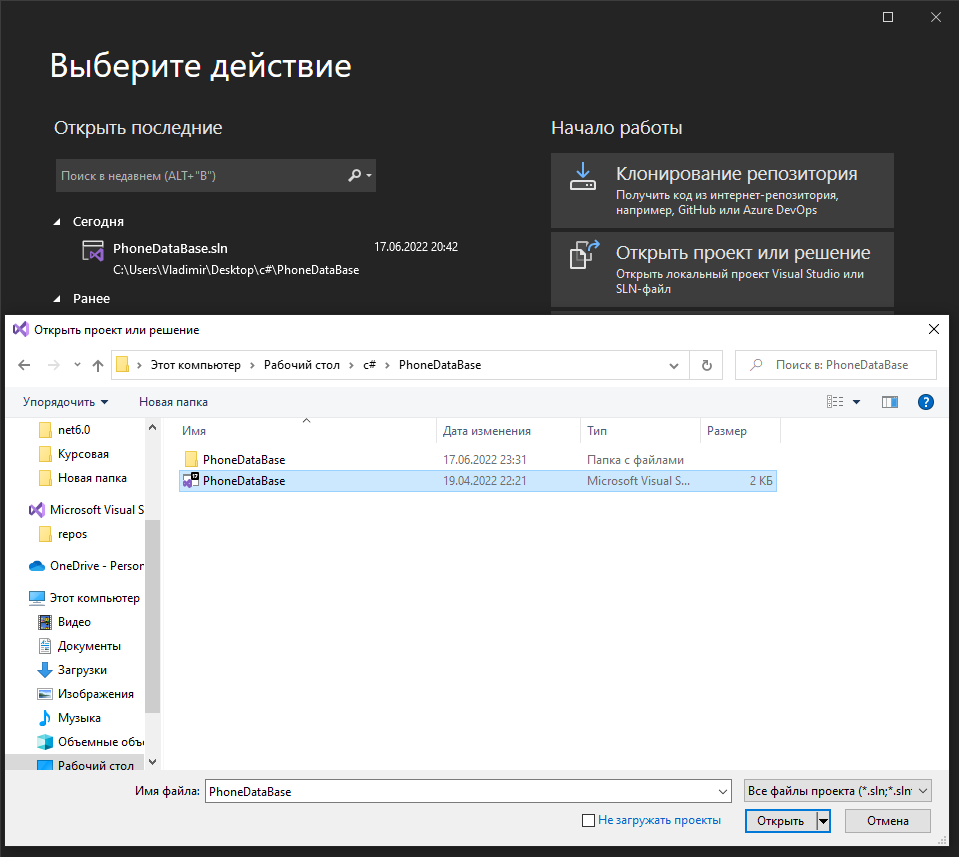


Рисунок 19. Открытие решения

Открываем проект. Нажимаем на клавишу F5, или же в разделе Откладка выбираем. Начать откладку. Запускается командная строка, после завершения которой открывается вкладка браузера с нашим проектом.

Появляется ошибка, в которой сказано, что невозможно получить доступ к базе данных. Жмём на голубую кнопку Apply Migrations, ждём завершения и перезапускаем страницу.

Это необходимо проделать только при первом запуске. В дальнейшем, можно не запускать Visual Studio, а открывать PhoneDataBase.exe файл, по пути PhoneDataBase\PhoneDataBase\bin\Debug\net6.0. Также, приложение можно опубликовать и собрать в папку для дальнейшего удобства.

* 1. **Добавление сотрудника**

Чтобы добавить сотрудника заполнить поля со значениями и нажмите на зелёную кнопку

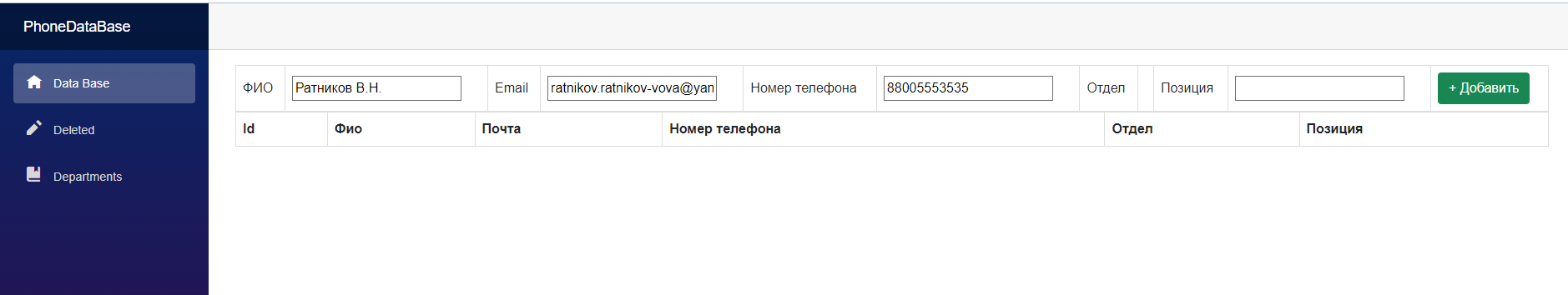


Рисунок 20. Добавление

* 1. **Редактирование сотрудника**

Чтобы изменить данные сотрудника, нажмите на желтую кнопку Редактировать. Измените нужные вам параметры и нажмите на зелёную кнопку Сохранить. Если нужно отметить редактирование, нажмите на красную кнопку Отмена

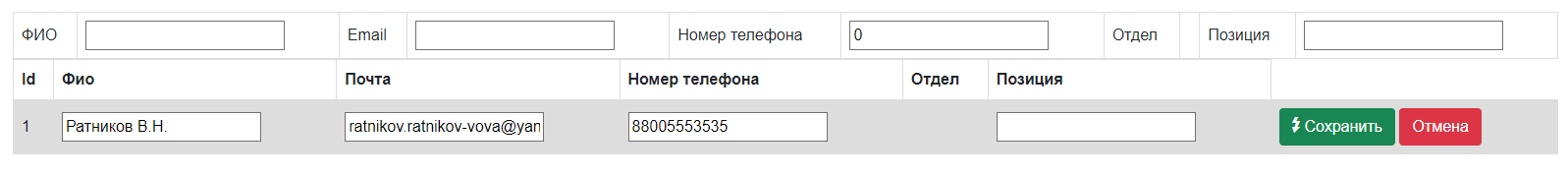


Рисунок 21. Редактирование

* 1. **Удаление сотрудника**

Чтобы удалить сотрудника и перенести его на страницу Deleted, нажмите на кнопку Удалить

* 1. **Восстановление сотрудника**

Чтобы восстановить сотрудника, первоначально на левой панели выберете пункт Deleted, для перехода на новую страницу. Затем для выбранного сотрудника нажмите на кнопку Вернуть

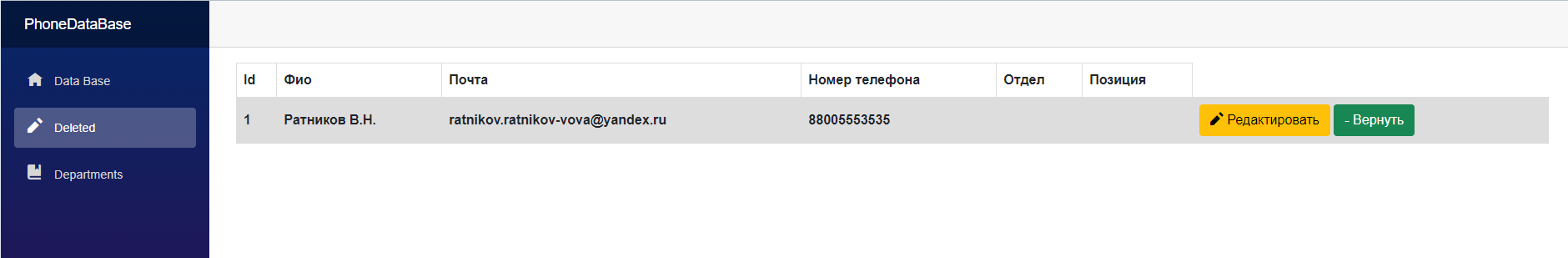


Рисунок 22. Восстановление

* 1. **Добавление нового отдела**

Чтобы добавить новый отдел, перейдите на страницу с отделами, выбрав на левой панели пункт Departments. Введите название отдела и нажмите на плюс

* 1. **Редактирование отдела**

Чтобы редактировать отдел, нажмите на кнопку Редактировать, измените название отдела и нажмите Сохранить.

* 1. **Удаление отдела**

Чтобы удалить отдел, на строке с выбранным отделом нажмите на кнопку Удалить

* 1. **Завершение работы**

Для завершения работы закройте вкладку браузера и запущенный exe файл.

## **To-Do лист**

* Исправление критических багов (реализация смены отдела у сотрудника и проверка на правильность вводимых данных).
* Реализация использование программы без необходимости установки Microsoft Visual Studio.
* Добавление новых параметров для сотрудников(Заработная плата, стаж работы, возраст, дата рождения, график работы и т.д.).
* Добавление поиска и фильтрации.
* Авторизация и добавление прав доступа для разных сотрудников.
* Реализация запуска программы на одном компьютере и её доступе на других компьютерах в локальной сети.
* Доработка уже существующих функций и классов.
* Экспорт и импорт данных в excel файлы.
* Изменение дизайна интерфейса
* Оптимизация программы.

# **Заключение**

Во время выполнения данной курсовой работы была исследована и написана программа PhoneDataBase, для автоматизации учета кадров на производственном предприятии.

При написании программы была исследована среда разработки Microsoft Visual Studio и работа с базой данных SQL. Был изучен язык программирования C#, а также было проведено ознакомление с платформами ASP.NET и Blazor Server, для разработки приложений.

В результате данного исследования был сделан вывод, что процесс написания приложения на новом языке программирования, с использованием баз данных и ранее неизвестными модулями, оказался очень трудоёмким. Подобные программы лучше всего писать группами из нескольких человек, с распределением обязанностей и областей программы.

# **Список использованной литературы и интернет-ресурсов**

1. PhoneDataBase — Приложения для учета сотрудников производственного предприятия

<https://github.com/BlackNetik/PhoneDataBase>

1. Что входит в обязанности начальника отдела кадров: перечень:

<http://buhguru.com/kadrovaya-rabota/chto-vhodit-v-obyazannosti-nachalnika-otdela-kadrov-perechen.html>

1. Программы автоматизации ведения кадрового учета и делопроизводства на предприятии:

<https://soft.mydiv.net/win/cname85/buchgalteriya/kadrovyy-uchet/>

1. Сотрудники предприятия

<https://araxgroup.ru/index.php/gallery-categories/second-gallery/28-sotrudniki-predpriyatiya#1>

1. Руководство по ASP.NET Core 6

<https://metanit.com/sharp/aspnet6/>

1. Руководство по фреймворку Blazor

<https://metanit.com/sharp/blazor/>

# **Приложение 1. Интерфейс IUserService**

public interface IUserService

{

Task<List<User>> GetUsers();

Task<List<User>> GetDeletedUsers();

Task<List<Department>> GetDepartments();

void Insert(User user);

void InsertDepartment(Department depp);

void Delete(User user);

void DeleteDepartment(Department depp);

void Update(User user);

void UpdateDepartment(Department depp);

void Back(User user);

void BackDepartment(Department depp);

}

# **Приложение 2. Функции Counter.razor**

protected override async Task OnInitializedAsync() @\*Инициализация\*@

{

DeletedUsers = await UserServices.GetDeletedUsers();

Listdepartments = await UserServices.GetDepartments();

}

public void BackUser(User user) @\*Вернуть удалённого пользователя\*@

{

UserServices.Back(user);

}

public void SaveUser(User user) @\*Сохранить данные после редактирования\*@

{

user.Fio = EditFio;

user.Email = EditEmail;

user.NumberPhone = EditNumberPhone;

user.DepartmentPosition = EditDepartmentPosition;

user.StatusId = (int)Statuses.Deleted;

UserServices.Update(user);

}

public void EditUser(User user) @\*Редактировать удаленного пользователя\*@

{

DeletedUsers.ForEach(t => t.StatusId = (int)Statuses.Deleted);

user.StatusId = (int)Statuses.Edit;

EditFio = user.Fio;

EditEmail = user.Email;

EditNumberPhone = user.NumberPhone;

EditDepartmentPosition = user.DepartmentPosition;

}

public void CancelUser(User user) @\*Отменить редактирование\*@

{

user.StatusId = (int)Statuses.Deleted;

}