|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ | | | | |
| НАБЕРЕЖНОЧЕЛНИНСКИЙ ИНСТИТУТ (ФИЛИАЛ)  ФЕДЕРАЛЬНОГО ГОСУДАРСТВЕННОГО АВТОНОМНОГО ОБРАЗОВАТЕЛЬНОГО УЧРЕЖДЕНИЯ ВЫСШЕГО ОБРАЗОВАНИЯ  «КАЗАНСКИЙ (ПРИВОЛЖСКИЙ) ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ» | | | | |
| КАФЕДРА «ИНФОРМАЦИОННЫЕ СИСТЕМЫ» | | | | |
| Направление подготовки 09.03.04  «Программная инженерия» | | | | |
|  | | | | |
|  | | | Утверждаю | |
|  | | | Заведующий кафедрой ИС | |
|  | | | | |
| \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Р.А.Валиев | | | | |
|  | | | | |
| \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_г. | | | | |
|  | | | | |
|  | | | | |
| **КУРСОВАЯ РАБОТА** | | | | |
| по дисциплине: | | | | |
| **«Программирование»** | | | | |
| на тему: | | | | |
| **«Разработка прикладной программы с использованием объектно-ориентированной технологии»** | | | | |
|  | | | | |
|  | | | | |
|  | | | | |
|  | | | | |
| Автор: | | Оценка: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | | |
| студент группы 2221121 | |  | | |
|  | | Руководитель: | | |
| \_\_\_\_\_\_\_\_\_\_\_\_\_\_ | А.А.Хайрутдинов | к.т.н., доцент кафедры ИС | | |
|  |  |  | |  |
|  |  | \_\_\_\_\_\_\_\_\_\_\_\_\_\_ | | Е.В. Зубков |
|  |  |  | | |
|  |  | Дата защиты:  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_г. | | |
|  | | | | |
|  | | | | |
|  | | | | |
|  | | | | |
|  | | | | |
|  | | | | |
| Набережные Челны | | | | |
| 2024 | | | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ | | | | |
| НАБЕРЕЖНОЧЕЛНИНСКИЙ ИНСТИТУТ (ФИЛИАЛ)  ФЕДЕРАЛЬНОГО ГОСУДАРСТВЕННОГО АВТОНОМНОГО ОБРАЗОВАТЕЛЬНОГО УЧРЕЖДЕНИЯ ВЫСШЕГО ОБРАЗОВАНИЯ  «КАЗАНСКИЙ (ПРИВОЛЖСКИЙ) ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ» | | | | |
| КАФЕДРА «ИНФОРМАЦИОННЫЕ СИСТЕМЫ» | | | | |
| Направление подготовки 09.03.04  «Программная инженерия» | | | | |
|  | | | | |
|  | | | Утверждаю | |
|  | | | Заведующий кафедрой ИС | |
|  | | | | |
| \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Р.А.Валиев | | | | |
|  | | | | |
| \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_г. | | | | |
|  | | | | |
| **ЗАДАНИЕ НА КУРСОВУЮ РАБОТУ** | | | | |
|  | | | | |
| Студент | | | | |
| Хайрутдинов Аяз Алмазович | | | | |
| 1 Тема | | | | |
| «Разработка прикладной программы с использованием объектно-ориентированной технологии» | | | | |
| 2 Срок представления к защите | | | | |
| \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_г. | | | | |
| 3 Исходные данные  - информация по работе с ASP.NET Core ;  - информация по работе с Entity Framework;  - информация о простых функциях и константах применяемых при вычислениях | | | | |
|  | | | | |
| 4 Перечень подлежащих разработке вопросов | | | | |
| - анализ предметной области; ­  -проектирование системы с помощью методологи UML | | | | |
|  | | | | |
| Задание выдано | г. | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | | Е.В. Зубков |
|  | | | | |
| Задание принято | \_\_\_\_\_\_\_\_\_\_\_\_\_г. | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | | А.А.Хайрутдинов |

Содержание

[1. Проектирование программного продукта 4](#_Toc166063343)

[1.1 UML диаграмма Прецедентов 4](#_Toc166063344)

[1.2 UML диаграмма Классов 5](#_Toc166063345)

[1.3 UML диаграмма Состояний 7](#_Toc166063346)

[1.4 UML диаграмма Последовательностей 8](#_Toc166063347)

[2. Листинг кода прикладной программы(КОД) 9](#_Toc166063348)

[2.1 EmploeeController 9](#_Toc166063349)

[2.1 Emploee 13](#_Toc166063350)

[2.2 Human 13](#_Toc166063351)

[2.3 SortHeaderTagHelper 13](#_Toc166063352)

[2.4 ApplicationContext 15](#_Toc166063353)

[3. Результат выполнения программы 16](#_Toc166063354)

[4. Заключение 22](#_Toc166063355)

# **1. Проектирование программного продукта**

## **1.1 UML диаграмма Прецедентов**

Данная диаграмма, изображенная на рис.1.1, описывает возможные сценарии работы пользователя (актера) с приложением на ASP.Net Core. В качестве овалов (прецедентов) выступают различные действия, доступные пользователю, такие как "Создание пользователя", "Удаление пользователя", "Редактирование пользователя" и другие. Связь между прецедентами и пользователем осуществлена пунктирными линиями (зависимость), которые показывают, что все представленные прецеденты зависят от действий пользователя и возвращают ему результат своей работы.

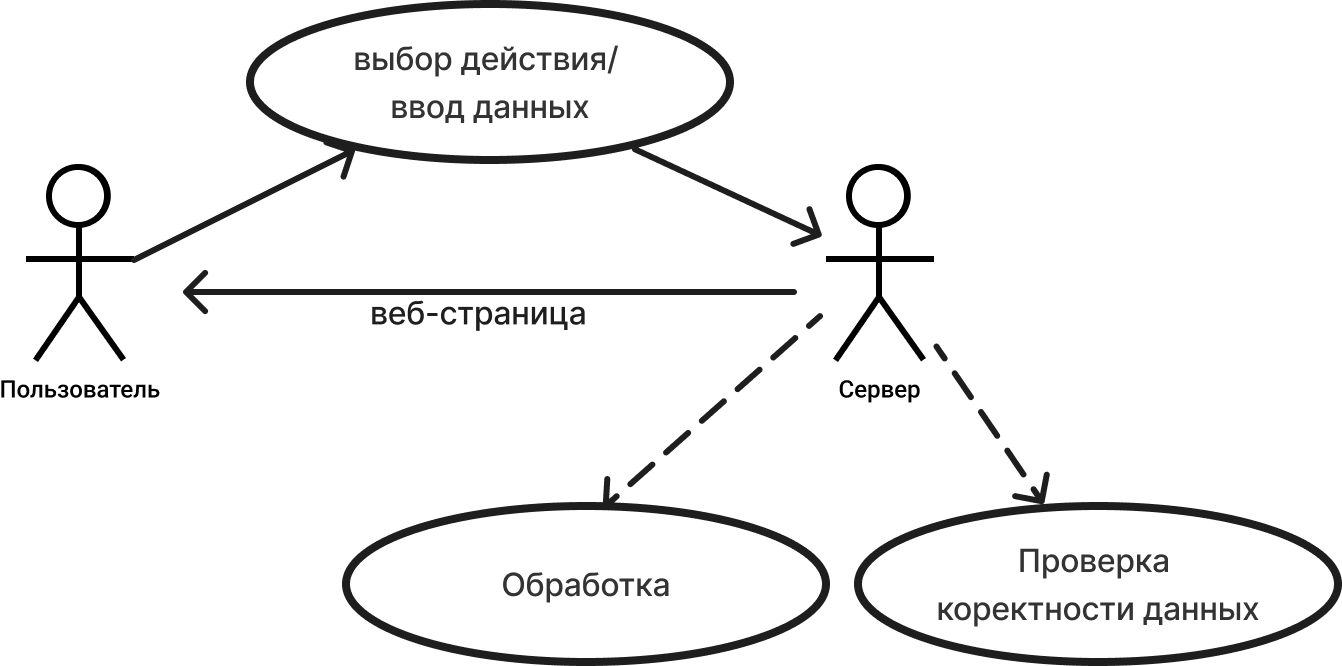


Рисунок 1.1 UML диаграмма Прецедентов

## **1.2 UML диаграмма Классов**

Данный вид диаграммы, изображенная на рис.1.2, определяет типы объектов системы и различного рода связи, которые существуют между ними.

Программа представляет собой веб-приложение для управления информацией о сотрудниках в компании. Основной класс КонтроллерСотрудника обрабатывает HTTP-запросы и управляет логикой приложения. Взаимодействие с базой данных осуществляется через класс КонтекстБдПриложения который представляет собой контекст базы данных Entity Framework.

Каждый сотрудник представлен объектом классаСотрудник, который содержит информацию о его персональных данных, таких как имя, фамилия, адрес электронной почты, дата рождения, дата приема на работу и должность.

Взаимодействие между методами контроллера КонтроллерСотрудника и представлениями (Views) осуществляется через передачу моделей данных. Например, метод Создать отображает представление для создания нового сотрудника, а метод ПолучитьВсехСотрудников осуществляет поиск и фильтрацию списка сотрудников и передает результаты в представление.

Класс IndexViewModel представляет модель представления для отображения списка сотрудников на странице. Он также содержит информацию о параметрах фильтрации и сортировки.

Классы SortState и SortViewModel отвечают за управление сортировкой списка сотрудников на странице. Они определяют возможные варианты сортировки и текущее состояние сортировки.

Класс SortHeaderTagHelper представляет собой пользовательский тег-помощник ASP.NET Core, который используется для создания ссылок на сортировку списка сотрудников на веб-странице. Он реализует интерфейс TagHelper, что позволяет ему взаимодействовать с HTML-элементами на странице и модифицировать их в процессе отрисовки

Программа состоит из нескольких взаимосвязанных классов, которые обеспечивают функциональность для управления информацией о сотрудниках и взаимодействия с пользовательским интерфейсом.

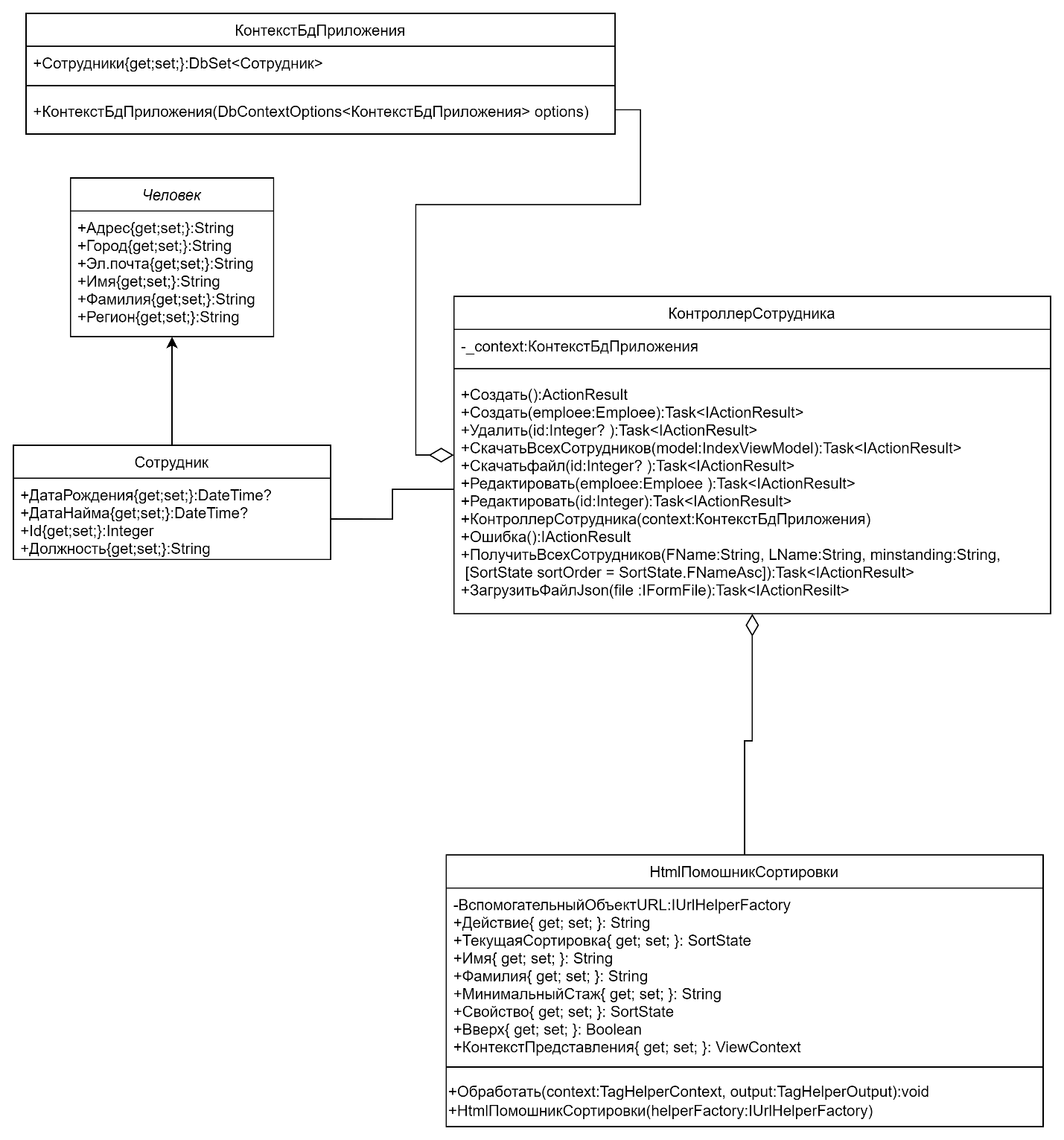


Рисунок 1.2 UML диаграмма Классов

## **1.3 UML диаграмма Состояний**

Диаграмма Состояний, изображенная на рис.1.3, отражает динамический алгоритм работы прикладной программы в зависимости от условий. Вход в программу осуществляется с помощью черного закрашенного круга, а выход- с помощью такого же закрашенного круга, на фоне с белым внутренним ободом. В качестве прямоугольника с закругленными краями выступает поведение нашего главного объекта(формы), и в зависимости от направления стрелки, изменяется и состояние объекта.

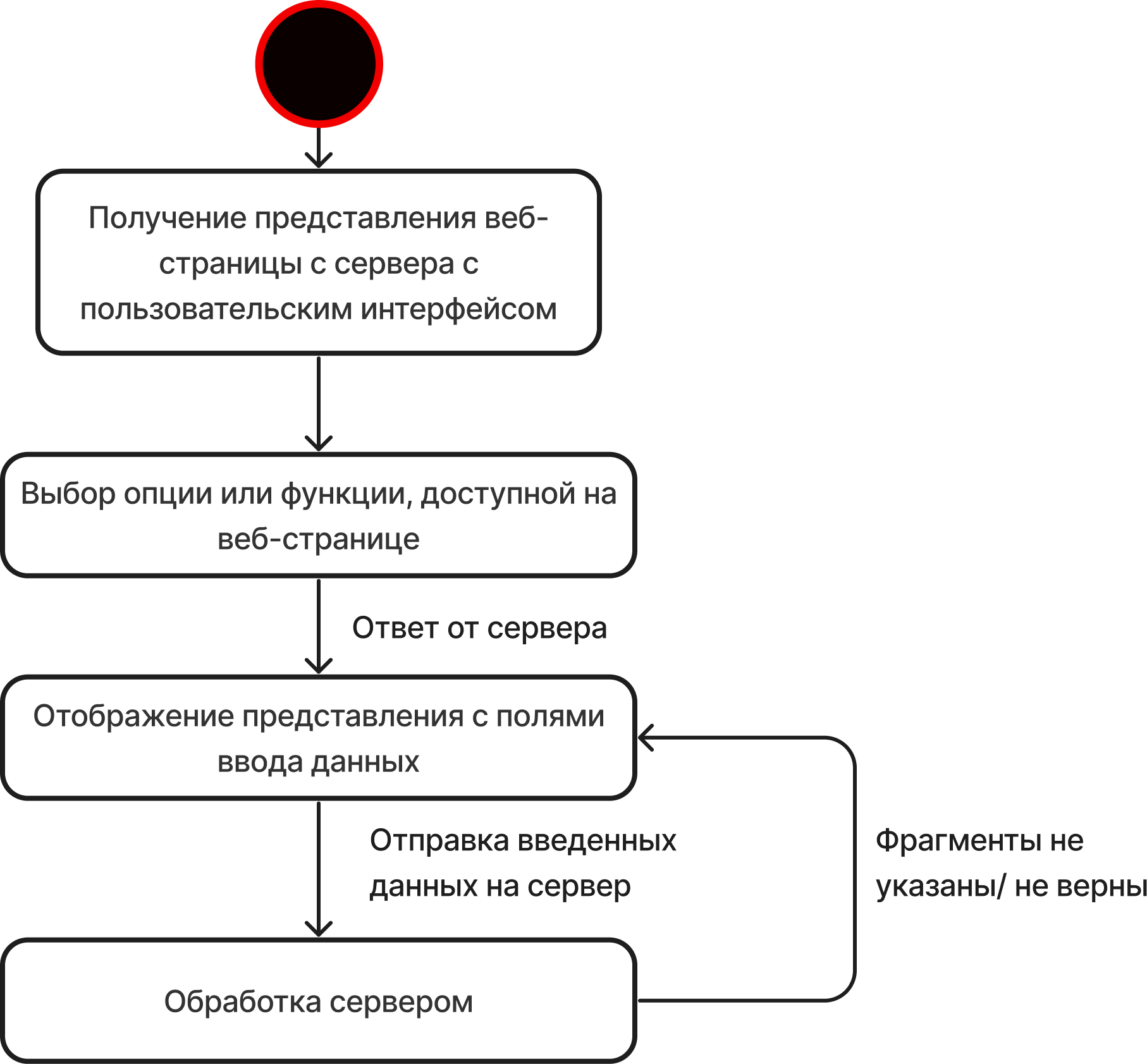


Рисунок 1.3 UML диаграмма Состояний

## **1.4 UML диаграмма Последовательностей**

Диаграмма Последовательностей, изображенная на рис.1.4, служит для описания более детального взаимодействия частей системы, нежели диаграмма Прецедентов. В верхней части диаграммы изображены “действующие лица” программы. Стрелка слеванаправо отражают очередность вызова какого-то действия со стороны пользователя и получения сообщения от исполняющего эту функцию метода. Горизонтальный прямоугольник под “действующим лицом” программы показывает его время действия в контексте выполнения всей системы.

Вся работа приложения начинается с запуска приложения пользователем, а все взаимодействия с функционалом программы производятся внутри системы исходя из запросов пользователя. Тем самым программа получив сигнал переходит уже к выполнению определенных действий, основанных на готовых алгоритмах самой программы.



Рисунок 1.4 UML диаграмма Последовательностей

# **2. Листинг кода прикладной программы(КОД)**

## **2.1 EmploeeController**

public class EmploeeController : Controller

{

private readonly ApplicationContext \_context;

public EmploeeController(ApplicationContext context)

{

\_context = context;

}

[ResponseCache(Duration = 0, Location = ResponseCacheLocation.None, NoStore = true)]

public IActionResult Error()

{

return View(new ErrorViewModel { RequestId = Activity.Current?.Id ?? HttpContext.TraceIdentifier });

}

public ActionResult Create()//Создание пользователья

{

return View();

}

[HttpPost]

public async Task<IActionResult> Create(Emploee emploee)

{

\_context.Emploees.Add(emploee);

await \_context.SaveChangesAsync();

TempData["SuccessMessage"] = "Работник успешно добавлен";

return RedirectToAction("Create");

}

[HttpGet]

public async Task<IActionResult> GetEmployees(string? FName, string? LName, string minstanding, SortState sortOrder = SortState.FNameAsc)

{

// Сохранение переданных параметров в TempData для сохранения их состояния между запросами

TempData["FName"] = FName;

TempData["LName"] = LName;

TempData["minstanding"] = minstanding;

// Получение IQueryable объектов Emploee из контекста базы данных

IQueryable<Emploee> users = \_context.Emploees;

// Рассчитывается дата стажа, отсчитываемая от текущей даты

DateTime standingDate = DateTime.Now.AddYears(Convert.ToInt32(minstanding) \* (-1));

// Фильтрация по имени

if (!string.IsNullOrWhiteSpace(FName))

{

users = users.Where(s => s.FName.Contains(FName));

}

// Фильтрация по фамилии

if (!string.IsNullOrWhiteSpace(LName))

{

users = users.Where(s => s.LName.Contains(LName));

}

// Фильтрация по стажу

if (minstanding != "0" || minstanding != null)

{

users = users.Where(s => s.DateOfHire < standingDate);

}

// Применение сортировки

users = sortOrder switch

{

SortState.FNameDesc => users.OrderByDescending(s => s.FName),

SortState.FNameAsc => users.OrderBy(s => s.FName),

SortState.LNameAsc => users.OrderBy(s => s.LName),

SortState.LNameDesc => users.OrderByDescending(s => s.LName),

SortState.EmailAsc => users.OrderBy(s => s.Email),

SortState.EmailDesc => users.OrderByDescending(s => s.Email),

SortState.DateOfHireAsc => users.OrderBy(s => s.DateOfHire),

SortState.DateOfHireDesc => users.OrderByDescending(s => s.DateOfHire),

SortState.DateOfBirthAsc => users.OrderBy(s => s.DateOfBirth),

SortState.DateOfBirthDesc => users.OrderByDescending(s => s.DateOfBirth),

SortState.PositionAsc => users.OrderBy(s => s.Position),

SortState.PositionDesc => users.OrderByDescending(s => s.Position),

SortState.AddressAsc => users.OrderBy(s => s.Address),

SortState.AddressDesc => users.OrderByDescending(s => s.Address),

SortState.CityAsc => users.OrderBy(s => s.City),

SortState.CityDesc => users.OrderByDescending(s => s.City),

SortState.RegionAsc => users.OrderBy(s => s.Region),

SortState.RegionDesc => users.OrderByDescending(s => s.Region),

\_ => users.OrderBy(s => s.FName), // По умолчанию сортировка по имени в порядке возрастания

};

// Создание ViewModel, которая будет передана в представление

IndexViewModel viewModel = new IndexViewModel

{

Emploees = await users.AsNoTracking().ToListAsync(), // Загрузка пользователей в список с учетом сортировки

SortViewModel = new SortViewModel(sortOrder) // Создание модели сортировки для передачи в представление

};

Console.WriteLine(sortOrder);

// Возвращение представления с ViewModel

return View(viewModel);

}

[HttpPost]

public async Task<IActionResult> Delete(int? id)

{

// Проверка, что передан идентификатор сотрудника

if (id != null)

{

// Создание нового объекта Emploee с указанным идентификатором

Emploee emploee = new Emploee { Id = id.Value };

// Установка состояния объекта в EntityState.Deleted,

// это говорит Entity Framework, что этот объект должен быть удален из базы данных

\_context.Entry(emploee).State = EntityState.Deleted;

// Сохранение изменений в базе данных

await \_context.SaveChangesAsync();

// Перенаправление на метод GetEmployees после успешного удаления

return RedirectToAction("GetEmployees");

}

// В случае отсутствия идентификатора, возвращается NotFound

return NotFound();

}

public async Task<IActionResult> Edit(int? id)

{

if (id != null)

{

Emploee? emploee = await \_context.Emploees.FirstOrDefaultAsync(p => p.Id == id);

if (emploee != null) return View(emploee);

}

return NotFound();

}

[HttpPost]

public async Task<IActionResult> Edit(Emploee emploee)

{

\_context.Emploees.Update(emploee);

await \_context.SaveChangesAsync();

return RedirectToAction("GetEmployees");

}

[HttpPost]

public async Task<IActionResult> UploadJsonFile(IFormFile file)

{

try

{

if (file == null || file.Length == 0)

{

@TempData["NoFileMessage"] = "Пожалуйста выберите файл";

return RedirectToAction("GetEmployees");

}

using (var streamReader = new StreamReader(file.OpenReadStream()))

{

var jsonString = await streamReader.ReadToEndAsync();

// Десериализация JSON

var employees = JsonSerializer.Deserialize<List<Emploee>>(jsonString);

if (employees != null && employees.Any())

{

// Устанавливаем Id в 0 для каждого объекта

foreach (var employee in employees)

{

employee.Id = 0;

}

// Добавление данных в базу данных

\_context.Emploees.AddRange(employees);

await \_context.SaveChangesAsync();

TempData["SuccessMessage"] = "Успешно добавлено";

return RedirectToAction("GetEmployees");

//return Ok("Данные успешно добавлены в базу данных.");

}

else

{

@TempData["ErrorMessage"] = "Ошибка";

return RedirectToAction("GetEmployees");

}

}

}

catch (Exception ex)

{

@TempData["ErrorMessage"] = "Ошибка";

return RedirectToAction("GetEmployees");

}

}

public async Task<IActionResult> DownloadFile(int? id)

{

try

{

if (id != null)

{

Emploee? emploee = await \_context.Emploees.FirstOrDefaultAsync(p => p.Id == id);

if (emploee != null)

{

// Сериализация объекта в JSON

string jsonString = JsonSerializer.Serialize(new[] { emploee });

// Создаем уникальное имя файла

string fileName = $"employee\_{emploee.FName}\_{emploee.LName}\_{DateTime.Now:yyyyMMddHHmmss}.json";

//// Записываем JSON в массив байтов

//byte[] fileContents = Encoding.UTF8.GetBytes(jsonString);

// Возвращаем файловый поток клиенту

return File(Encoding.UTF8.GetBytes(jsonString), "application/json", fileName);

}

}

return NotFound("Employee not found with the provided id.");

}

catch (Exception ex)

{

// Обработка ошибок, если необходимо

return StatusCode(500, $"Internal Server Error: {ex.Message}");

}

}

[HttpPost]

public async Task<IActionResult> DownloadAllEmploees([FromForm] IndexViewModel model)

{

try

{

// Получаем массив id из модели

int[] employeeIds = model.EmployeeIds;

// Получаем объекты Emploee из базы данных

var employees = await \_context.Emploees

.Where(e => employeeIds.Contains(e.Id))

.ToListAsync();

if (employees.Any())

{

// Сериализуем объекты Emploee в JSON

string jsonString = JsonSerializer.Serialize(employees);

// Получаем уникальное имя файла

string fileName = $"all\_employees\_forusers{DateTime.Now:yyyyMMddHHmmss}.json";

// Составляем путь для сохранения файла на стороне клиента

string filePath = Path.Combine("Saves", fileName);

// Возвращаем файл для скачивания на стороне клиента

return File(Encoding.UTF8.GetBytes(jsonString), "application/json", fileName);

}

return NotFound("No employees found with the provided ids.");

}

catch (Exception ex)

{

// Обработка ошибок, если необходимо

return StatusCode(500, $"Internal Server Error: {ex.Message}");

}

}

## **2.1 Emploee**

public class Emploee:Human

{

public int Id { get; set; }

public DateTime? DateOfHire { get; set; }

public DateTime? DateOfBirth { get; set; }

public string Position { get; set; }

}

## **2.2 Human**

public abstract class Human

{

public string FName { get; set; }

public string LName { get; set; }

public string Email { get; set; }

public string Address { get; set; }

public string City { get; set; }

public string Region { get; set; }

}

## **2.3 SortHeaderTagHelper**

public class SortHeaderTagHelper : TagHelper

{

public string FName { get; set; }

public string LName { get; set; }

public string minstanding { get; set; }

public SortState Property { get; set; } // значение текущего свойства, для которого создается тег

public SortState Current { get; set; } // значение активного свойства, выбранного для сортировки

public string? Action { get; set; } // действие контроллера, на которое создается ссылка

public bool Up { get; set; } // сортировка по возрастанию или убыванию

[ViewContext]

[HtmlAttributeNotBound]

public ViewContext ViewContext { get; set; } = null!;

IUrlHelperFactory urlHelperFactory;

public SortHeaderTagHelper(IUrlHelperFactory helperFactory)

{

urlHelperFactory = helperFactory;

}

public override void Process(TagHelperContext context, TagHelperOutput output)

{

IUrlHelper urlHelper = urlHelperFactory.GetUrlHelper(ViewContext);

output.TagName = "a";

//string? url = urlHelper.Action(Action, new { sortOrder = Property });

var routeValues = new

{

FName,

LName,

minstanding,

sortOrder = Property

};

string url = urlHelper.Action(Action, routeValues);

output.Attributes.SetAttribute("href", url);

// если текущее свойство имеет значение CurrentSort

if (Current == Property)

{

string arrowSymbol = Up ? "▲" : "▼";

output.PreContent.Append(arrowSymbol);

}

}

}

## **2.4 ApplicationContext**

public class ApplicationContext : DbContext

{

public DbSet<Emploee> Emploees { get; set; } = null!;

public ApplicationContext(DbContextOptions<ApplicationContext> options)

: base(options)

{

Database.EnsureCreated(); // создаем базу данных при первом обращении

}

}

# **3. Результат выполнения программы**

При запуске веб-приложения перед нами открывается главная страница, изображенная на рисунке 3.1 на которой находятся:

-Кнопка «Все сотрудники» и «Добавить сотрудника» в шапке страницы.

-Таблица всех сотрудников

-Поля фильтрации таблицы

-Кнопки с названиями колонок таблицы, отвечающие за сортировки

-Кнопки с добавлением сотрудника/сотрудников из файла

-Кнопка «Добавить»

-Кнопка для сохранения всех сотрудников

-Кнопки «Изменить», «Сохранить» и «Удалить» в строке каждого сотрудника таблицы

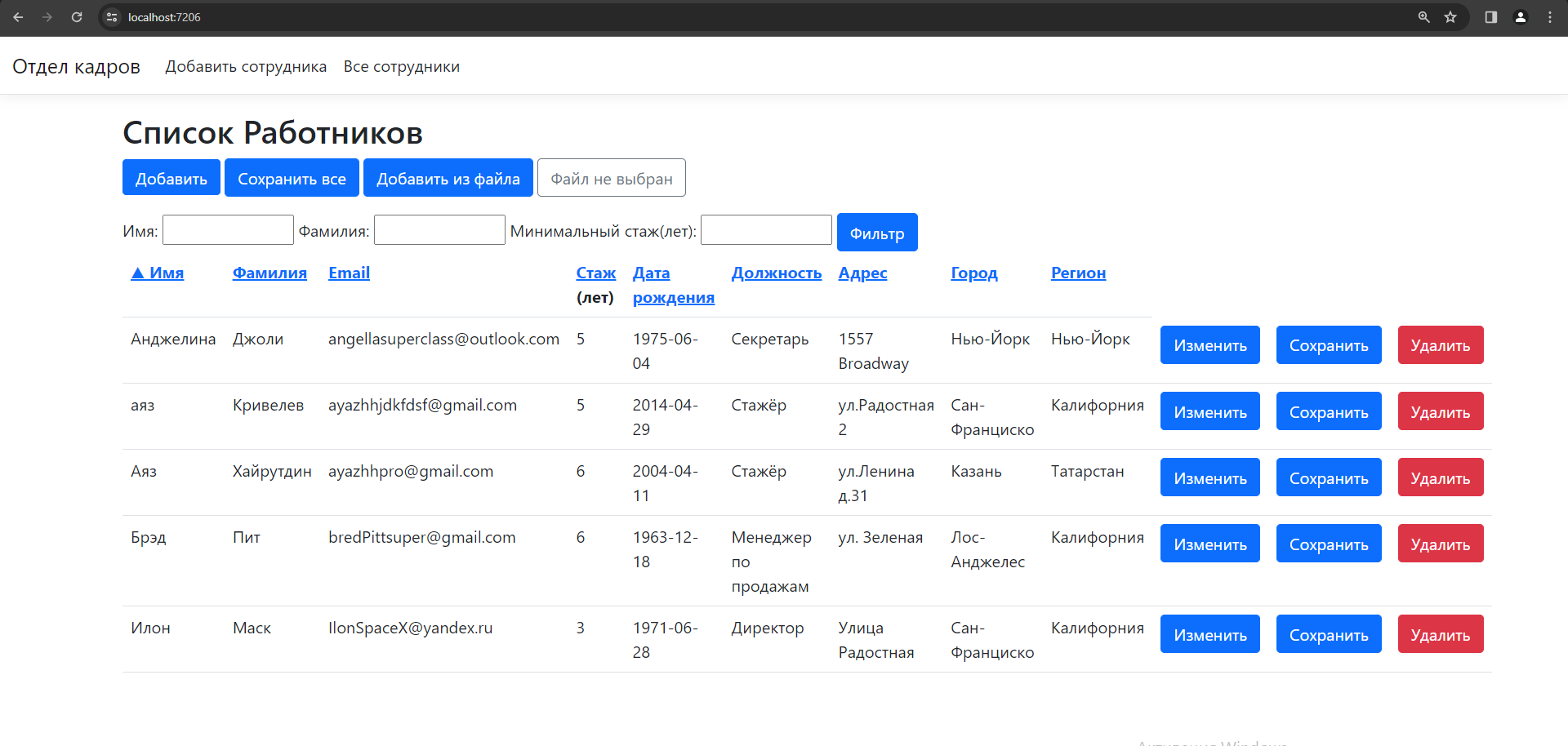


Рисунок 3.1 - главная страница.

При нажатии на названия колонок («Имя», «Фамилия», «Email», «Стаж», «Дата Рождения», «Должность», «Адрес», «Город», «Регион»), таблица будет автоматически сортироваться по возрастанию или по убыванию в зависимости от выбранной сортировки. Направление текущей сортировки будет указано стрелкой рядом с заголовком колонки. Для полей строкового типа сортировка будет происходить по алфавиту. Пример указан на рис. 3.2 и рис. 3.3.

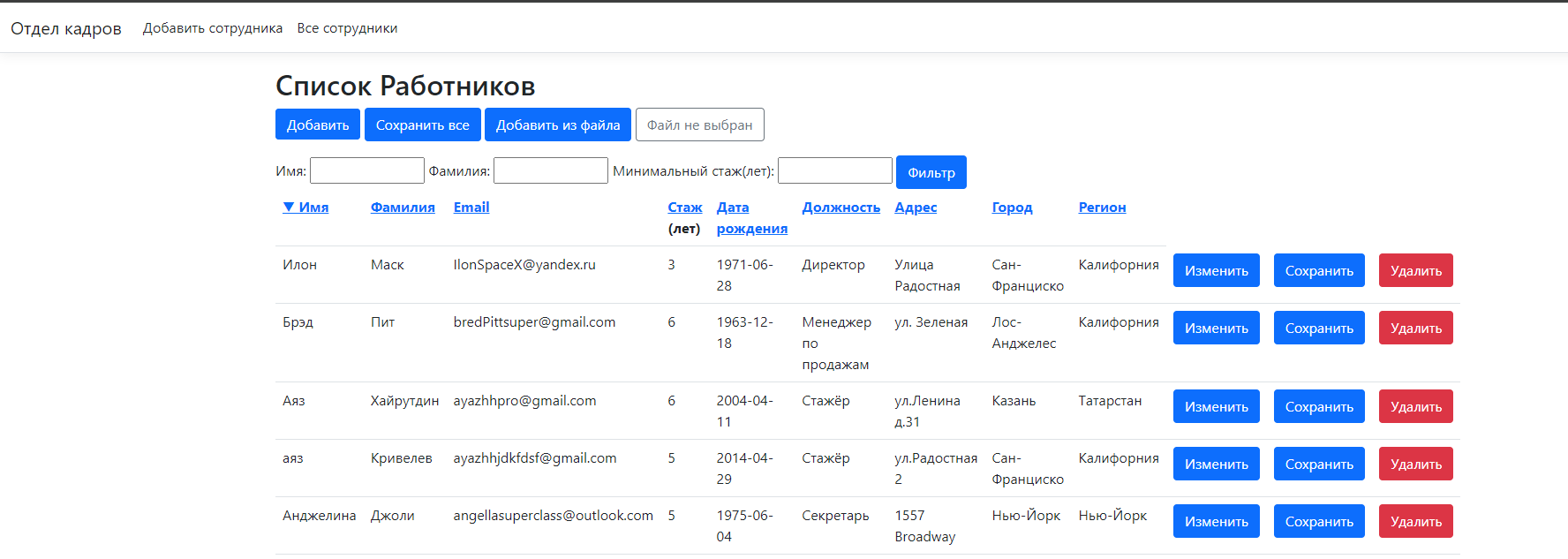


Рисунок 3.2 – список работников.

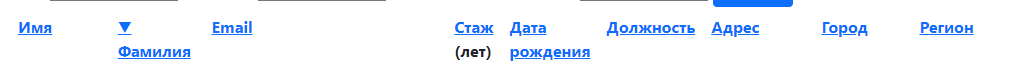


Рисунок 3.3 – поля сортировок.

При нажатии кнопки «Сохранить все» в загрузки вашего браузера добавится .json файл с информацией о всех сотрудниках. Это показано на рис. 3.4

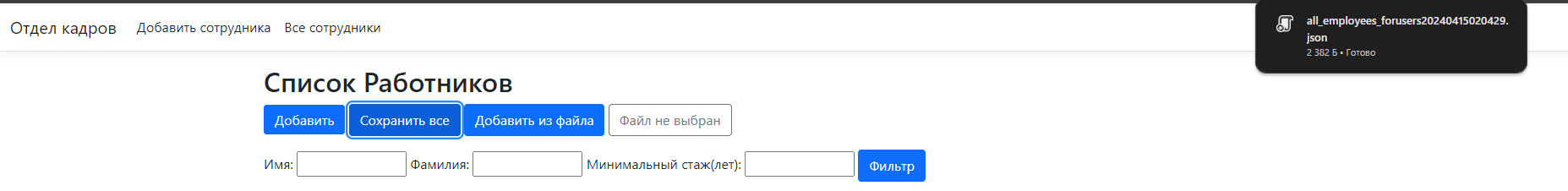


Рисунок 3.4 – кнопка «Сохранить все».

На рис. 3.5 показано как выглядит файл .json, если открыть его в специальном текстовом редакторе

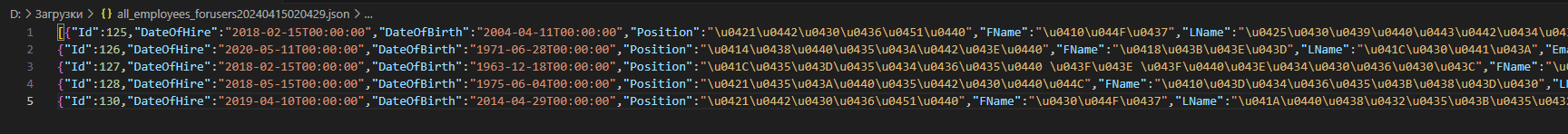


Рисунок 3.5 – файл .json

При нажатии кнопки «Удалить» информация о данном человеке будет стерта из Базы данных. При нажатии кнопки «Сохранить», находящиеся напротив информации о каждом сотруднике, изображенной на рис. 3.6. В загрузки вашего браузера добавится .json файл с информацией только о выбранном сотруднике.

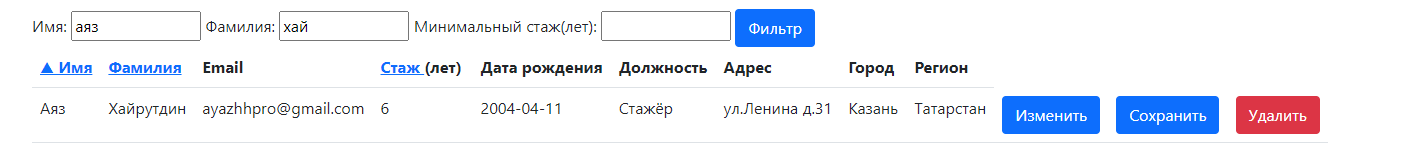


Рисунок 3.6 – строка с информацией о сотруднике

При нажатии кнопки «Изменить» нас перекинет на веб-страницу с полями данных о сотруднике, доступных для редактирования, изображено на рис. 3.7. После внесения изменений, нужно нажать кнопку «Сохранить», тогда изменения вступят в силу и нас перебросит обратно на главную страницу.

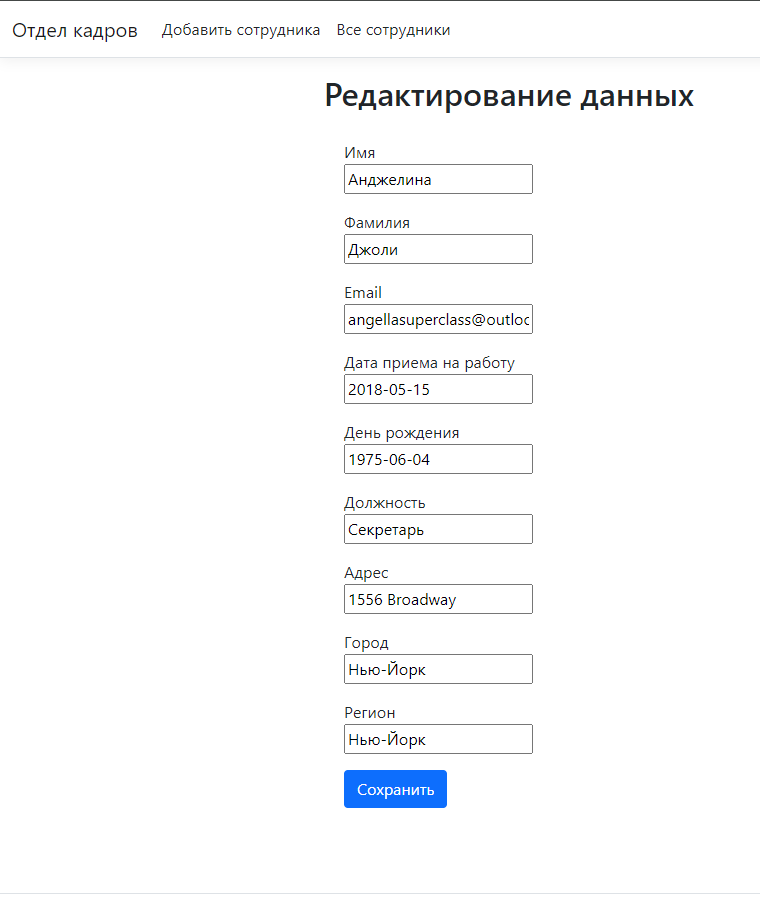


Рисунок 3.7 – Окно редактирование данных сотрудника

При вводе имени, фамилии или минимального стажа сотрудника, а затем нажатии на кнопку "Фильтр", произойдет фильтрация таблицы. Будут отображены только те записи, которые соответствуют введенным данным. Эта функция также может использоваться для поиска необходимой информации. На рис. 3.8 указан фильтр который отобразит только тех сотрудников, чье имя начинается на «Аяз» и фамилия начинается на «хай».

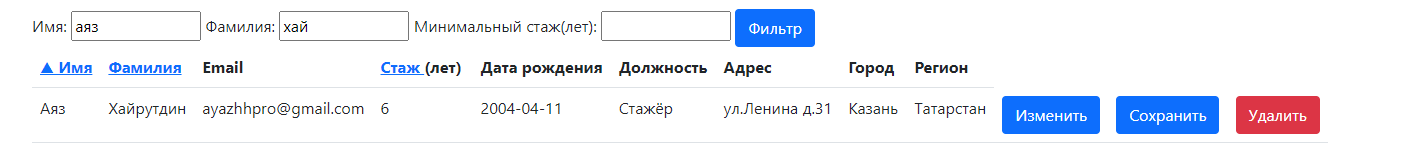


Рисунок 3.8 – Использование фильтрации.

Также можно отфильтровать только по одному полю. Например на рис. 3.9 указан фильтр который отобразит всех сотрудников, чье имя начинается на «аяз». На рис. 3.10 Отображаются только, те чей стаж работы больше четырех лет.

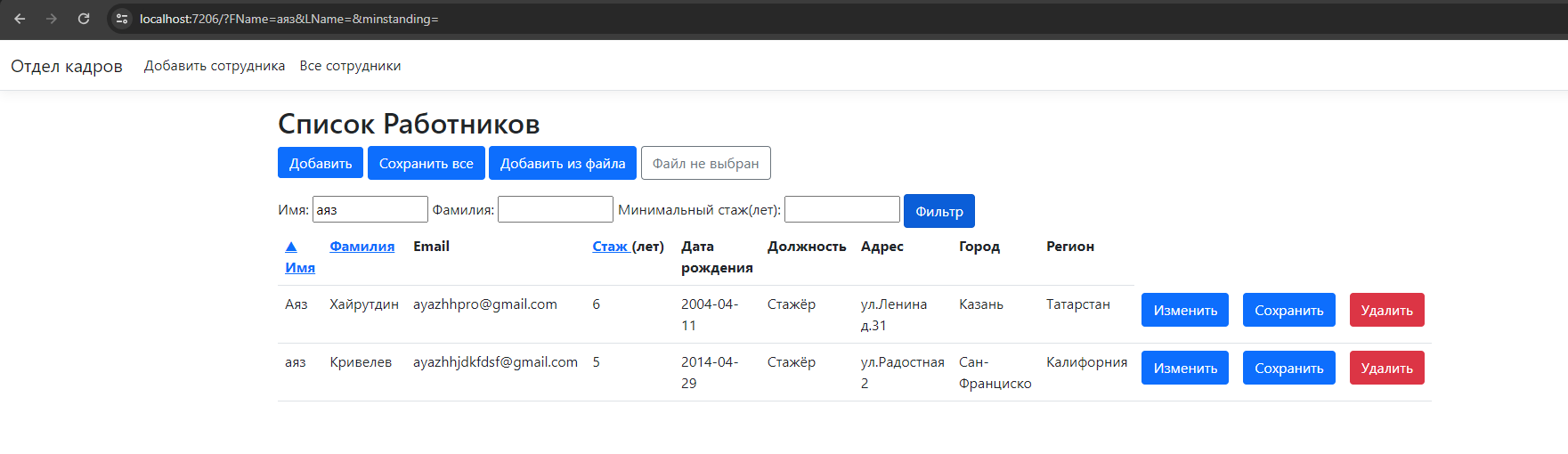


Рисунок 3.9 – Отображение отфильтрованных данных по имени.

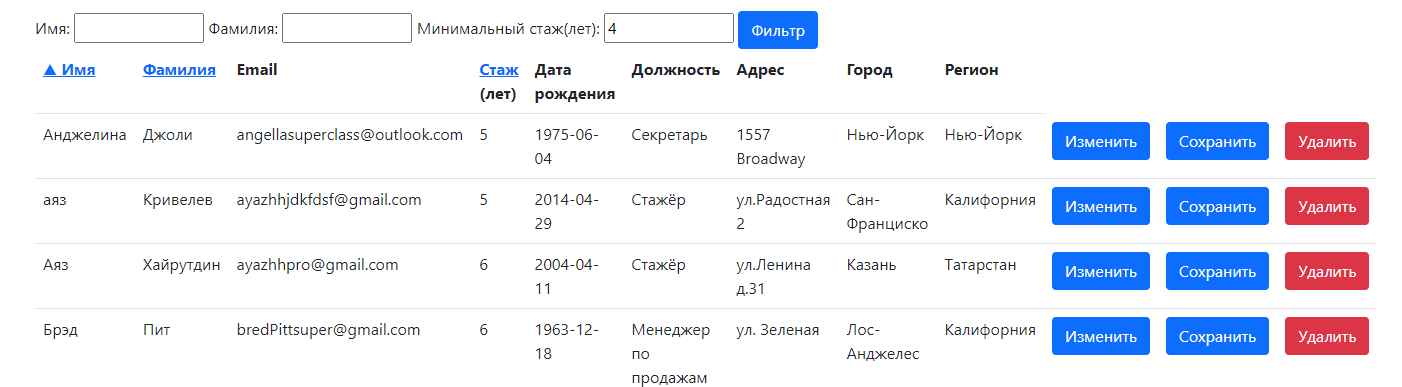


Рисунок 3.10 – Отображение отфильтрованных данных по минимальному стажу работы.

При нажатии на кнопку «Добавить» или на кнопку «Добавить сотрудника», находящуюся в шапке страницы, нас перекинет на веб-страницу с полями для ввода данных нового сотрудника, это показано на рис. 3.11

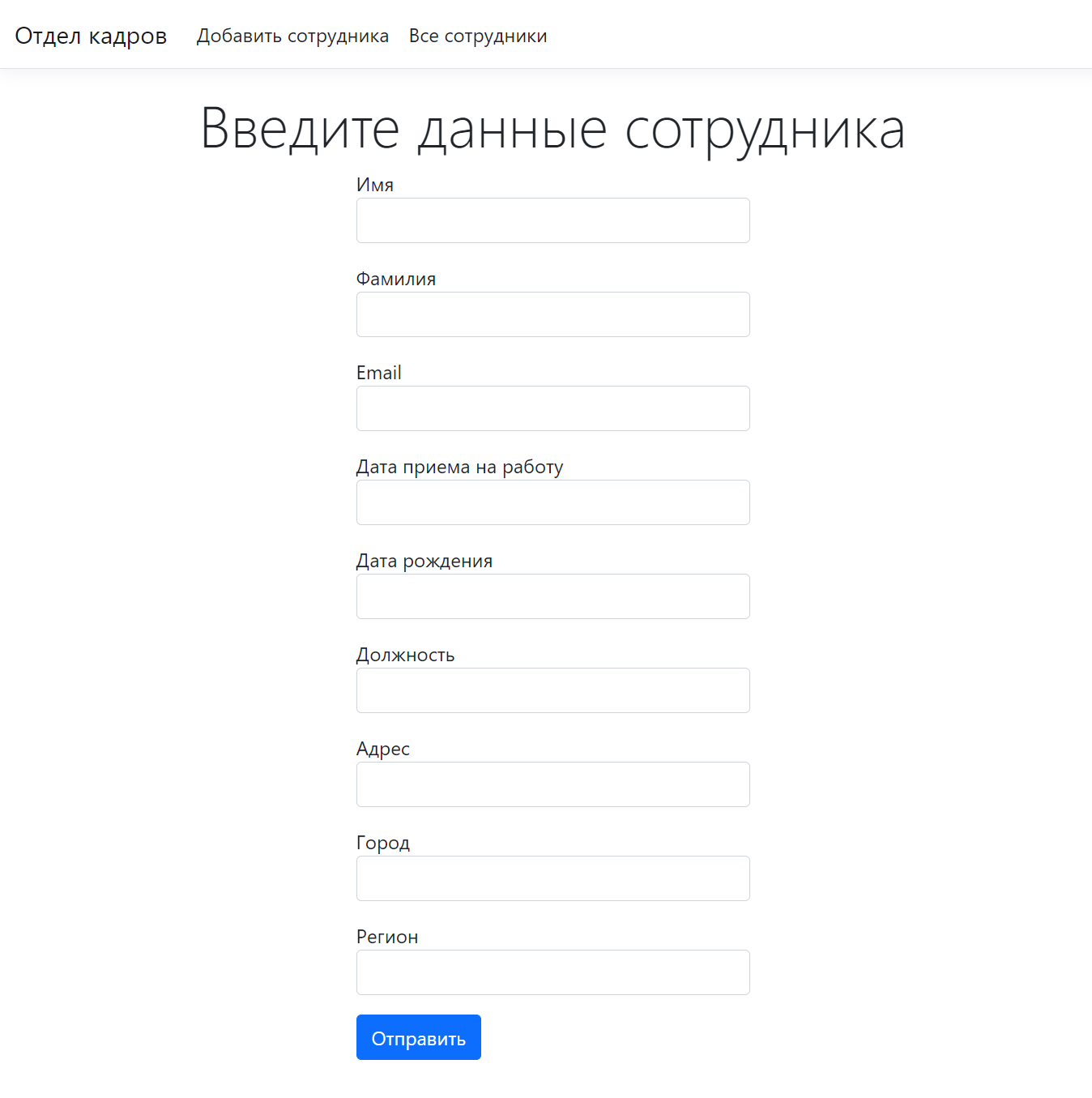


Рисунок 3.11 – Страница добавления сотрудника.

В случае успешного добавления сотрудника система выведет окно, указанное на рисунке 3.12



Рисунок 3.11 – Окно успешного добавления сотрудника.

Чтобы добавить нового сотрудника или группу сотрудников через файл, нажмите кнопку «Выбрать файл», как показано на рис. 3.13. После этого откроется окно выбора файла на вашем компьютере, ищите файл в формате .json. По умолчанию кнопка будет отображаться в сером цвете. После успешного выбора файла, кнопка станет зеленой и отобразит надпись «Файл выбран», как показано на рис. 3.14. Далее нажмите кнопку «Добавить из файла», чтобы загрузить данные из файла в систему. В случае, если файл не выбран система выведет красное окно с информацией об этом, изображено на рис. 3.15.

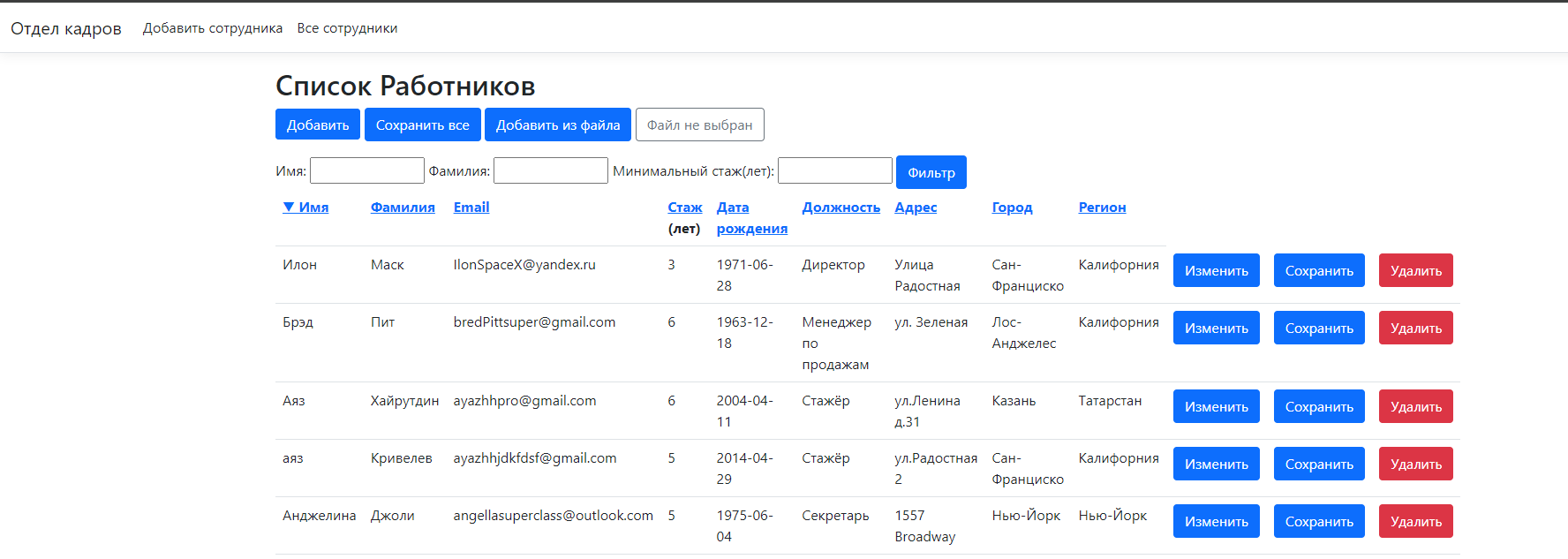


Рисунок 3.13 – Кнопка выбора файла.

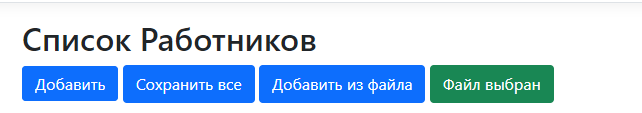


Рисунок 3.14 – Зеленая кнопка с выбранным файлом.

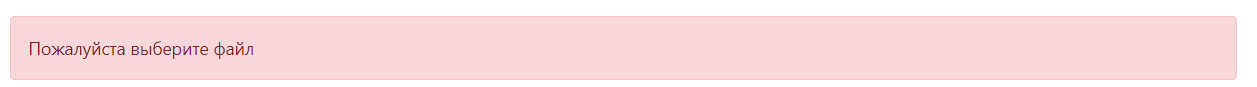


Рисунок 3.15 – Окно информирующее об отсутствии файла.

После успешного добавления сотрудника из файла система выведет окно, указанное на рисунке 3.16

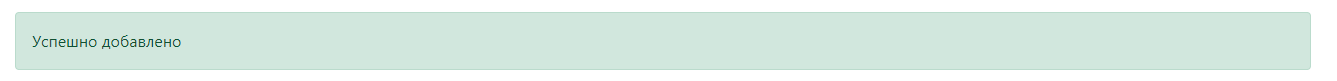


Рисунок 3.16 – Окно информирующее об успешном добавлении файла.

# **4. Заключение**

В заключение, разработанное веб-приложение обладает рядом функциональных возможностей, таких как редактирование, фильтрация, сортировка, добавление и удаление записей, что обеспечивает пользователям удобство и эффективность в управлении информацией.

Это приложение будет полезно для руководителей и отделу по управлению персоналом, которым необходимо оперативно управлять данными о персонале компании, а также для сотрудников, которым требуется быстрый доступ к информации о коллегах.

Однако, чтобы приложение стало еще более полезным и функциональным, можно внести несколько улучшений. В частности, можно добавить больше функций для манипуляции данными сотрудников, таких как возможность редактировать дополнительные поля информации о сотрудниках или отслеживать их рабочий процесс. Также стоит рассмотреть внедрение механизма миграции данных для обеспечения безопасности и целостности информации. Кроме того, важным улучшением будет добавление функционала аутентификации и авторизации с возможностью входа под индивидуальным логином и паролем с различными уровнями доступа к данным о сотрудниках.

В целом, разработанное приложение представляет собой инструмент для управления данными о сотрудниках компании, однако его функциональность может быть расширена для удовлетворения более широкого спектра потребностей пользователей.