|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 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](#_Toc164117364)

[1.1 UML диаграмма Прецедентов 4](#_Toc164117365)

[1.2 UML диаграмма Классов //Изменить названия 5](#_Toc164117366)

[1.3 UML диаграмма Состояний 7](#_Toc164117367)

[1.4 UML диаграмма Последовательностей 8](#_Toc164117368)

[2. Листинг кода прикладной программы(КОД) 9](#_Toc164117369)

[3. Результат выполнения программы 10](#_Toc164117370)

[4. Заключение 16](#_Toc164117371)

# **1. Проектирование программного продукта**

## **1.1 UML диаграмма Прецедентов**

Данная диаграмма 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.

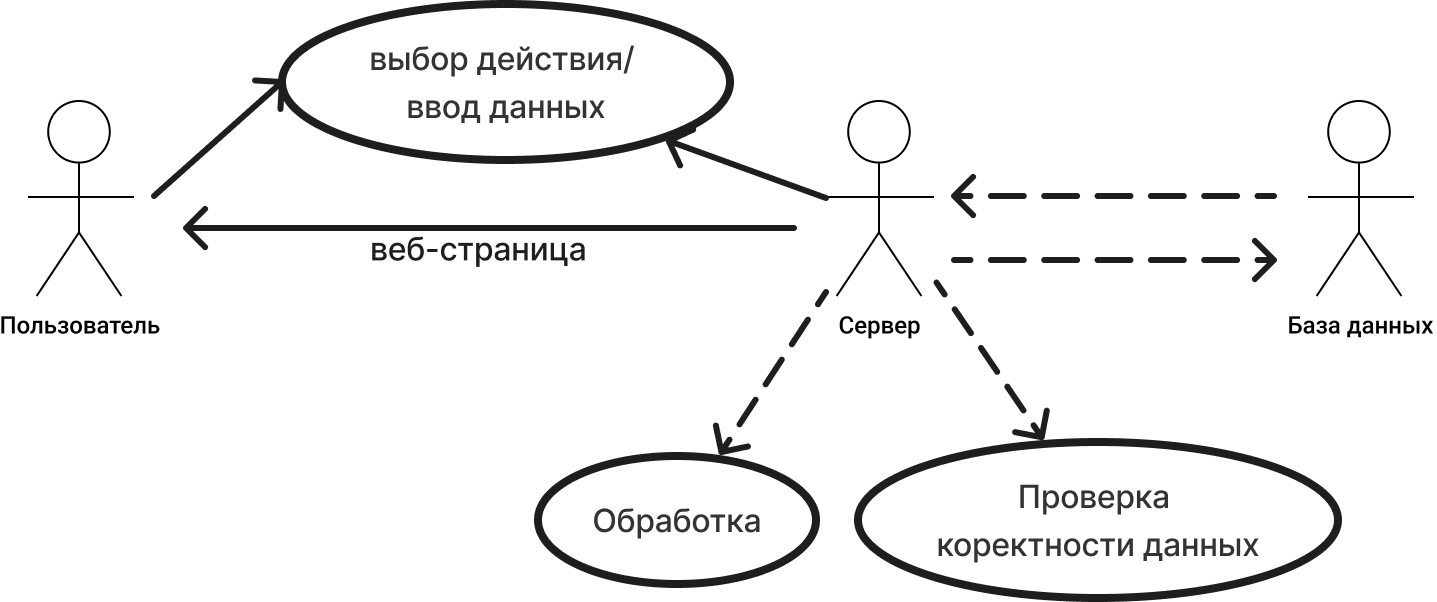


Рисунок 1.1 UML диаграмма Прецедентов

## **1.2 UML диаграмма Классов //**Изменить названия

Данный вид диаграммы 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 EmployeeController обрабатывает HTTP-запросы и управляет логикой приложения. Взаимодействие с базой данных осуществляется через класс ApplicationContext, который представляет собой контекст базы данных Entity Framework.

Каждый сотрудник представлен объектом класса Employee, 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 EmployeeController и представлениями (Views) осуществляется через передачу моделей данных. Например, метод Create отображает представление для создания нового сотрудника, а метод GetEmployees 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.

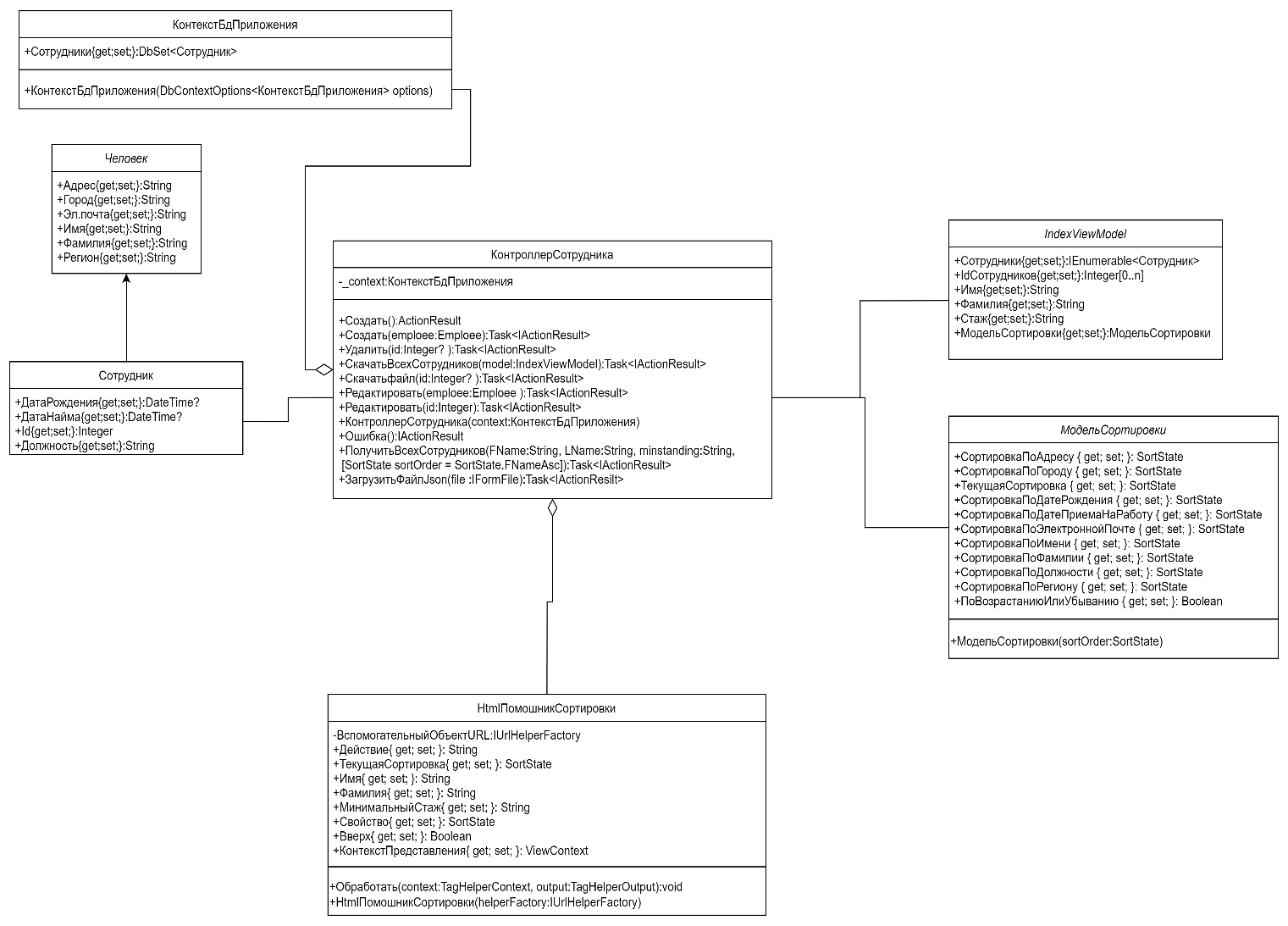
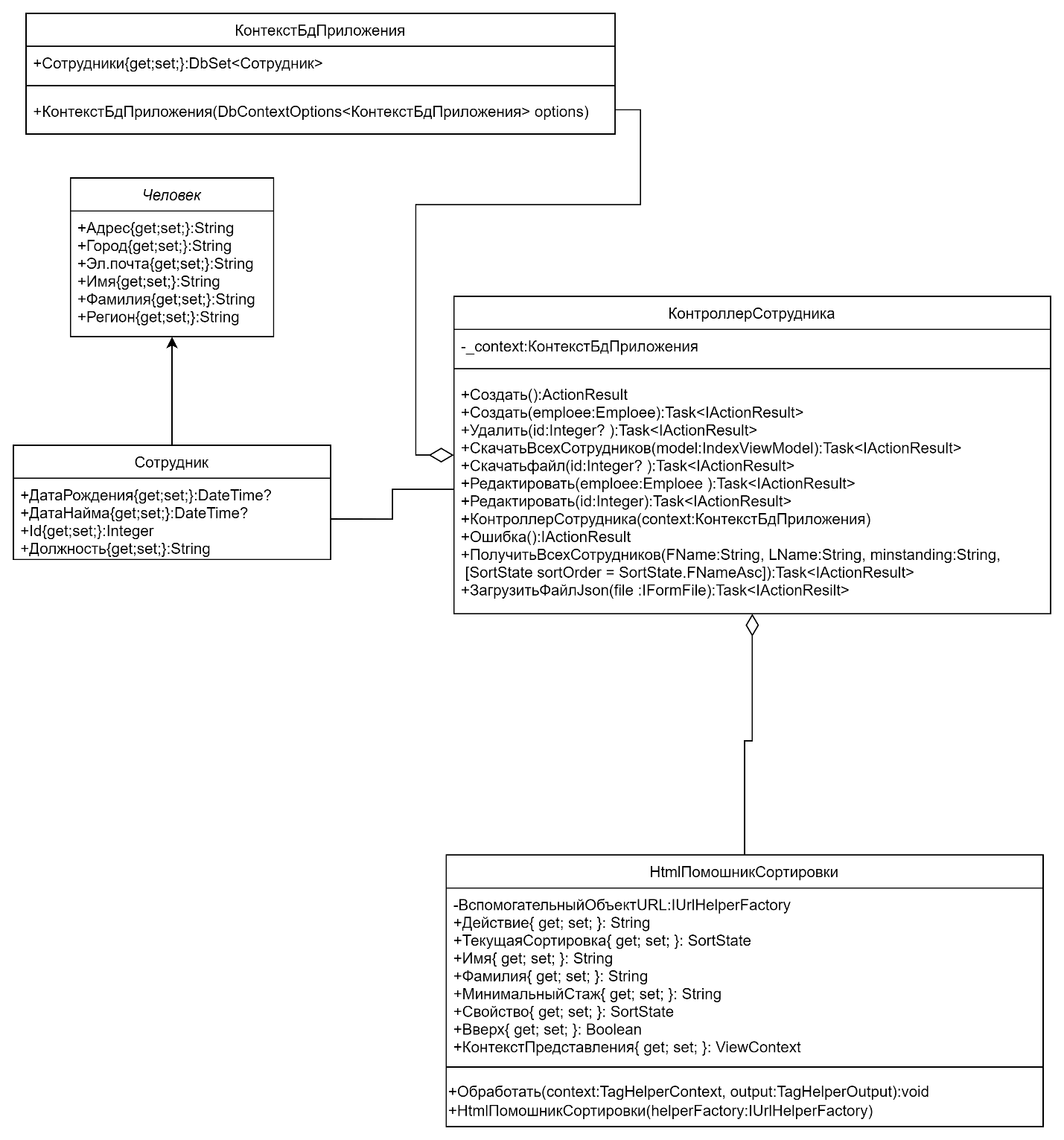


Рисунок 1.2 UML диаграмма Классов



## **1.3 UML диаграмма Состояний**

Диаграмма Состояний 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.

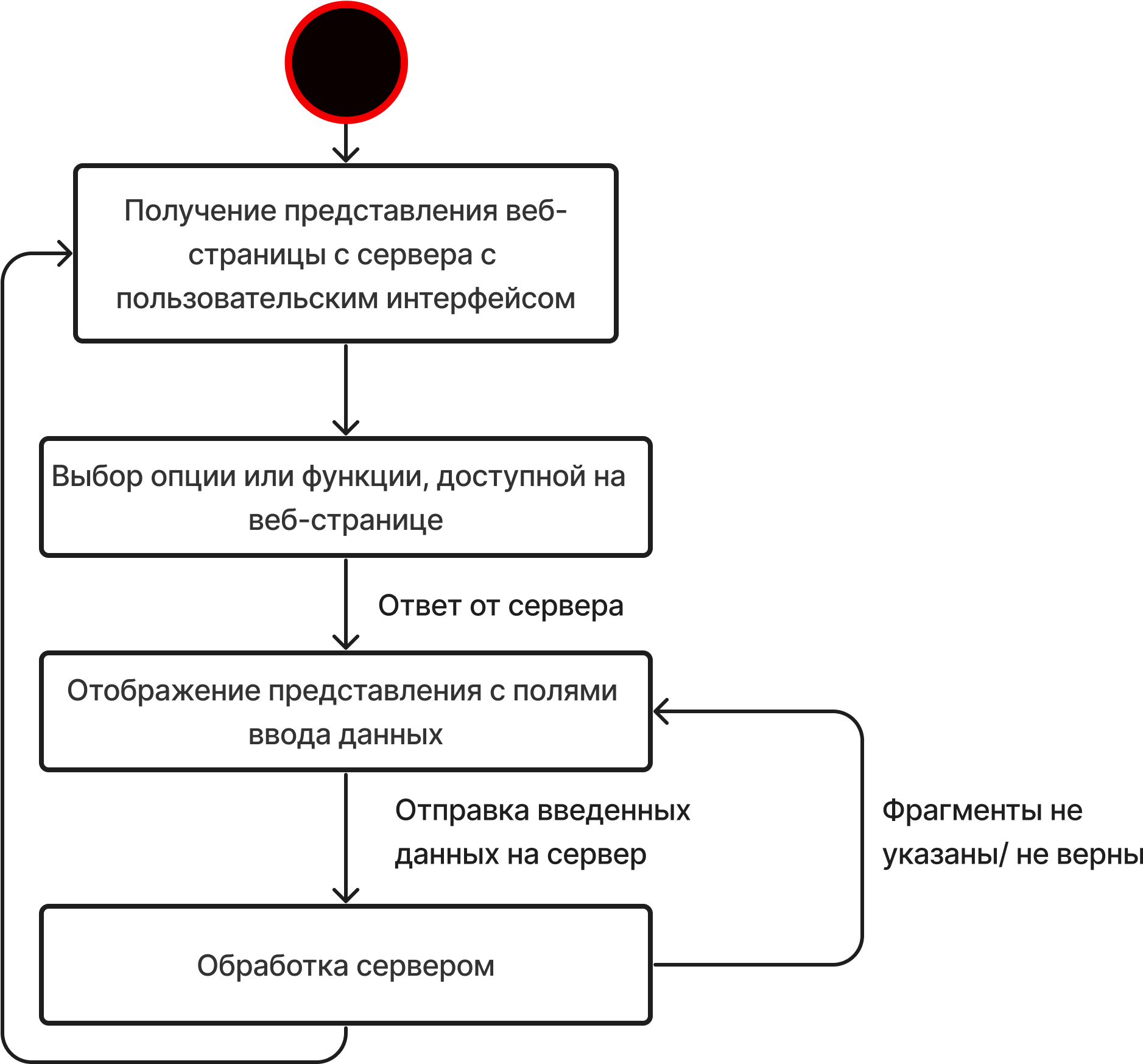


Рисунок 1.3 UML диаграмма Состояний

## **1.4 UML диаграмма Последовательностей**

Диаграмма Последовательностей 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 Последовательносте

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

# **3. Результат выполнения программы**

При запуске веб-приложения перед нами открывается главная страница, изображенная на странице 4.1 на которой находятся:

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

-Таблица всех сотрудников

-Поля фильтрации таблицы

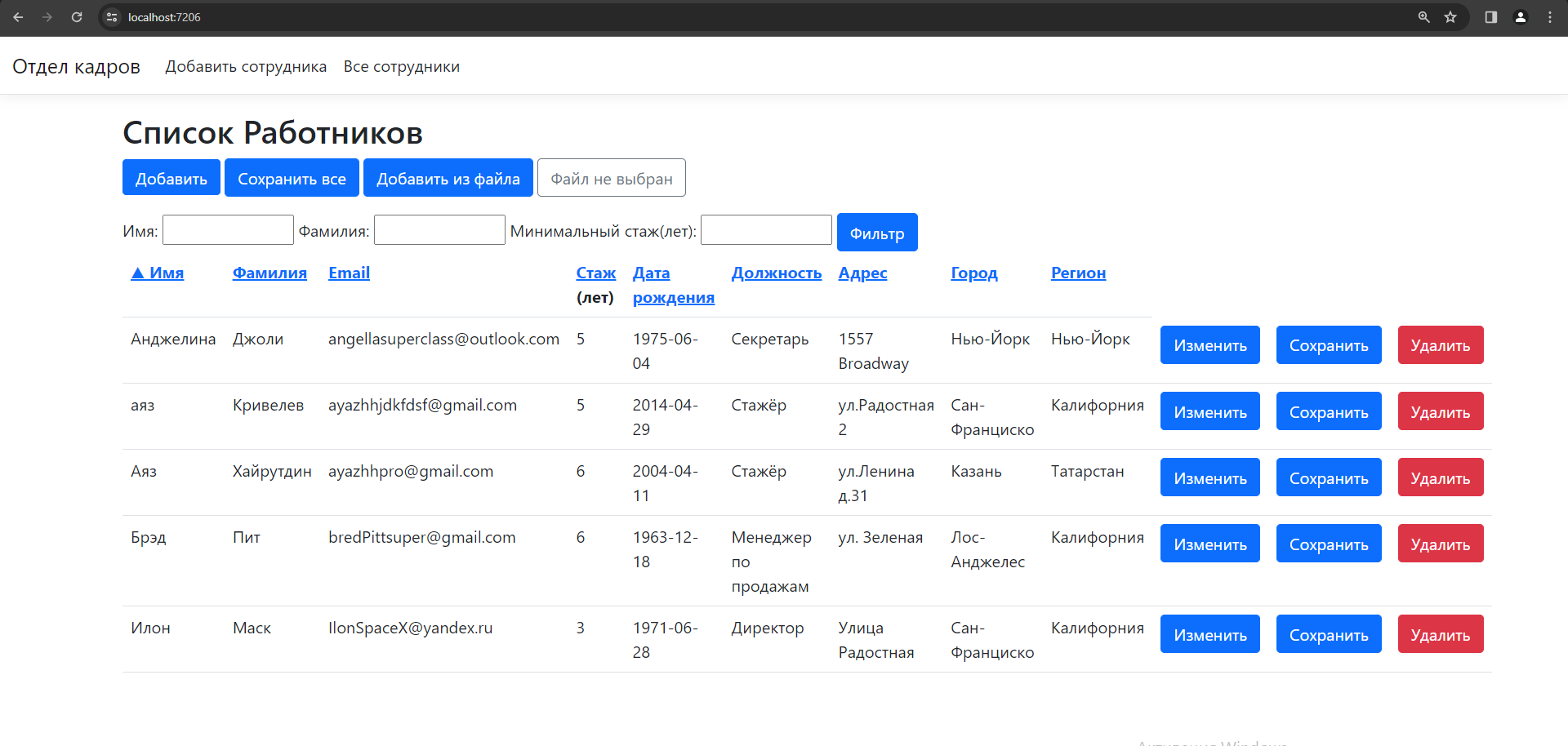
-Кнопки с названиями колонок таблицы, отвечающие за сортировки

-Кнопки с добавлением сотрудника/сотрудников из файла

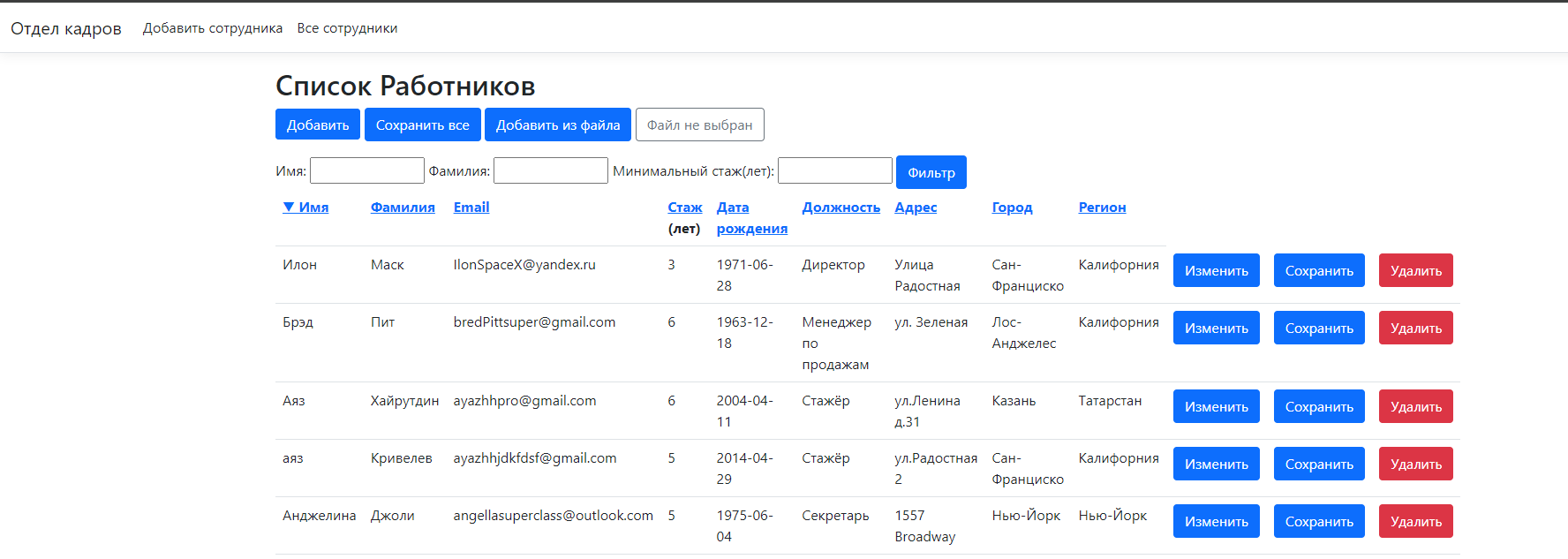
-Кнопка «Добавить»

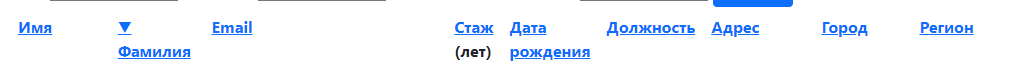
-Кнопка для сохранения всех сотрудников

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

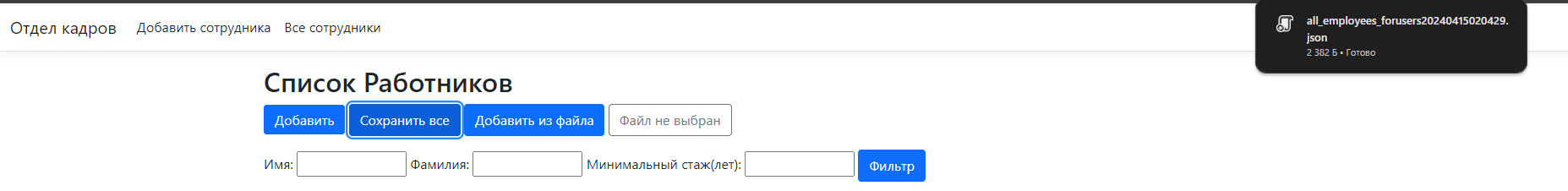


При нажатии на названия колонок («Имя», «Фамилия», «Email», «Стаж», «Дата Рождения», «Должность», «Адрес», «Город», «Регион»), таблица будет автоматически сортироваться по возрастанию или по убыванию в зависимости от выбранной сортировки. Направление текущей сортировки будет указано стрелкой рядом с заголовком колонки. Для полей строкового типа сортировка будет происходить по алфавиту. Пример указан на рис. 4.№ и 4.№.

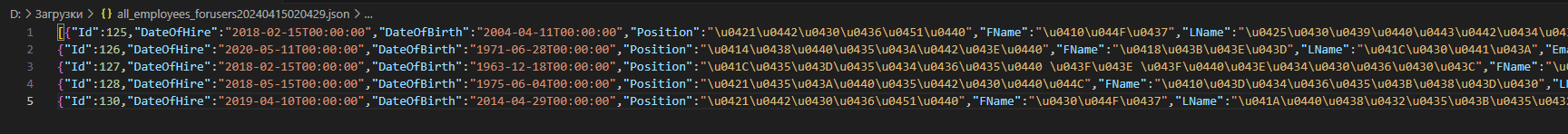




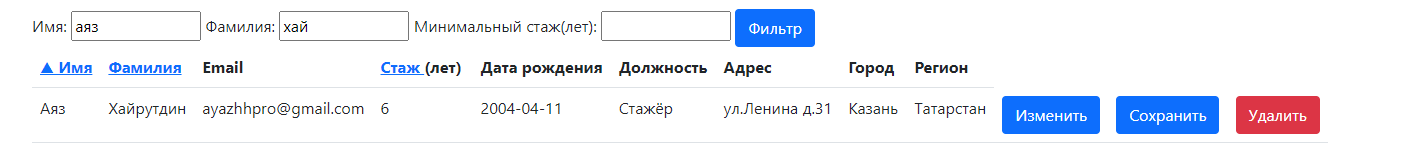
При нажатии кнопки «Сохранить все» в загрузки вашего браузера добавится .json файл с информацией о всех сотрудниках. Это показано на рис. 4.№.



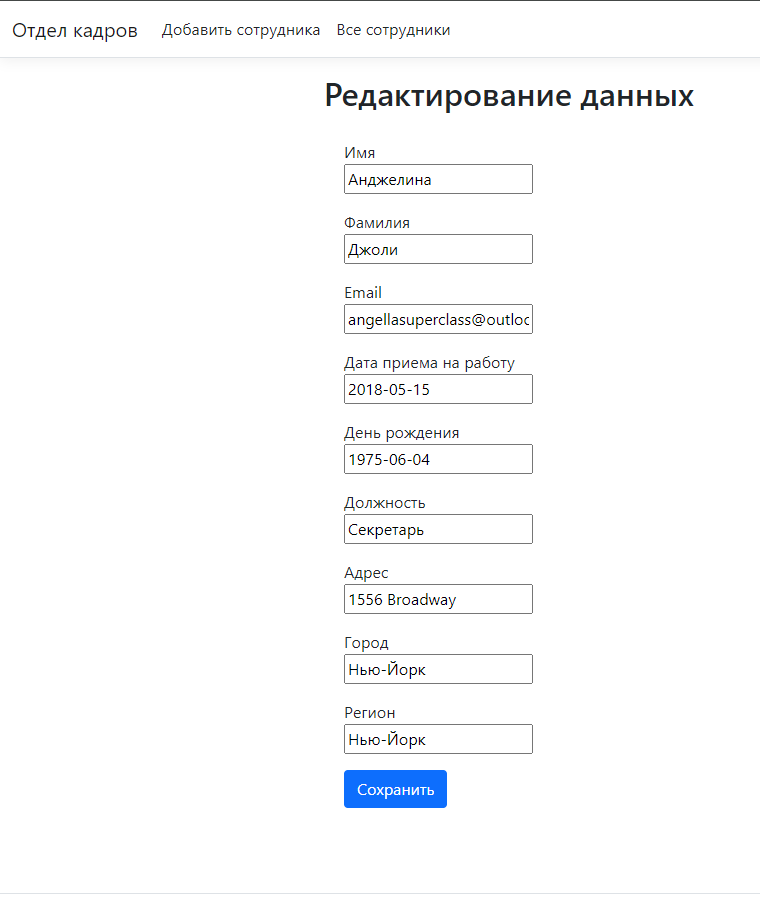
На рис. 4.№ показано как выглядит файл .json, если открыть его в специальном текстовом редакторе



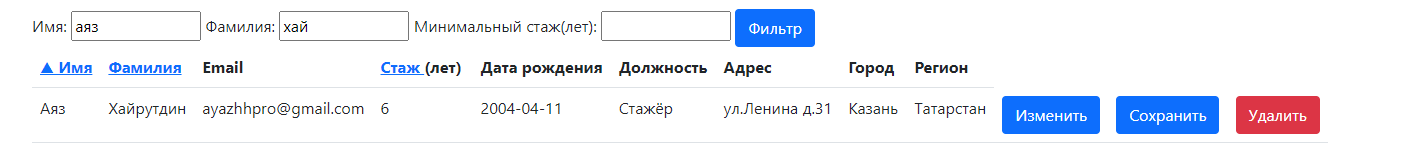
При нажатии кнопки «Удалить» информация о данном человеке будет стерта из Базы данных. При нажатии кнопки «Сохранить», находящиеся напротив информации о каждом сотруднике, изображенной на рис. 4.№. В загрузки вашего браузера добавится .json файл с информацией только о выбранном сотруднике.



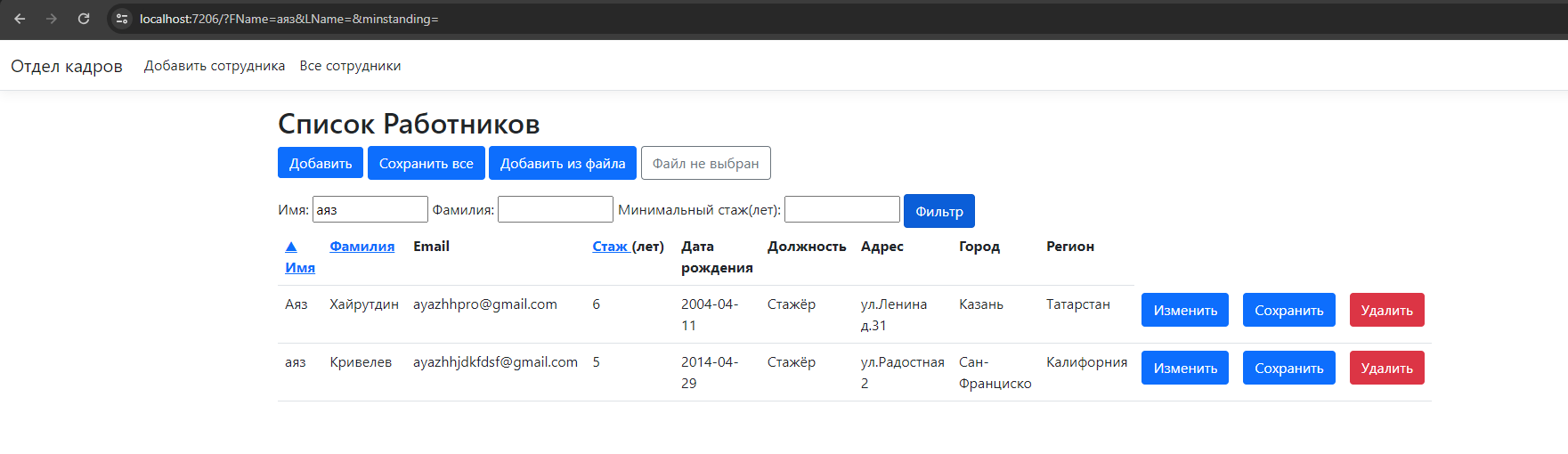
При нажатии кнопки «Изменить» нас перекинет на веб-страницу с полями данных о сотруднике, доступных для редактирования, изображено на рис. 4.%. После внесения изменений, нужно нажать кнопку «Сохранить», тогда изменения вступят в силу и нас перебросит обратно на главную страницу.

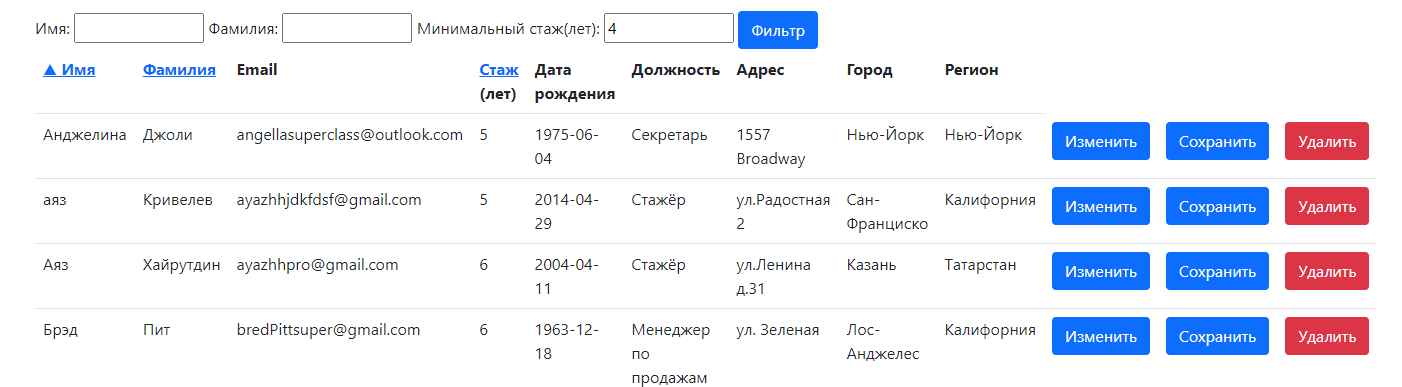


При вводе имени, фамилии или минимального стажа сотрудника, а затем нажатии на кнопку "Фильтр", произойдет фильтрация таблицы. Будут отображены только те записи, которые соответствуют введенным данным. Эта функция также может использоваться для поиска необходимой информации. На рис. 4.№ указан фильтр который отобразит только тех сотрудников, чье имя начинается на «Аяз» и фамилия начинается на «хай».

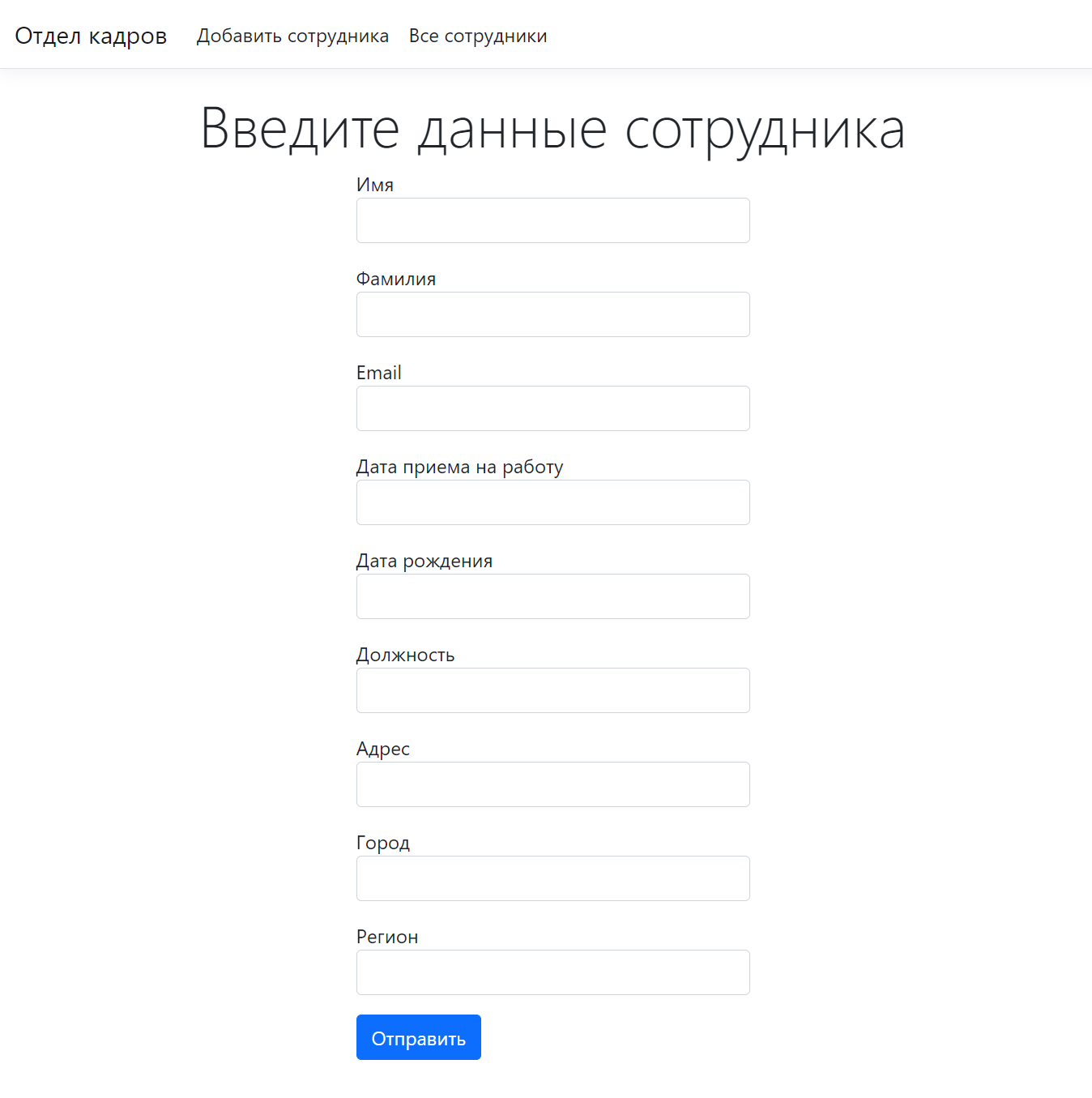


Также можно отфильтровать только по одному полю. Например на рис. 4.№ указан фильтр который отобразит всех сотрудников, чье имя начинается на «аяз». На рис. 4.№ Отображаются только, те чей стаж работы больше четырех лет.





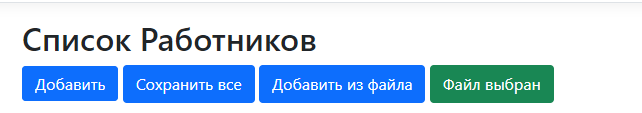
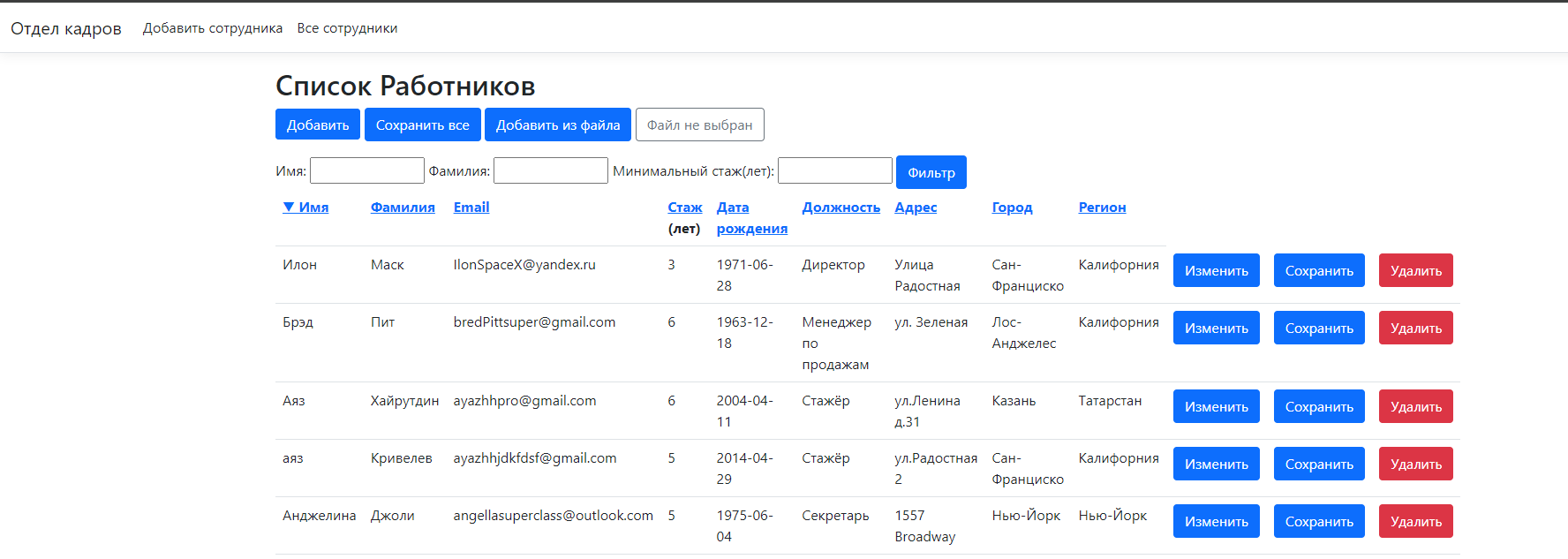
При нажатии на кнопку «Добавить» или на кнопку «Добавить сотрудника», находящуюся в шапке страницы, нас перекинет на веб-страницу с полями для ввода данных нового сотрудника, это показано на рис. 4.№.

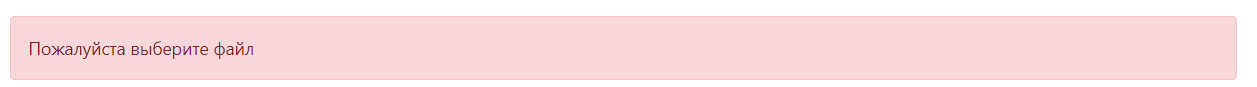


После заполнения полей данными, следует нажать кнопку «Отправить». В случае успешного добавления сотрудника система выведет окно, указанное на рисунке 4.№

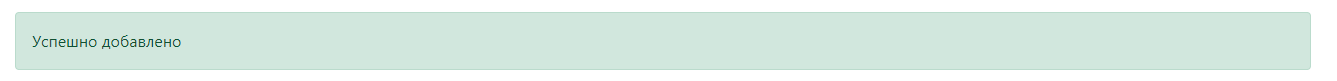


Чтобы добавить нового сотрудника или группу сотрудников, нажмите кнопку «Выбрать файл», как показано на рис. 4.1. После этого откроется окно выбора файла на вашем компьютере, ищите файл в формате .json. По умолчанию кнопка будет отображаться в сером цвете. После успешного выбора файла, кнопка станет зеленой и отобразит надпись «Файл выбран», как показано на рис. 4.2. 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 4.№.





После успешного добавления система выведет окно, указанное на рисунке 4№.



# **4. Заключение**

Класс **EmploeeController** является контроллером, который обрабатывает запросы пользователя и взаимодействует с моделью данных и представлениями. Давайте разберем его методы:

1. **Конструктор**: Принимает контекст базы данных **ApplicationContext**, который используется для выполнения операций с базой данных.
2. **Метод Error**: Отображает представление с информацией об ошибке.
3. **Метод Create**: Отображает представление для создания нового работника.
4. **Метод Create (POST)**: Обрабатывает POST-запрос с данными нового работника, добавляет их в базу данных и перенаправляет пользователя на страницу создания.
5. **Метод GetEmployees**: Получает список работников с учетом фильтрации и сортировки, а затем отображает их на странице.
6. **Метод Delete**: Обрабатывает POST-запрос для удаления работника из базы данных по его идентификатору.
7. **Метод Edit**: Отображает представление для редактирования информации о работнике с указанным идентификатором.
8. **Метод Edit (POST)**: Обрабатывает POST-запрос с обновленными данными о работнике и сохраняет их в базе данных.
9. **Метод UploadJsonFile**: Обрабатывает POST-запрос для загрузки данных о работниках из JSON-файла, добавляет их в базу данных и возвращает сообщение об успешной загрузке.
10. **Метод DownloadFile**: Обрабатывает GET-запрос для скачивания JSON-файла с информацией о работнике с указанным идентификатором.
11. **Метод DownloadAllEmploees**: Обрабатывает POST-запрос для скачивания JSON-файла с информацией о всех работниках, указанных в модели представления.
12. **SortState**: Это перечисление, которое определяет возможные состояния сортировки для различных полей (имени, фамилии, электронной почты, даты приема на работу и т. д.).
13. **SortHeaderTagHelper**: Это класс тег-помощник, который генерирует HTML-теги ссылок для сортировки данных. Он принимает свойства, такие как **Property** (текущее свойство для сортировки), **Current** (активное свойство для сортировки), **Action** (действие контроллера для создания ссылки), и т. д. В зависимости от состояния сортировки он генерирует соответствующие HTML-теги и добавляет стрелку (вверх или вниз), указывающую на направление сортировки.
14. **SortViewModel**: Этот класс представляет модель представления для управления состоянием сортировки на странице. Он содержит свойства для всех полей, по которым можно сортировать, а также для текущего состояния сортировки и флага для указания направления сортировки (вверх или вниз).
15. **IndexViewModel**: Этот класс представляет модель представления для основной страницы, например, списка сотрудников. Он содержит коллекцию сотрудников, поля для фильтрации (**FName**, **LName**, **minstanding**) и объект модели представления **SortViewModel**, который используется для управления состоянием сортировки.