**Поволжский Государственный Университет Телекоммуникаций и Информатики**Кафедра «Программная инженерия»

|  |  |
| --- | --- |
| Сдана на проверку  «\_\_» \_\_\_\_2023 г. | Допустить к защите  «\_\_» \_\_\_\_\_ 2023 г.  Защищена с оценкой  «\_\_\_» \_\_\_\_\_\_\_\_ 2023 г. |

**КУРСОВАЯ РАБОТА**По дисциплине: «Прикладное программирование»  
На тему: «Разработка клиент-серверного web-приложения Медицинская Учетная Платформа - MediTrack»  
  
Пояснительная записка

|  |  |
| --- | --- |
| Студент группы ИВТ-11\_\_\_\_\_\_\_ Клинский А. Е.  (роспись) (ФИО)  Руководитель \_\_\_\_\_\_\_\_ к.т.н., доц. Ахметшина Э. Г.  (роспись) (ФИО) | 0 210212 0  (№ зачетной книжки) |

Самара 2023

**Рецензия**

**Содержание**

[Описание предметной области. Актуальность 4](#_Toc1)

[Описание программы. Общие сведения 5](#_Toc2)

[Функциональное назначение 5](#_Toc3)

[Описание технологий 6](#_Toc4)

[Логическая модель базы данных 7](#_Toc5)

[Физическая модель базы данных 9](#_Toc6)

[Диаграмма классов 10](#_Toc7)

[Диаграмма компонентов 11](#_Toc8)

[Диаграмма вариантов использования 12](#_Toc9)

[Диаграмма последовательности 13](#_Toc10)

[Демонстрация работы приложения 14](#_Toc11)

[Список используемых источников 25](#_Toc12)

[Приложение А – Листинг программного кода 26](#_Toc13)

# Описание предметной области. Актуальность

В современных медицинских учреждениях эффективное управление ресурсами и данных является ключевым фактором для обеспечения качественного предоставления медицинских услуг. В этом контексте создание платформы, специально адаптированной для учета медицинских ресурсов, представляет собой актуальное и перспективное направление.

Медицинские учреждения ежедневно сталкиваются с управлением различными видами ресурсов, такими как медицинское оборудование, расходные материалы, лекарства, персонал, и другие аспекты, необходимые для предоставления качественных услуг. Оптимизация процессов учета и управления этими ресурсами становится приоритетной задачей в условиях постоянных изменений в медицинской сфере.

Существующие методы учета медицинских ресурсов, такие как ручной учет, таблицы Excel или устаревшие информационные системы, не обеспечивают достаточной эффективности и точности. С учетом динамичной природы медицинской сферы, необходимость в современной, автоматизированной платформе для учета и управления ресурсами становится более чем актуальной.

Разработка и внедрение специализированной платформы для учета медицинских ресурсов позволит улучшить процессы управления, снизить вероятность ошибок и улучшить качество предоставляемых медицинских услуг. Это также обеспечит возможность оперативного реагирования на изменения в потребностях и требованиях здравоохранения.

Разработка платформы для учета медицинских ресурсов не только актуальна в современной медицинской среде, но и является важным шагом в направлении повышения эффективности управления медицинскими ресурсами. Ее внедрение существенно улучшит качество предоставляемых медицинских услуг и обеспечит учреждениям возможность адаптироваться к постоянно меняющимся требованиям здравоохранения.

# Описание программы. Общие сведения

Приложение – веб-ресурс. Установка не требуется.

Необходимые требования: доступ к интернету.

Языки программирования: Java, JavaScript.

Фреймворки/Библиотеки: Spring Boot, React.js

Среда разработки: Intellij IDEA, VS Code

Объем проекта: 458 МБ (вместе с исходным кодом).

Исходный код (классы и код форм): 1697 строк.

# Функциональное назначение

Платформа разработана с учетом сложности управления медицинскими ресурсами в современных условиях. Основной целью является предоставление медицинским учреждениям эффективного инструмента для учета и управления различными видами ресурсов, такими как медицинское оборудование, расходные материалы, лекарства и персонал.

**Основные функции приложения:**

**Выбор сущности:** пользователи могут выбирать конкретные сущности, такие как оборудование, расходные материалы или персонал, для управления.

**Просмотр записей:** позволяет просматривать все сохраненные записи выбранной сущности, обеспечивая полный обзор текущего состояния ресурсов.

**Редактирование и удаление записей:** пользователи имеют возможность изменять или удалять существующие записи для актуализации данных.

**Создание новых записей:** позволяет добавлять новые записи, открывая всплывающую форму для ввода необходимой информации.

# Описание технологий

Приложение разработано с использованием современных технологий, обеспечивающих высокую производительность, масштабируемость и удобство использования.

Фронтенд разработан с использованием библиотеки **React**, обеспечивающей динамичный пользовательский интерфейс и удобное взаимодействие пользователя с приложением.

Бэкенд приложения реализован с использованием фреймворка **Spring Boot**, что обеспечивает высокую производительность, удобство в построении **RESTful API** и надежность.

Для обмена данными между клиентом и сервером используется протокол **HTTP**, а данные представляются в формате **JSON**, обеспечивая стандартизированный и легко читаемый формат.

Для хранения данных использована система управления базами данных **PostgreSQL**, что обеспечивает надежное и эффективное хранение информации.

Эти технологии были выбраны с учетом их популярности, поддержки сообщества, а также способности обеспечивать требования по масштабируемости и безопасности, необходимые для успешного функционирования платформы в медицинских учреждениях.

# Логическая модель базы данных

**Пациенты**

* Пациент\_ID (PK)
* Имя
* Фамилия
* Дата рождения
* Пол
* Контактная информация

**Медицинские сотрудники**

* Сотрудник\_ID (PK)
* Имя
* Фамилия
* Должность
* Контактная информация

**Медицинское оборудование**

* Оборудование\_ID (PK)
* Название
* Производитель
* Дата приобретения
* Инвентарный номер

**Типы оборудования**

* Тип\_ID (PK)
* Наименование типа

**Записи о приемах**

* Запись\_ID (PK)
* Дата и время приема
* Пациент\_ID (FK)
* Сотрудник\_ID (FK)
* Диагноз
* Описание приема

**Поставщики**

* Поставщик\_ID (PK)
* Название
* Контактная информация

Связь между Записи о приемах и Пациенты: Отношение "многие к одному" (Many-to-One), так как у одного пациента может быть много записей о приемах, но каждая запись о приеме принадлежит одному пациенту.

Связь между Записи о приемах и Медицинские сотрудники: Отношение "многие к одному" (Many-to-One), так как у одного сотрудника может быть много записей о приемах, но каждая запись о приеме связана с одним сотрудником.

Связь между Записи о приемах и Медицинское оборудование: Отношение "многие к одному" (Many-to-One), так как у каждой записи о приеме может быть связано с одним медицинским оборудованием, но у каждого оборудования может быть много записей о приемах.

Связь между Медицинское оборудование и Типы оборудования: Отношение "многие ко многим" (Many-to-Many), так как у каждого оборудования может быть связано с несколькими типами, и у каждого типа может быть связано с несколькими оборудованиями.

Остальные связи являются прямыми связями "многие к одному" (Many-to-One).

# Физическая модель базы данных

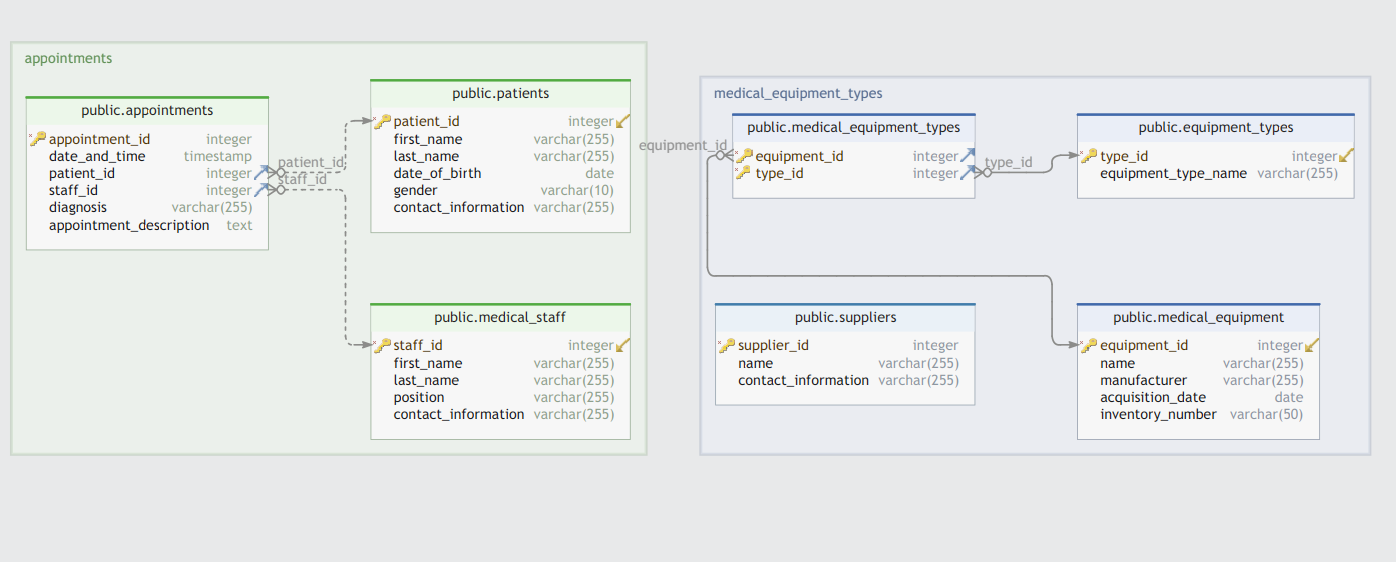


Рис. 1.1 – Физическая модель базы данных

# Диаграмма классов

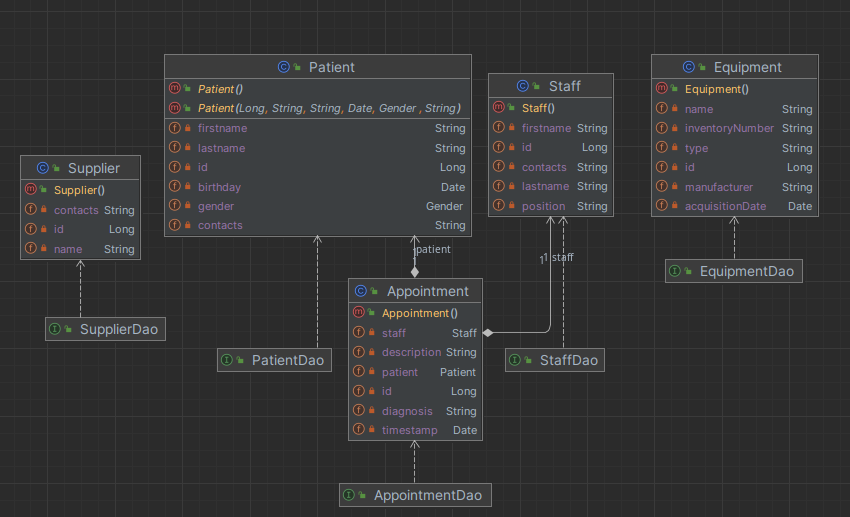


Рис. 2.1 – Диаграмма Java классов



Рис. 2.2 – Диаграмма JavaScript классов

# Диаграмма компонентов

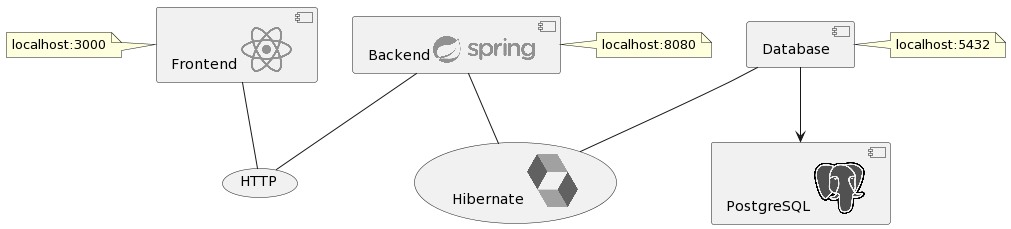


Рис. 3.1 – Диаграмма компонентов системы.

# Диаграмма вариантов использования

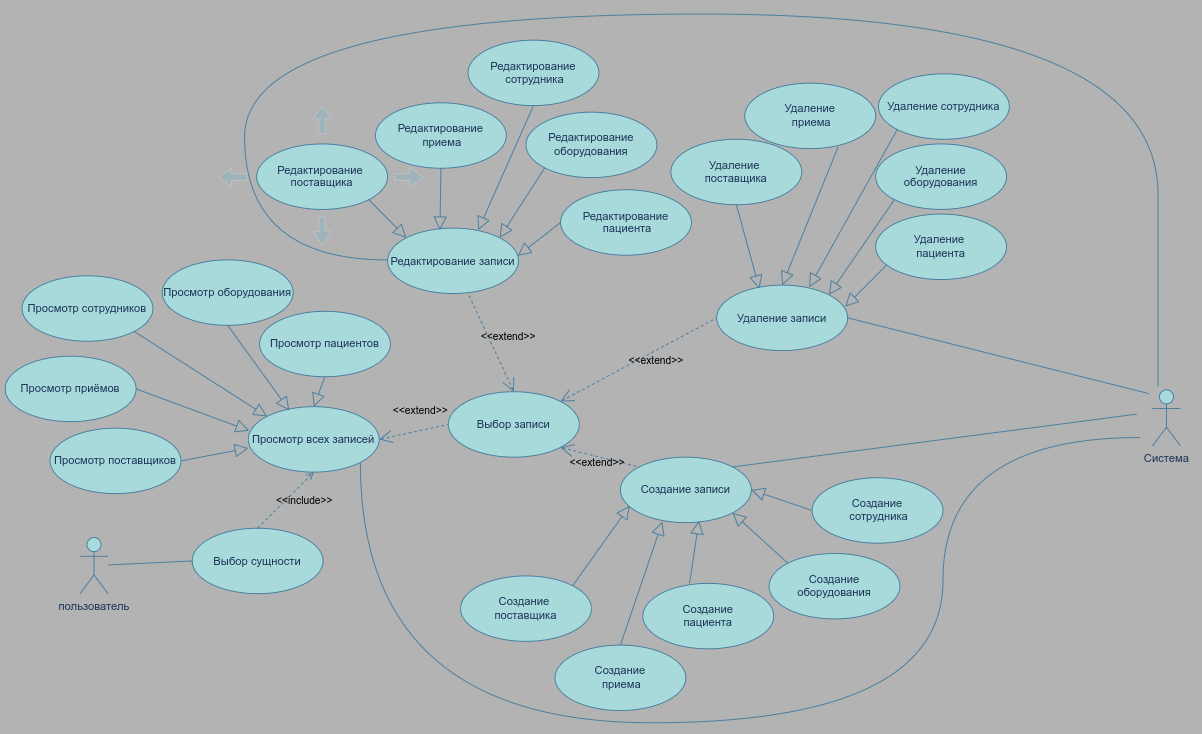


Рис. 4.1 – Диаграмма вариантов использования системы

Диаграмма вариантов использования (Use Case Diagram) представляет собой средство визуализации функциональности системы, позволяя описать, как различные акторы взаимодействуют с системой на основе их вариантов использования.

# Диаграмма последовательности

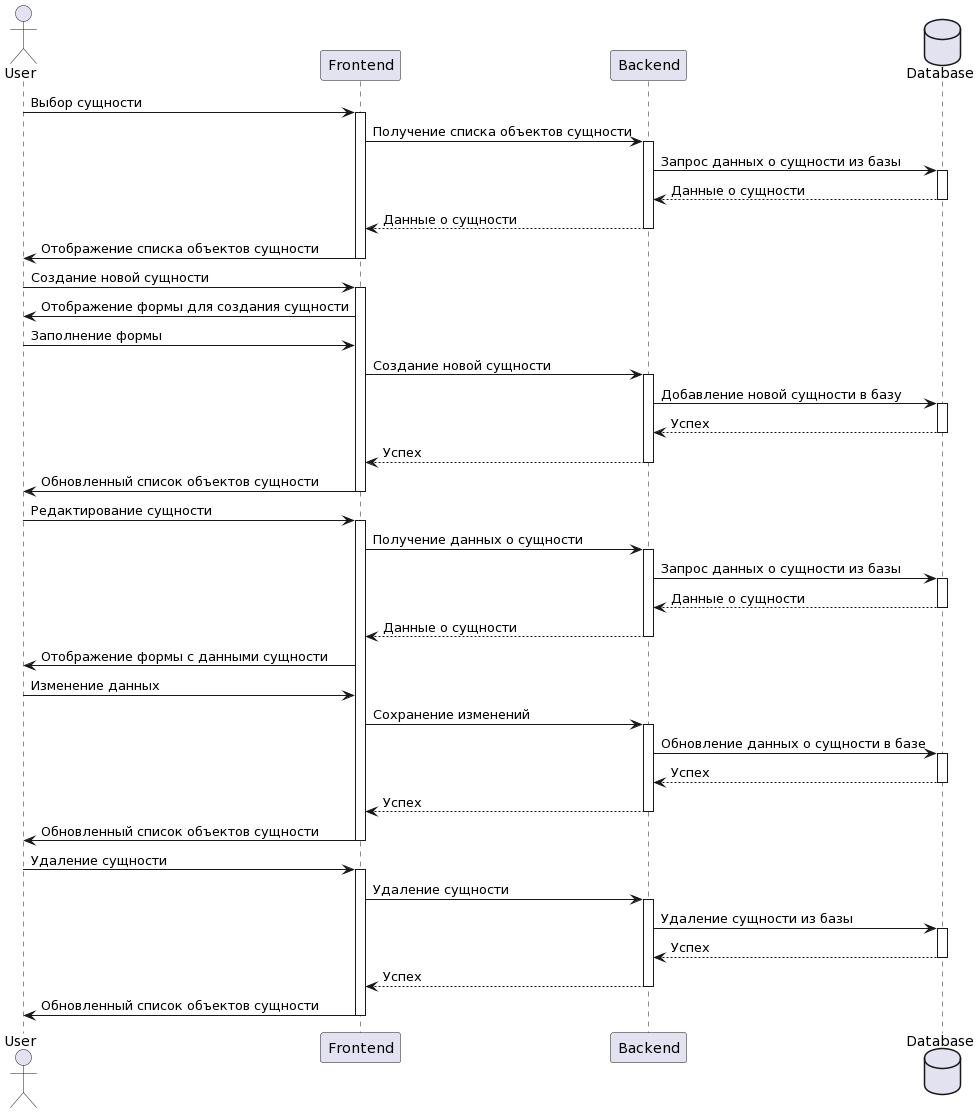


Рис. 5.1 – Диаграмма последовательности

# Демонстрация работы приложения

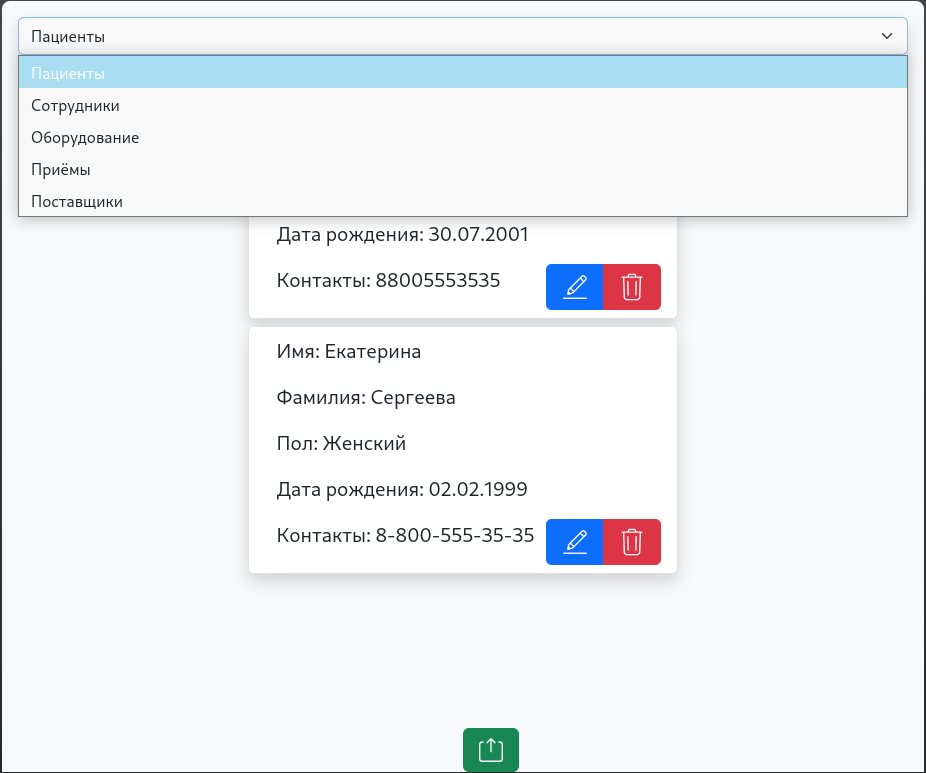


Рис. 6.1 – Выбор сущности «Пациенты»

На главной странице выбираем сущность, которую хотим просмотреть. Для начала выбираем сущность «Пациенты». (Рис. 6.1)

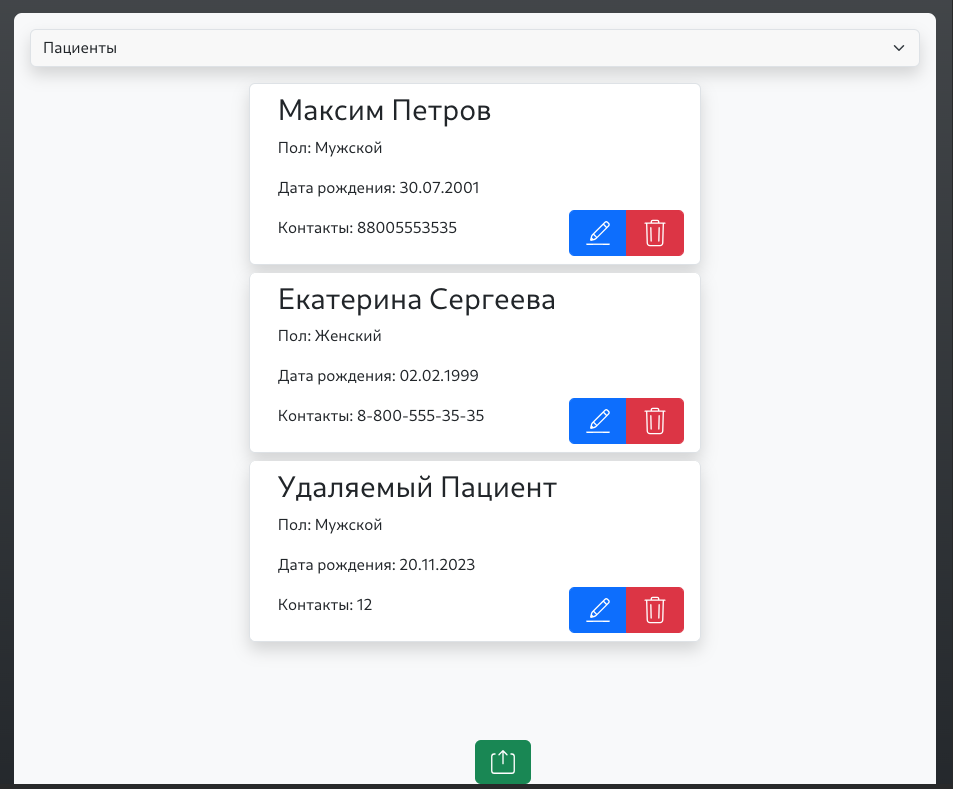


Рис. 6.2 – Список всех пациентов

После выбора мы получаем список существующих пациентов. (Рис. 6.2)

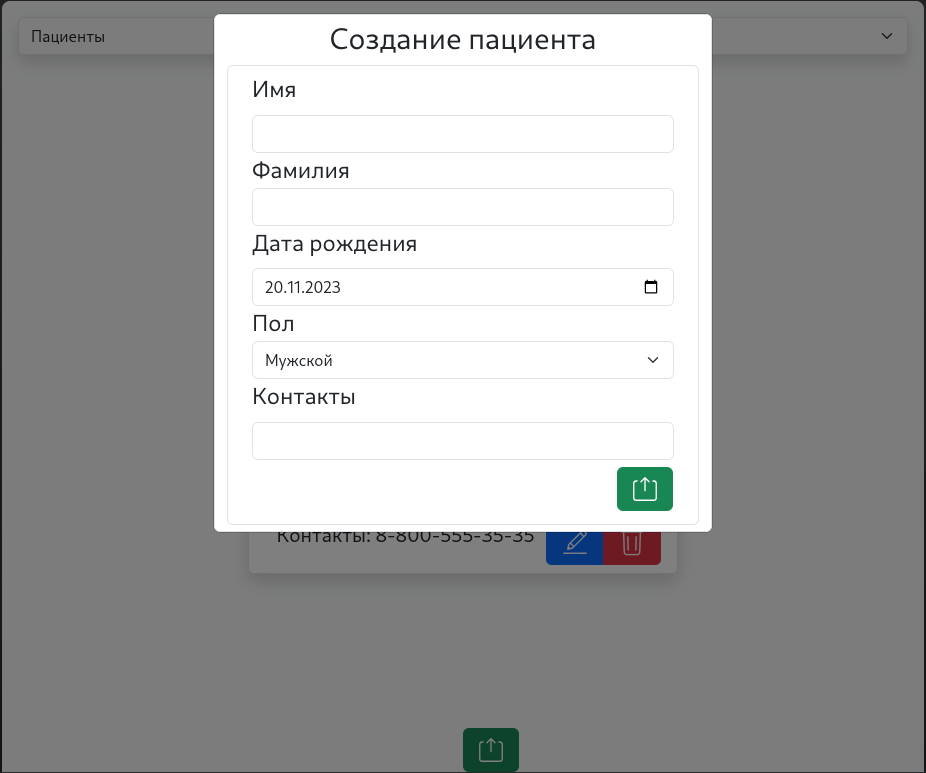


Рис. 6.3 – Форма создания пациента

Далее мы открываем форму для создания нового пациента в системе (Рис. 6.3)

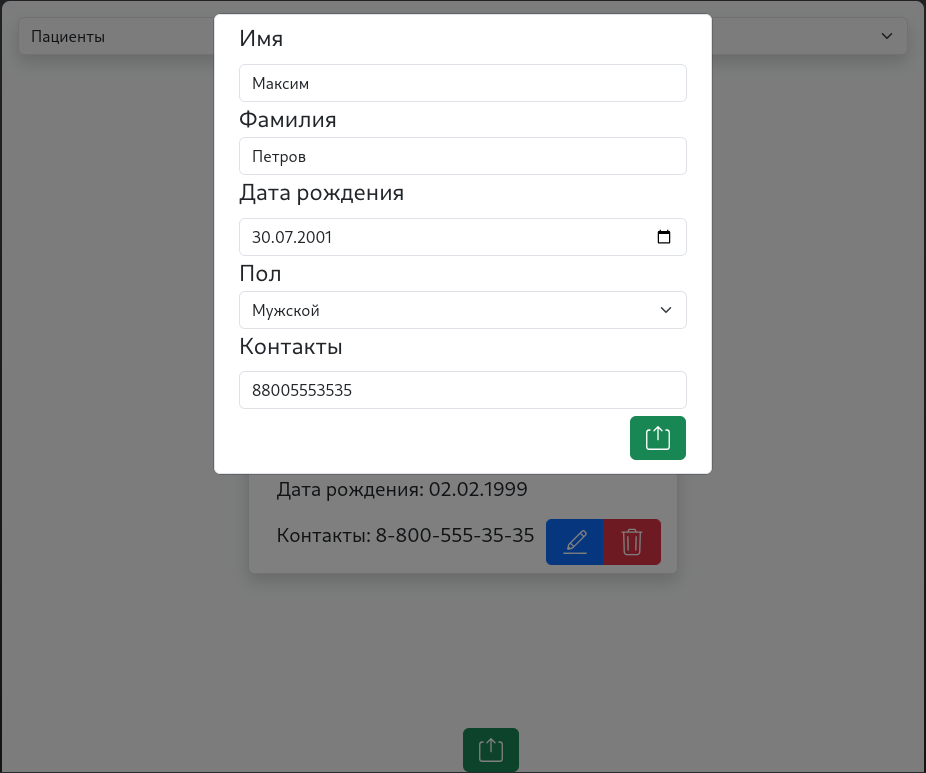


Рис. 6.4 – форма редактирования пациента

После создания нового пациента, редактируем данные существующего. (Рис. 6.4)

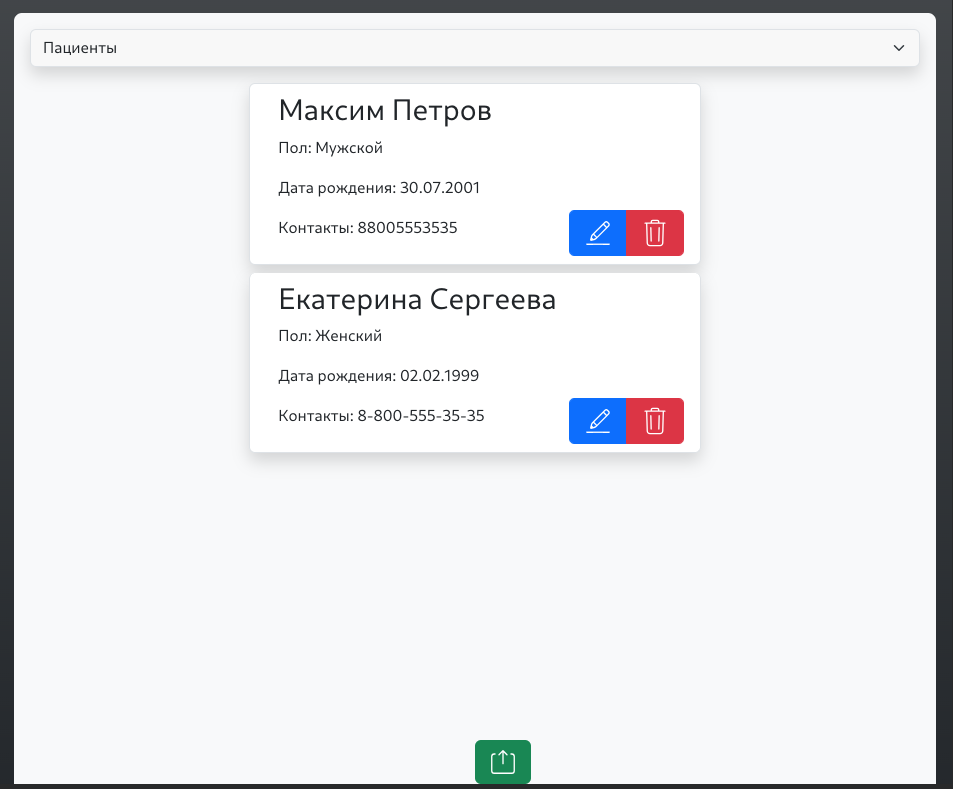


Рис. 6.5 – удаление пациента

На рис. 6.5 представлено последствие удаления после нажатия на красную кнопку с иконкой мусорного бака на карточке пациента. (Рис. 6.5)

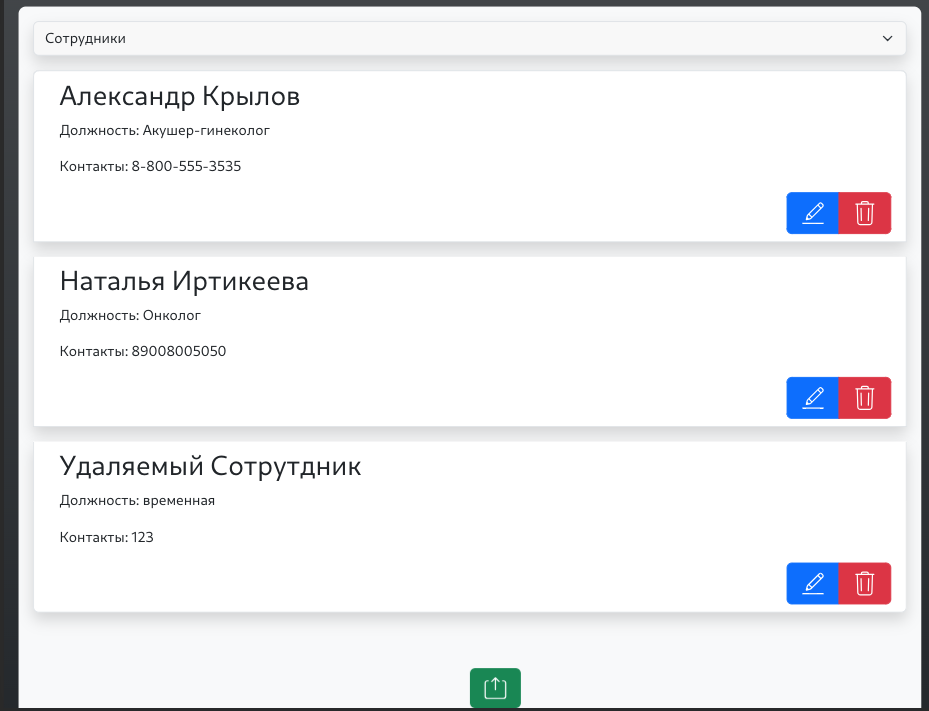


Рис. 6.6 – Выбор сущности «Сотрудники»

Следующим шагом будет просмотр сущности «Сотрутдники». (Рис. 6.6)

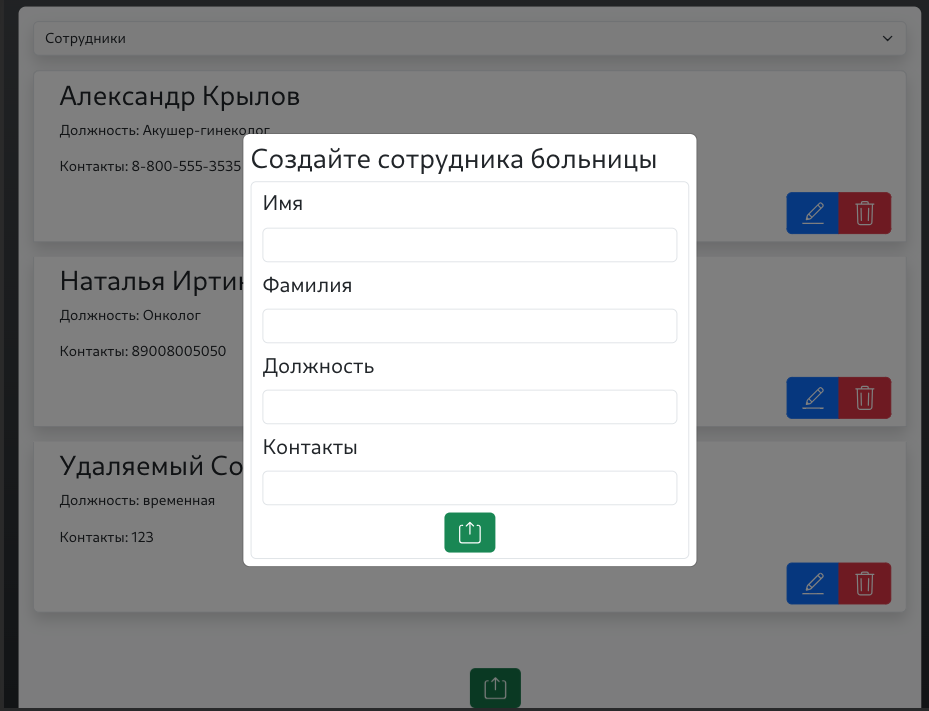


Рис. 6.7 – Форма создания сотрудника

При нажатии на зеленую кнопку создания выводится форма создания сотрудника больницы. (Рис. 6.7)

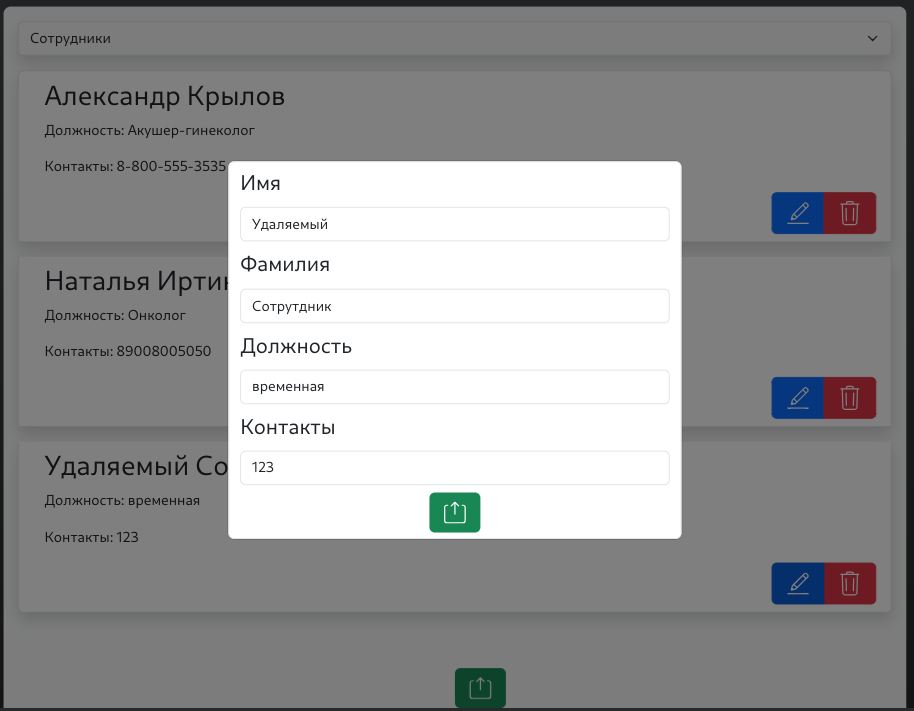


Рис. 6.8 – Форма редактирования сотрудника

При нажатии на синюю кнопку с карандашом на карточке сотрудника выводится форма редактирования сотрудника с исходными данными. (Рис. 6.8)

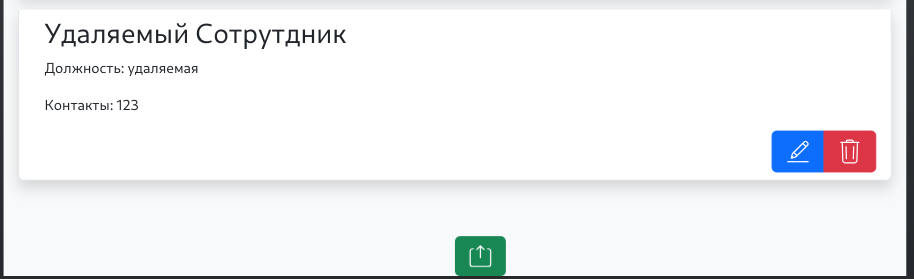


Рис. 6.9 – Отредактированный удаляемый сотрудник

После редактирования мы можем убедиться, что должность поменялась с «временная» на «удаляемая». (Рис. 6.9)

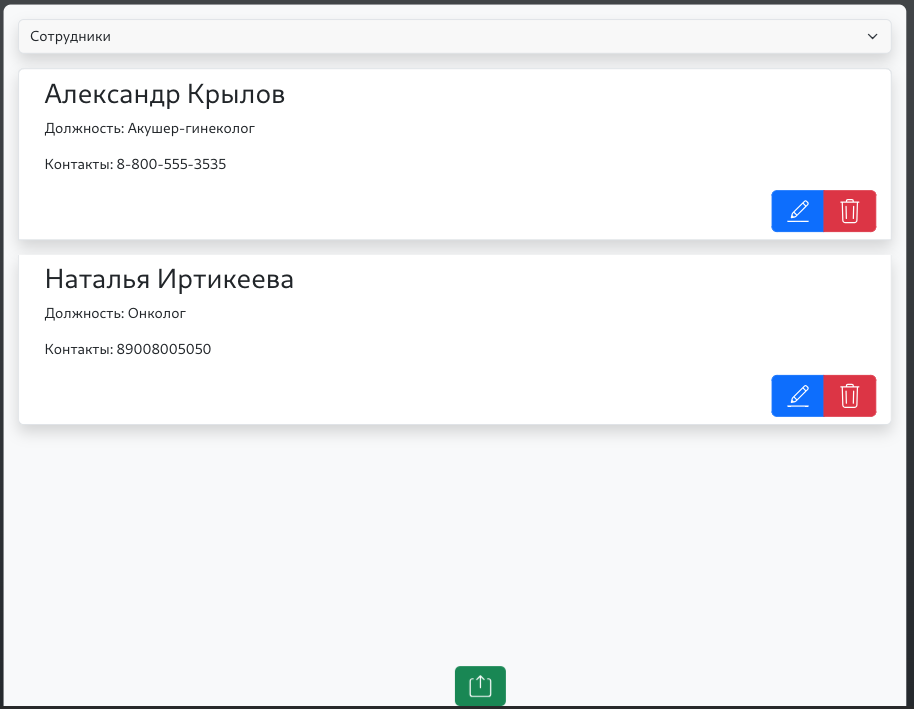


Рис. 6.10 – Список сотрудников после удаления

После нажатия на кнопку удаления, расположенную на карточке сотрудника, сотрудник удаляется из системы. (Рис. 6.10)

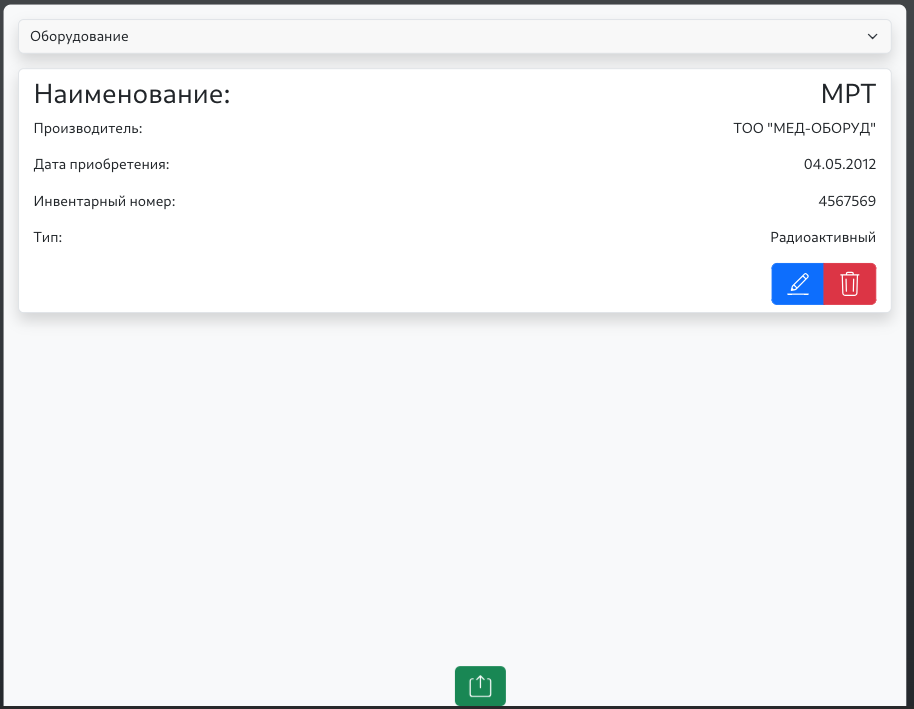


Рис. 6.11- Выбор сущности «Оборудование»

Теперь выбираем сущность «Оборудование» из списка сущностей. (Рис. 6.11)

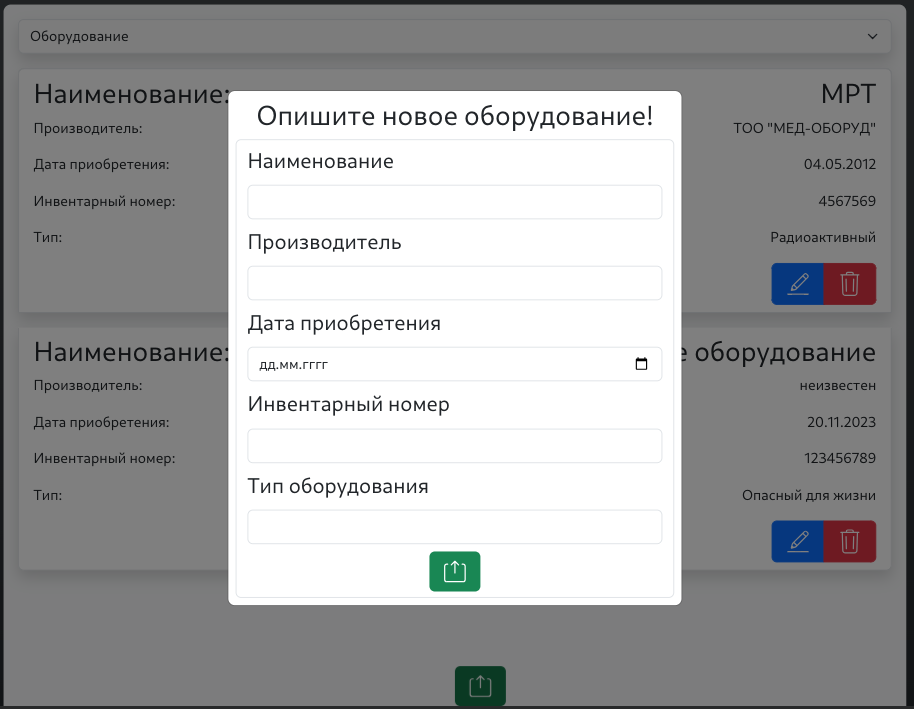


Рис. 6.12 – Форма создания оборудования

При нажатии зеленой кнопки внизу страницы выводится форма для создания оборудования. (Рис. 6.12)

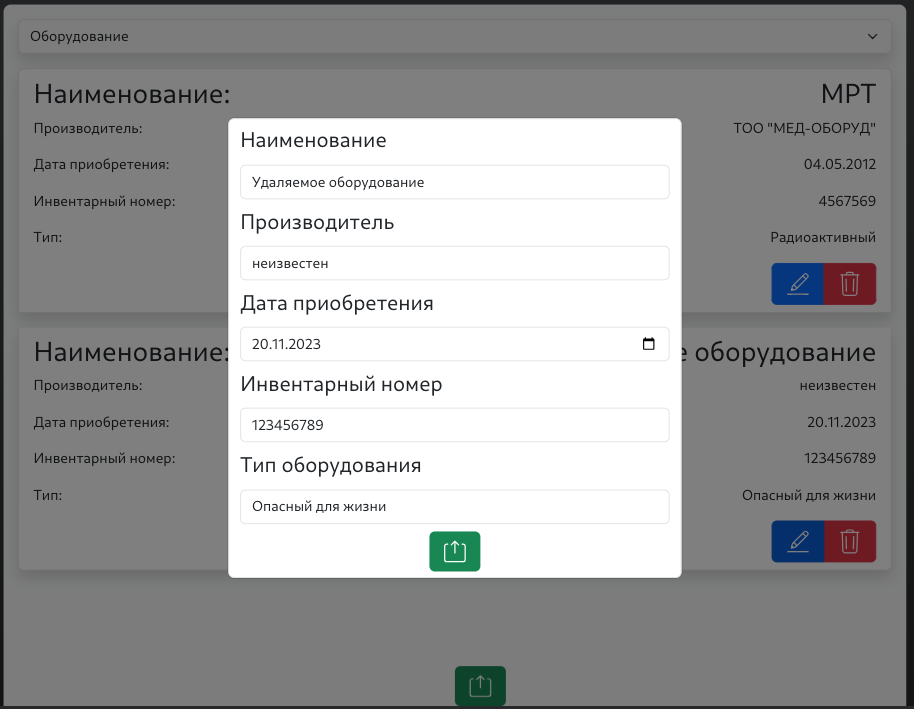


Рис. 6.13 – Форма редактирования оборудования

При нажатии синей кнопки на карточке оборудования открывается форма редактирования с заполненными полями. (Рис. 6.13)

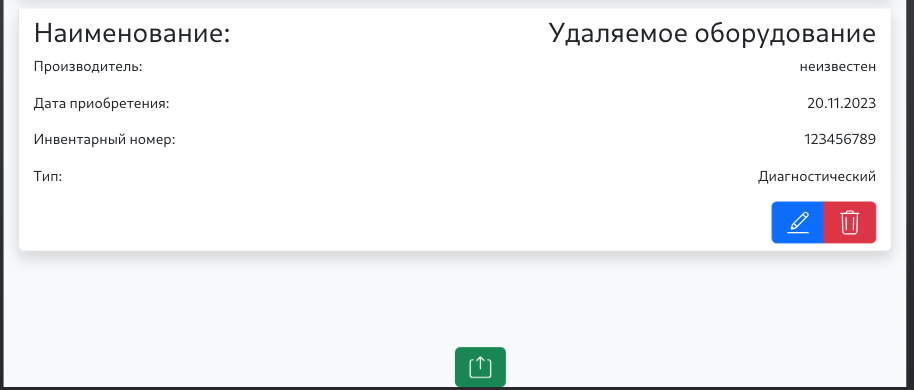


Рис. 6.14 – Оборудование после редактирования

После редактирования можно увидеть результат на карточке оборудования. (Рис. 6.14)

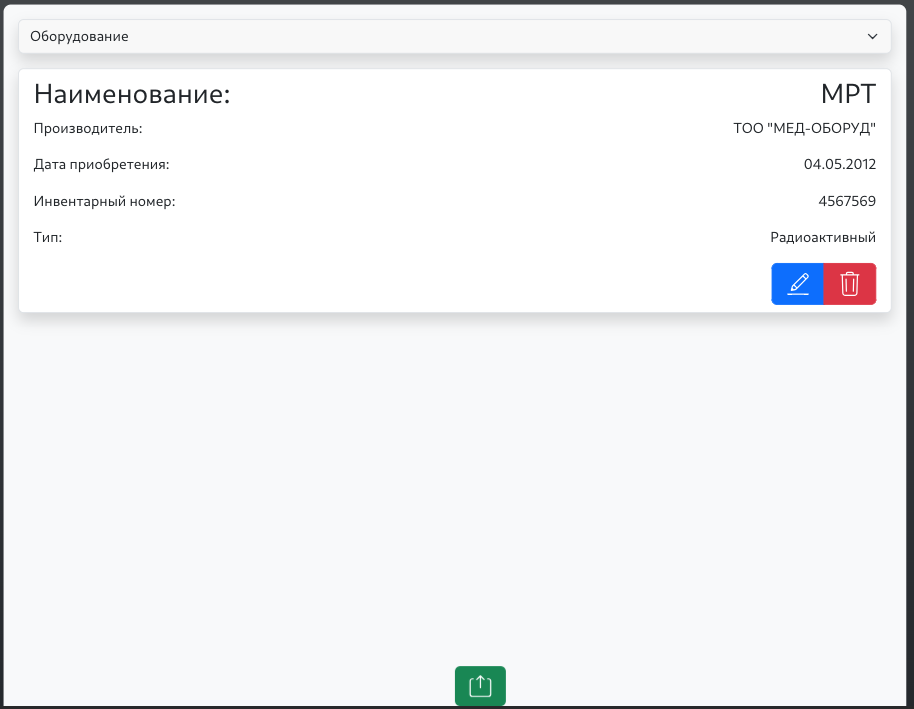


Рис. 6.15 – Удаление оборудования

После нажатия на красную кнопку последует удаление оборудования из системы. (Рис. 6.15)

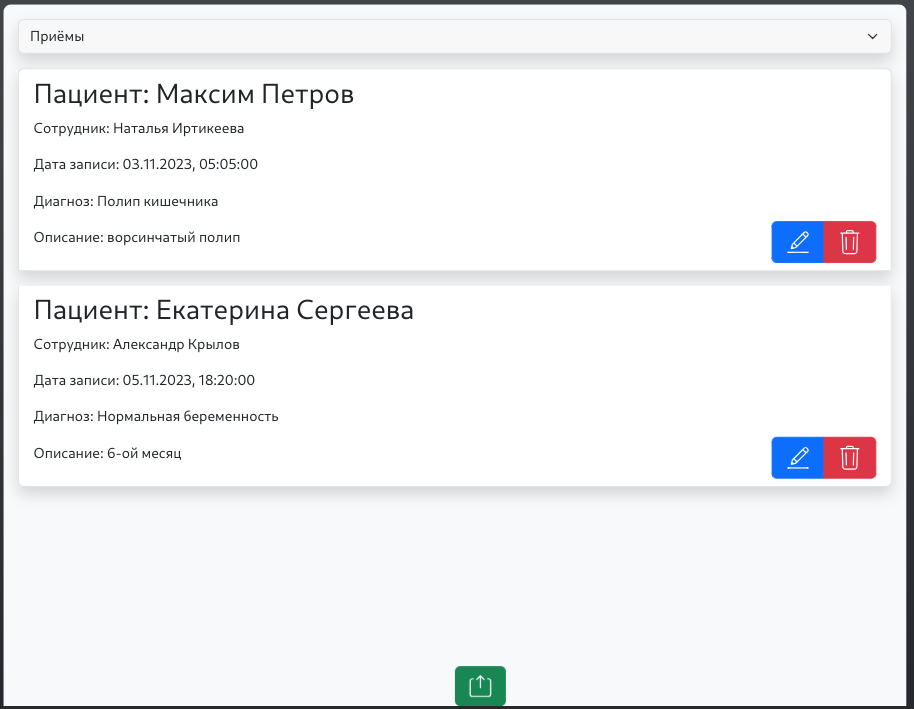


Рис. 6.16 – Выбор сущности «Приёмы»

Теперь из списка сущностей мы выбираем сущность «Приёмы». (Рис. 6.16)

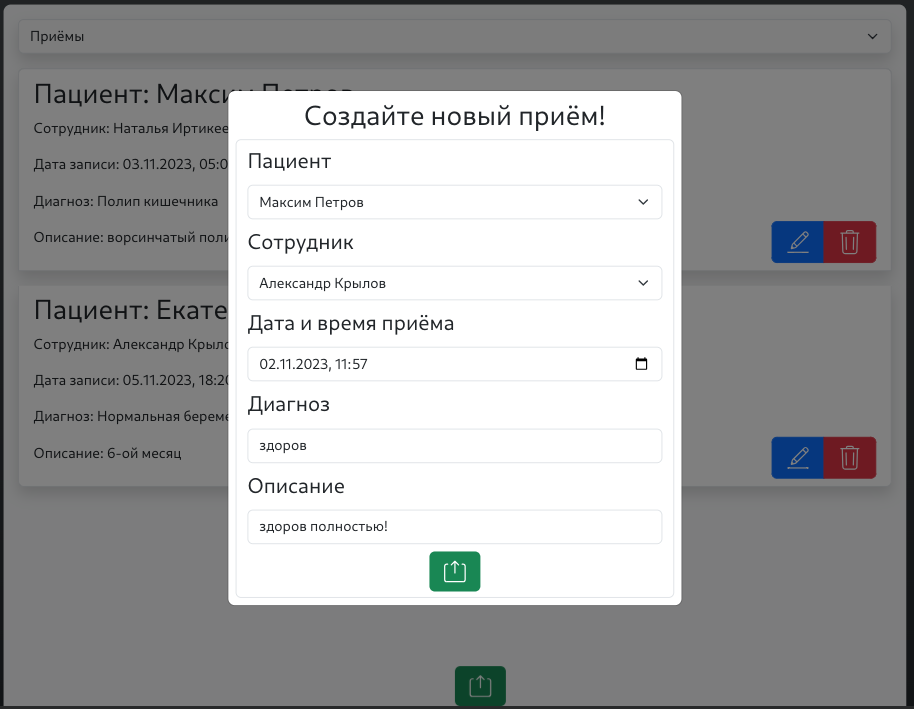


Рис. 6.17 – Создание тестового приёма

Создадим тестовый приём у Александра Крылова. (Рис. 6.17)

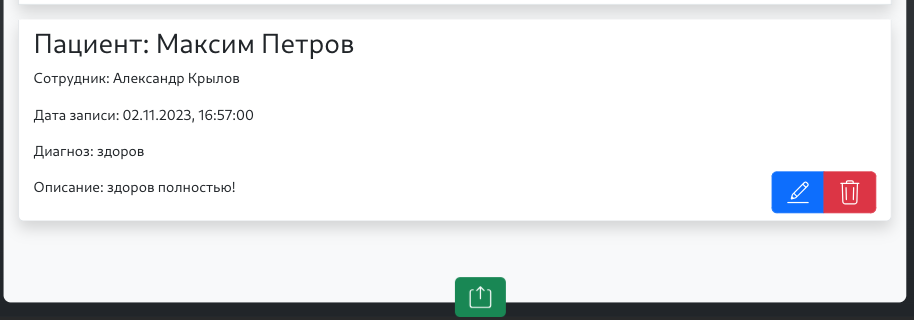


Рис. 6.18 – Созданный тестовый приём

После создания видим приём в общем списке. (Рис. 6.18)

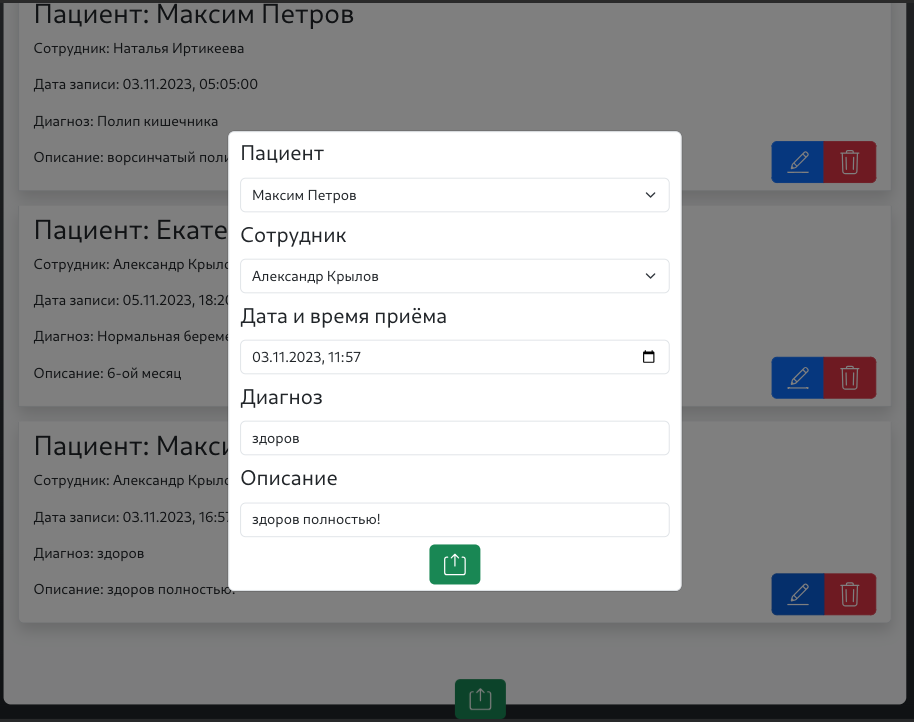


Рис. 6.19 – Редактирование тестового приёма

Теперь редактируем тестовый приём. Попробуем поменять дату приёма. (Рис. 6.19)

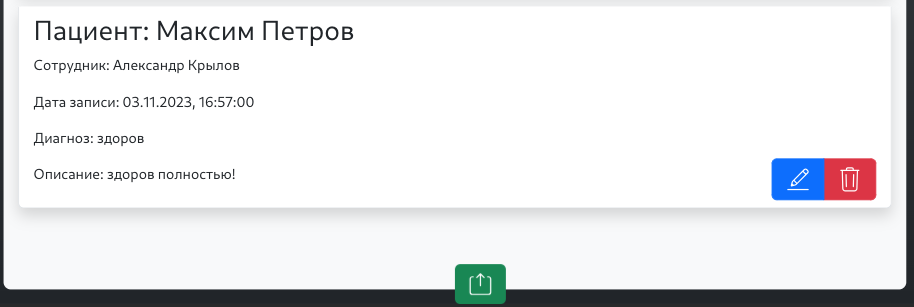


Рис. 6.20 – Отредактированный приём

После редактирования мы можем убедиться, что дата записи действительно изменилась. (Рис. 6.20)

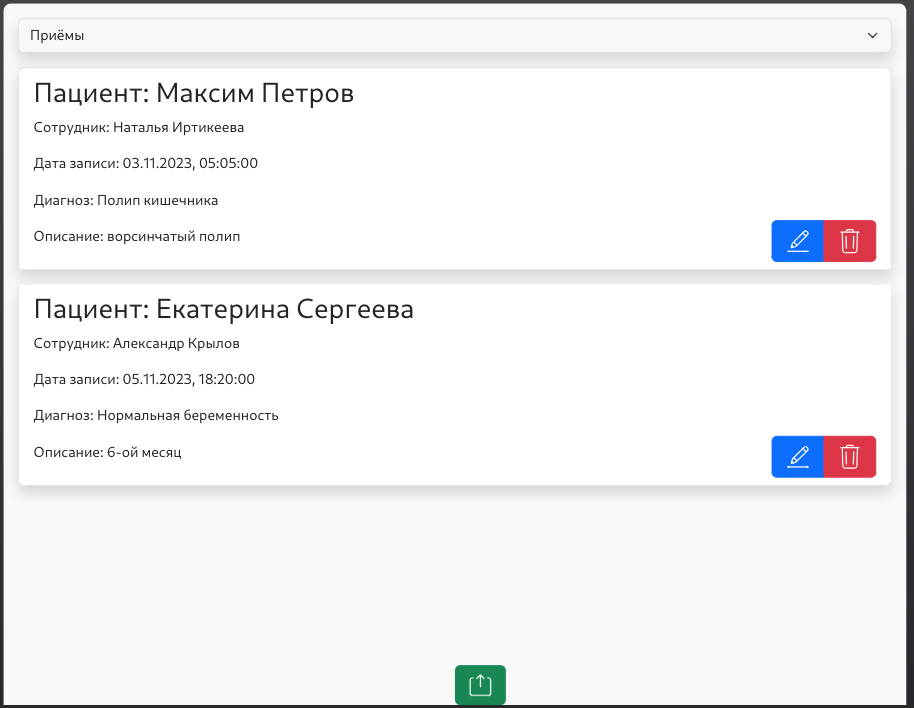


Рис. 6.21 – Удаление тестового приёма

После того, как мы создали и отредактировали наш тестовый приём, благополучно его удаляем. (Рис. 6.21)

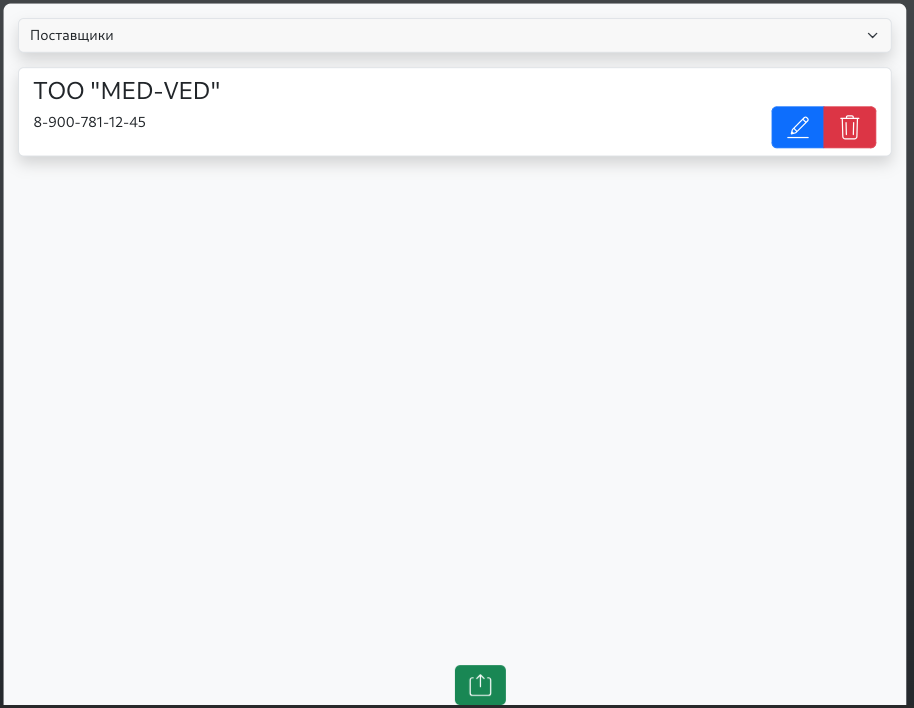


Рис. 6.22 – Выбор сущности «Поставщики»

После «Приёмы» выбираем сущность «Поставщики» из списка. (Рис. 6.22)

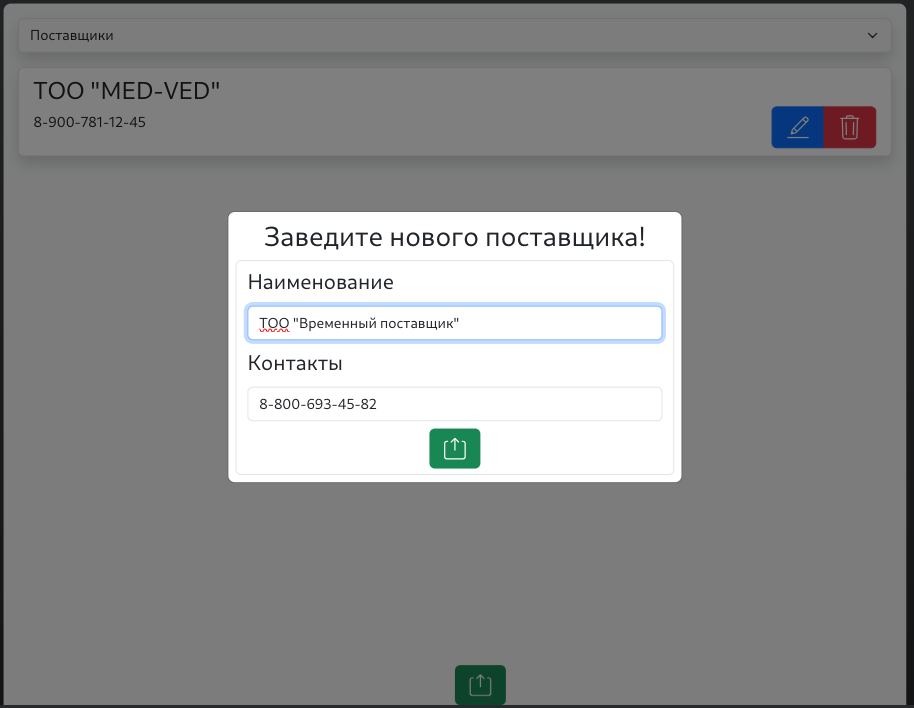


Рис. 6.23 – Создание временного поставщика

Создадим поставщика, который будет представлять временную запись. (Рис. 6.23)

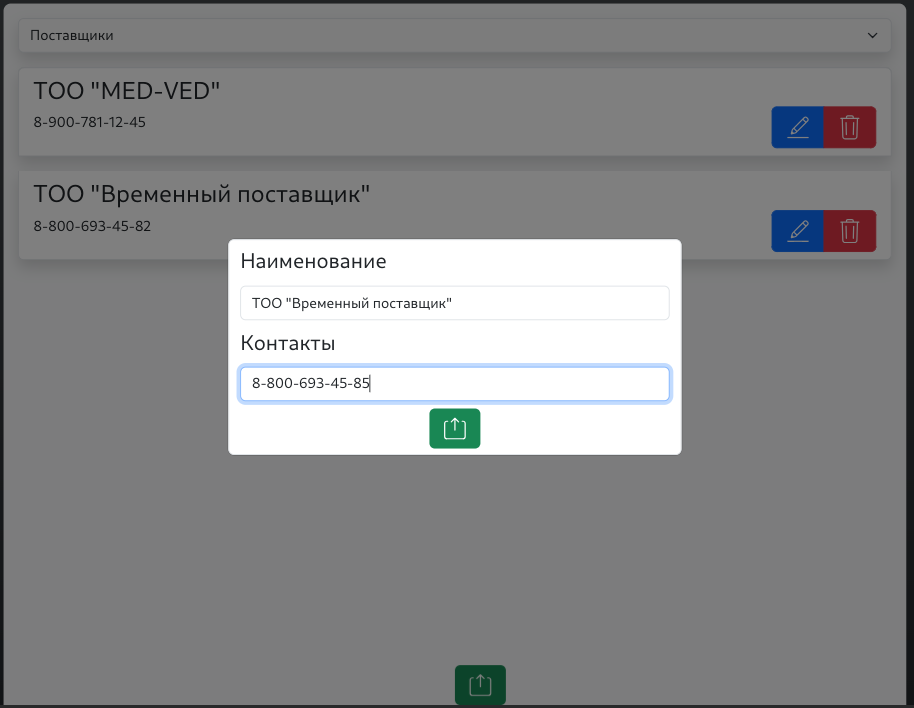


Рис. 6.24 – Редактирование временного поставщика

После создания сразу же редактируем его. Изменим его номер. (Рис. 6.24)

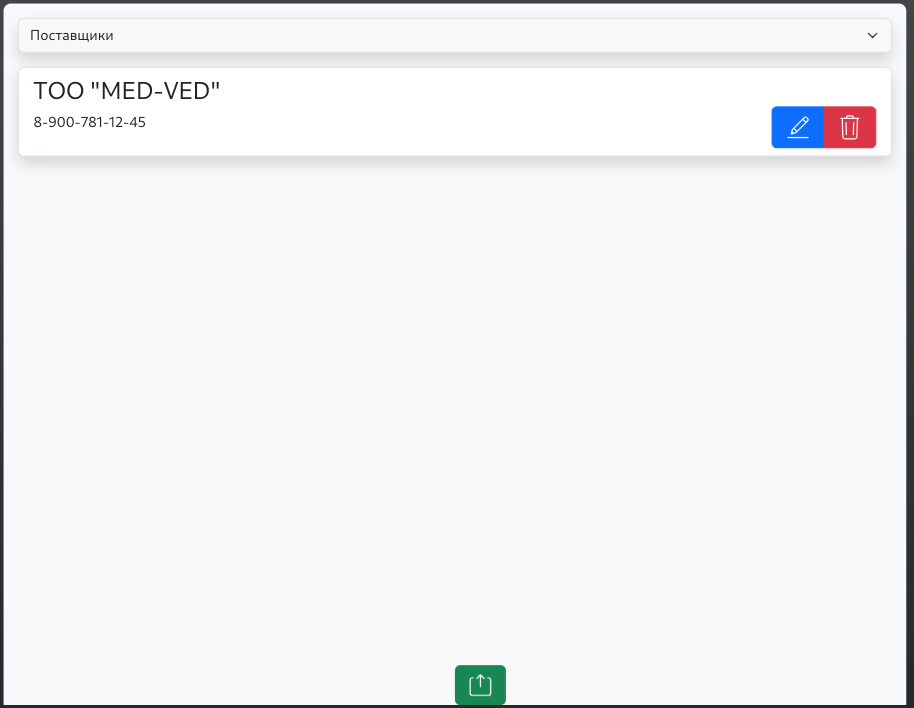


Рис. 6.25 – Удаление временного поставщика

После редактирования успешно удаляем нашего временного поставщика из общего списка. (Рис. 6.25)

# Список используемых источников

1. Джесси Шоу (Jesse Show) - "Разработка с использованием React: от новичка до профессионала", Техностиль, 2020, 320 стр. (ориг. название: "React Development: From Novice to Professional", O'Reilly)
2. Эрик Майер (Eric Mayer) - "Изучаем JavaScript: полное руководство", БХВ-Петербург, 2019, 480 стр. (ориг. название: "Learning JavaScript: The Complete Guide", Wiley)
3. Джошуа Блох (Joshua Bloch) - "Java. Эффективное программирование", Диалектика, 2019, 352 стр. (ориг. название: "Effective Java", Addison-Wesley)
4. Крейг Уоллс (Craig Walls) - "Spring в действии", Питер, 2017, 416 стр. (ориг. название: "Spring in Action", Manning Publications)
5. Джон Дакетт (Jon Duckett) - "HTML и CSS. Разработка и дизайн веб-сайтов", Вильямс, 2018, 736 стр. (ориг. название: "HTML and CSS: Design and Build Websites", Wiley)
6. Райан Бакер (Ryan Baker) - "PostgreSQL: справочник администратора", ДМК Пресс, 2020, 432 стр. (ориг. название: "PostgreSQL: Up and Running", O'Reilly)

# **Приложение А – Листинг программного кода**

import React from "react";  
import Error from "../util/Error";  
import Loading from "../util/Loading";  
import CircumIcon from "@klarr-agency/circum-icons-react";  
import StaffForm from "../form/staff/StaffForm";  
import FixedSaveBtn from "../util/FixedSaveBtn";  
import AppointmentForm from "../form/AppointmentForm";

export default class Appointment extends React.Component {  
 constructor(props) {  
 super(props);  
 this.state = {  
 isLoaded: false  
 }  
 }  
 componentDidMount() {  
 this.fetchItems().then(  
 json => {  
 let promises = [];

promises.push(...json.\_embedded.appointments.map(i=>this.fetchItemData(i)));  
 Promise.all(promises)  
 .then(resp=>{  
 console.log(resp);  
 this.setState({  
 items: [  
 ...resp  
 ]  
 },()=>{  
 console.log(`state: ${JSON.stringify(this.state)}`);  
 this.setState(prev=>({...prev, isLoaded: true}))  
 })  
 })  
 }  
 )  
 .catch(error => this.setState({error}));  
 }  
  
 fetchItems = async () => {  
 return await fetch("http://localhost:8080/appointments")  
 .then(resp => {  
 if (!resp.ok) {  
 throw new Error({message: JSON.stringify(resp.body)});  
 }  
 return resp.json();  
 })  
  
 }  
  
 fetchItemData = (item)=> {  
 let \_item = {  
 ...item  
 };  
 return Promise.all([  
 fetch(\_item.\_links.patient.href)  
 .then(resp => resp.json()),  
 fetch(\_item.\_links.staff.href)  
 .then(resp => resp.json())  
 ])  
 .then(resp => {  
 \_item.patient = {...resp[0]}  
 \_item.staff = {...resp[1]}  
 return \_item;  
 })  
 .catch(error => this.setState(error));  
 }  
  
 render() {  
  
 let {items, isLoaded, error} = this.state;  
 const patients = [], stuff=[];  
   
 if (error) {  
 return <Error message={`${error.message}`}/>  
 } else if (!isLoaded) {  
 return <Loading/>  
 }  
  
 return (  
 <>  
 <ul className="list-group min-vh-100">  
 {  
 items.map(  
 i => {  
 return (  
 <li className="list-group-item d-flex justify-content-between mt-3 shadow"  
 key={i.\_links.self.href}>  
 <div className="">  
 <h2>  
 Пациент: {i.patient.firstname} {i.patient.lastname}  
 </h2>  
 <p>  
 Сотрудник: {i.staff.firstname} {i.staff.lastname}  
 </p>  
 <p>  
 Дата записи: {new Date(Date.parse(i.timestamp)).toLocaleString()}  
 </p>  
 <p>  
 Диагноз: {i.diagnosis}  
 </p>  
 <p>  
 Описание: {i.description}  
 </p>  
 </div>  
 <div className="btn-group align-self-end">  
 <button data-bs-target={"#update-appointment-form" + i.\_links.self.href.split('/')[i.\_links.self.href.split('/').length-1]}  
 data-bs-toggle="modal"  
 className="btn btn-primary"  
 >  
 <CircumIcon name="edit"/>  
 </button>  
 <button className="btn btn-danger" onClick={() => {  
 fetch(i.\_links.self.href, {  
 method: "DELETE"  
 })  
 .then(resp => {  
 return this.fetchItems()  
 })  
 .then(json => {  
 this.setState({  
 items: [...json.\_embedded.appointments],  
 isLoaded: true  
 })  
 })  
 .catch(error => this.setState({error}))  
 }}>  
 <CircumIcon name="trash"/>  
 </button>  
 </div>  
 <div className="modal fade" id={"update-appointment-form" + i.\_links.self.href.split('/')[i.\_links.self.href.split('/').length-1]}>  
 <div className="modal-dialog-centered modal-dialog">  
 <div className="modal-content">  
 <AppointmentForm  
 initData={i} onSubmit={  
 data => {  
 fetch(i.\_links.self.href, {  
 method: "PATCH",  
 headers: {  
 "content-type": "application/json"  
 },  
 body: JSON.stringify(data)  
 })  
 .catch(error => this.setState({error}))  
 }  
 }  
 />  
 </div>  
 </div>  
 </div>  
 </li>  
 )  
 }  
 )  
 }  
 </ul>  
 <div data-bs-toggle="modal" data-bs-target="#create-appointment-form">  
 <FixedSaveBtn/>  
 </div>  
 <div className="modal fade" id="create-appointment-form">  
 <div className="modal-dialog modal-dialog-centered">  
 <div className="modal-content p-2">  
 <h2 className="text-center">Создайте новый приём!</h2>  
 <AppointmentForm onSubmit={(data => {  
 fetch("http://localhost:8080/appointments",  
 {  
 method: "POST",  
 headers: {  
 "Content-Type": "application/json"  
 },  
 body: JSON.stringify(data)  
 }  
 ).then(resp => {  
 return this.fetchItems()  
 })  
 .then(json => {  
 this.setState({  
 items: [...json.\_embedded.appointments],  
 isLoaded: true  
 })  
  
 })  
 .catch(error => this.setState({error}))  
 })}/>  
 </div>  
 </div>  
 </div>  
 </>  
 );  
 }  
}

import React from "react";  
import Error from "../util/Error";  
import Loading from "../util/Loading";  
import CircumIcon from "@klarr-agency/circum-icons-react";  
import FixedSaveBtn from "../util/FixedSaveBtn";  
import EquipmentForm from "../form/EquipmentForm";  
export default class Equipment extends React.Component{  
 constructor(props) {  
 super(props);  
 this.state = {  
 isLoaded: false  
 }  
 }  
  
 componentDidMount() {  
 this.fetchItems().then(json => {  
 this.setState({  
 items: [  
 ...json.\_embedded.equipments  
 ]  
 }, () => {  
 ***console***.log(`state: ${***JSON***.stringify(this.state)}`);  
 this.setState(prev => ({...prev, isLoaded: true}))  
 })  
 }  
 ).catch(error => this.setState({error}));  
 }  
}