**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**

[1 Описание предметной области. Актуальность 5](#_Toc1)

[1.1 Введение 5](#_Toc2)

[1.2 Описание предметной области 5](#_Toc3)

[1.3 Актуальность проекта 5](#_Toc4)

[1.4 Вывод 6](#_Toc5)

[2 Описание программы. Общие сведения 7](#_Toc6)

[3 Функциональное назначение 7](#_Toc7)

[3.1 Описание функционального назначения 7](#_Toc8)

[3.2 Основные функции приложения 7](#_Toc9)

[3.3 Описание технологий 8](#_Toc10)

[4 Логическая модель базы данных 9](#_Toc11)

[4.1 Сущности и атрибуты 9](#_Toc12)

[4.2 Связи сущностей 10](#_Toc13)

[5 Физическая модель базы данных 11](#_Toc14)

[6 Диаграмма классов 13](#_Toc15)

[7 Диаграмма компонентов 15](#_Toc16)

[8 Диаграмма вариантов использования 16](#_Toc17)

[9 Диаграмма последовательности 17](#_Toc18)

[10 Демонстрация работы приложения 18](#_Toc19)

[Список используемых источников 31](#_Toc20)

[Приложение А – Листинг программного кода 32](#_Toc21)

# 1 Описание предметной области. Актуальность

## 1.1 Введение

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.

## 1.2 Описание предметной области

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.

## 1.3 Актуальность проекта

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.

## 1.4 Вывод

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.

# 2 Описание программы. Общие сведения

Таблица 1. Общие сведения

|  |  |
| --- | --- |
| **Параметр** | **Значение** |
| Тип приложения | Web-приложение |
| Языки программирования | Java, JavaScript |
| Фреймворки | Spring Boot, React.js |
| Среда разработки | Intellij IDEA/VS Code |
| Размер проекта | 458 мб |
| Количество строк кода | 199 (Java) + 1498 (JavaScript) = 1697 |

# 3 Функциональное назначение

## 3.1 Описание функционального назначения

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.

## 3.2 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.

## 3.3 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.

# 4 Логическая модель базы данных

## 4.1 Сущности и атрибуты

**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

## 4.2 Связи сущностей

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).

# 5 Физическая модель базы данных

Физическая модель базы данных (БД) описывает, как структуры данных, созданные в логической модели, будут реализованы на физическом уровне, включая хранение данных, индексы, ключи и другие аспекты, связанные с физическим размещением информации. В контексте нашего проекта по учету медицинского оборудования в медицинских учреждениях, физическая модель включает следующие аспекты:

* **Таблицы**: Каждая сущность из логической модели представлена таблицей в физической модели.
* **Столбцы**: Атрибуты каждой сущности в логической модели становятся столбцами в соответствующих таблицах физической модели.
* **Ограничения**: Физическая модель включает ограничения целостности данных, такие как первичные ключи (Primary Keys) для уникальной идентификации записей в таблицах, внешние ключи (Foreign Keys) для установления связей между таблицами, а также другие ограничения, определяющие правила сохранения целостности данных.
* **Индексы**: Для оптимизации производительности запросов физическая модель может включать индексы. Например, индекс можно создать для ускорения поиска по Инвентарному номеру в таблице "Медицинское оборудование".
* **Типы данных**: Для каждого столбца определены соответствующие типы данных (например, текстовый, числовой, дата), что определяет формат и характеристики данных.

Физическая модель проектируемой базы данных представлена на рис. 1.

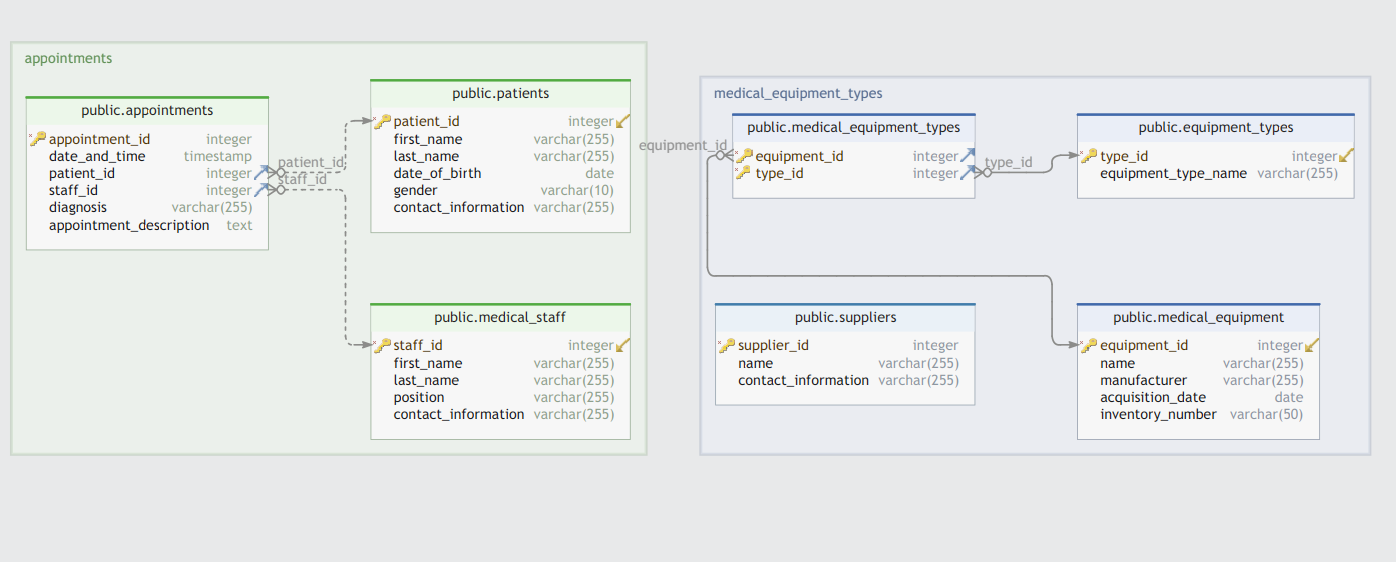


рис. 1 – Физическая модель базы данных

# 6 Диаграмма классов

Диаграмма классов – это UML-диаграмма (англ. Unified Modeling Language – рус. Унифицированный язык моделирования), которая описывает связи между классами.

Диаграмма классов представляет собой средство визуализации структуры системы, отражая классы, их атрибуты, методы и взаимосвязи.

На рис. 2.1, 2.2 представлены ключевые классы системы, их взаимодействие и структура.

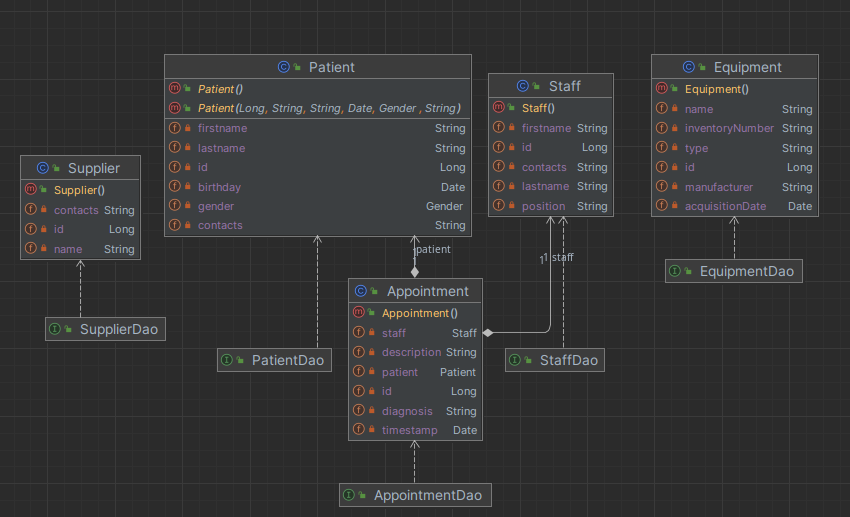


рис. 2.1 – Диаграмма Java классов



рис. 2.2 – Диаграмма JavaScript классов

# 7 Диаграмма компонентов

Диаграмма компонентов представляет из себя средство визуализации, которое позволяет получить представление об организации компонентов системы и взаимосвязей между ними.

На рис. 3 отражаются ключевые структурные элементы системы, включая фронтенд (Frontend), бэкенд (Backend), и базу данных (Database). Взаимодействие между компонентами и основные технологии, используемые в системе, такие как HTTP, Spring (Hibernate), и PostgreSQL, также представлены на диаграмме.

Описание компонентов:

* **Frontend**: Компонент, отвечающий за пользовательский интерфейс и взаимодействие с конечными пользователями. Работает на localhost:3000.
* **Backend**: Компонент, содержащий логику приложения и обработку запросов от Frontend. Работает на localhost:8080.
* **Database**: База данных, используемая системой для хранения данных. В данной диаграмме представлена PostgreSQL, работающая на localhost:5432.

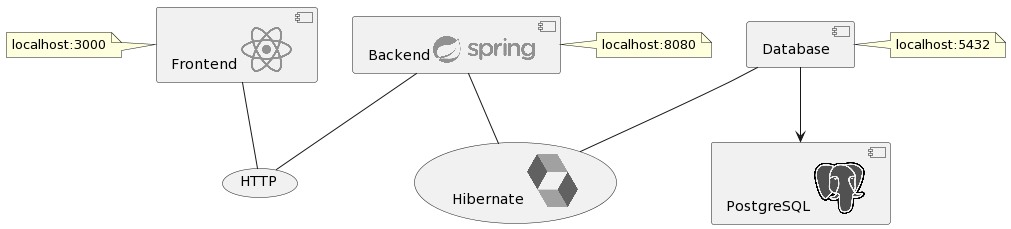


рис. 3 – Диаграмма компонентов системы.

# 8 Диаграмма вариантов использования

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 вариантов использования проектируемой системы представлена на рис. 4.

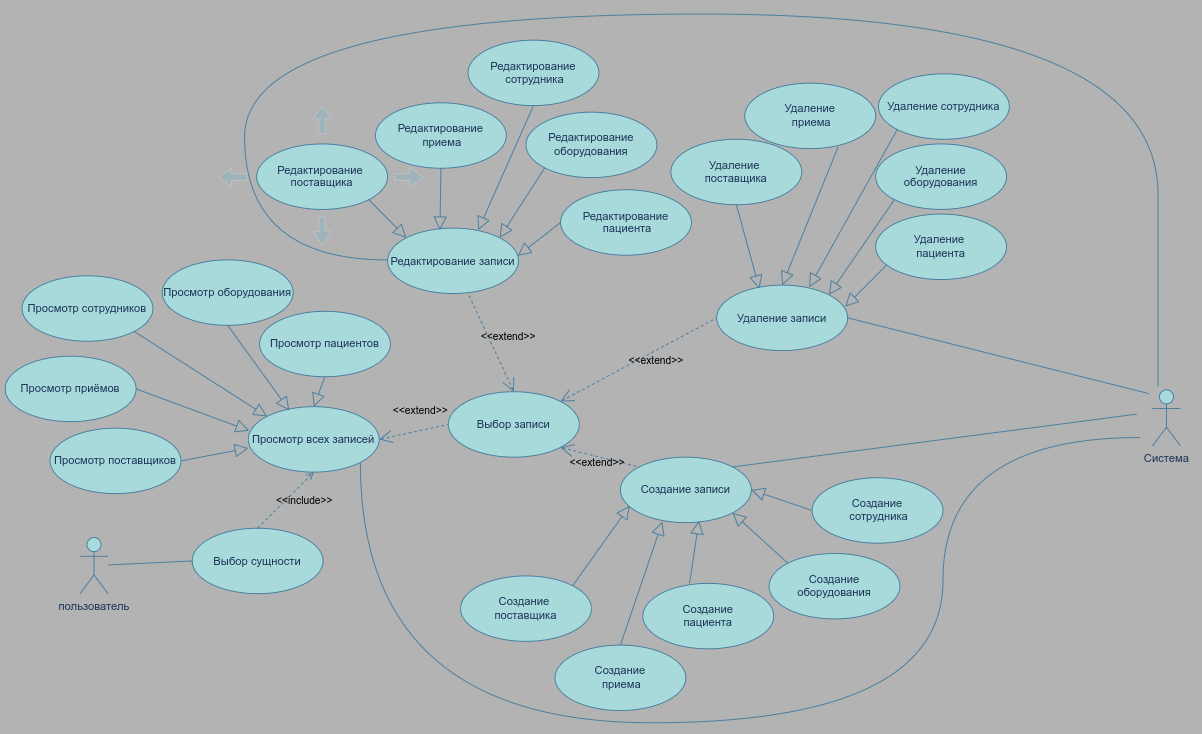


рис. 4 – Диаграмма вариантов использования системы

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.

# 9 Диаграмма последовательности

Диаграмма последовательности – это схема, иллюстрирующая как происходит взаимодействие различных частей системы друг с другом, также она показывает порядок выполнения определенных функций в проекте.

Диаграмма последовательности приведена на рис.

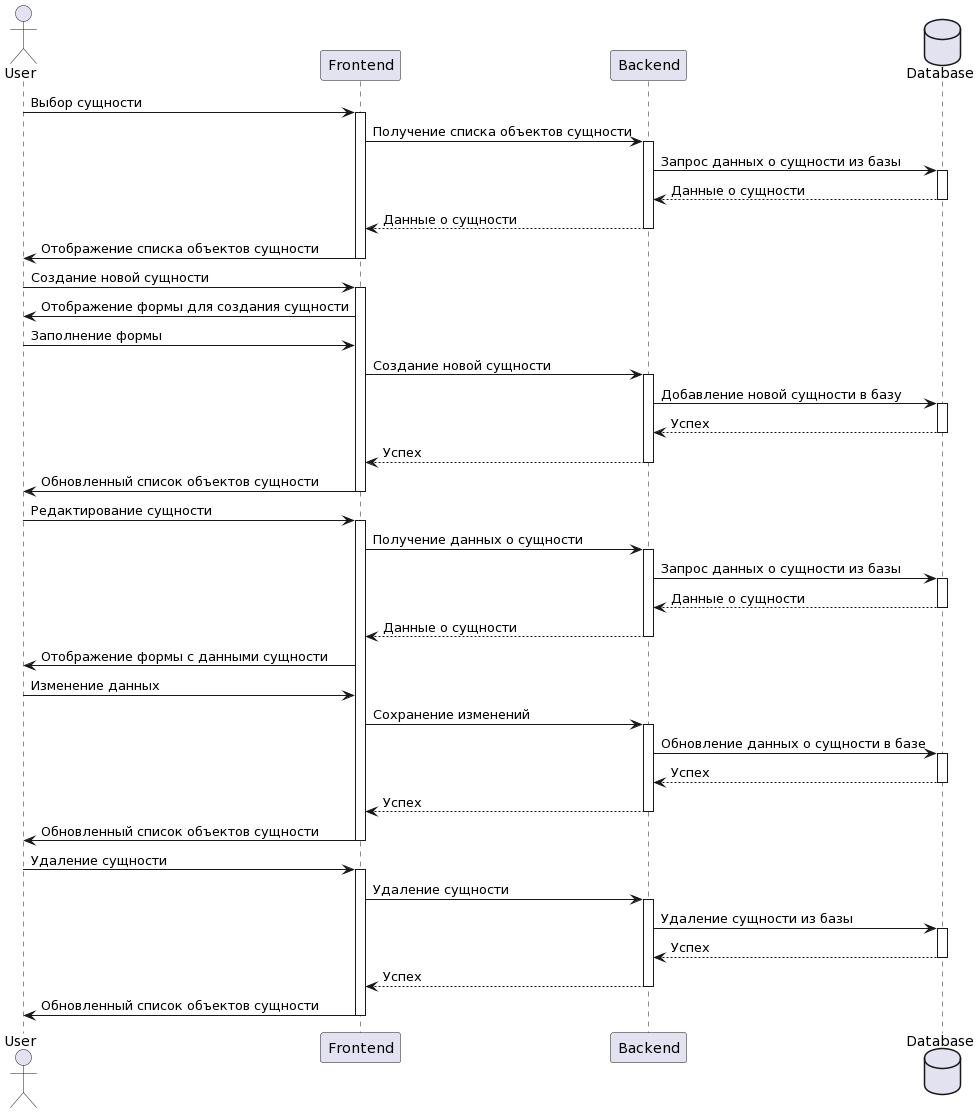


рис. 5 – Диаграмма последовательности

# 10 Демонстрация работы приложения

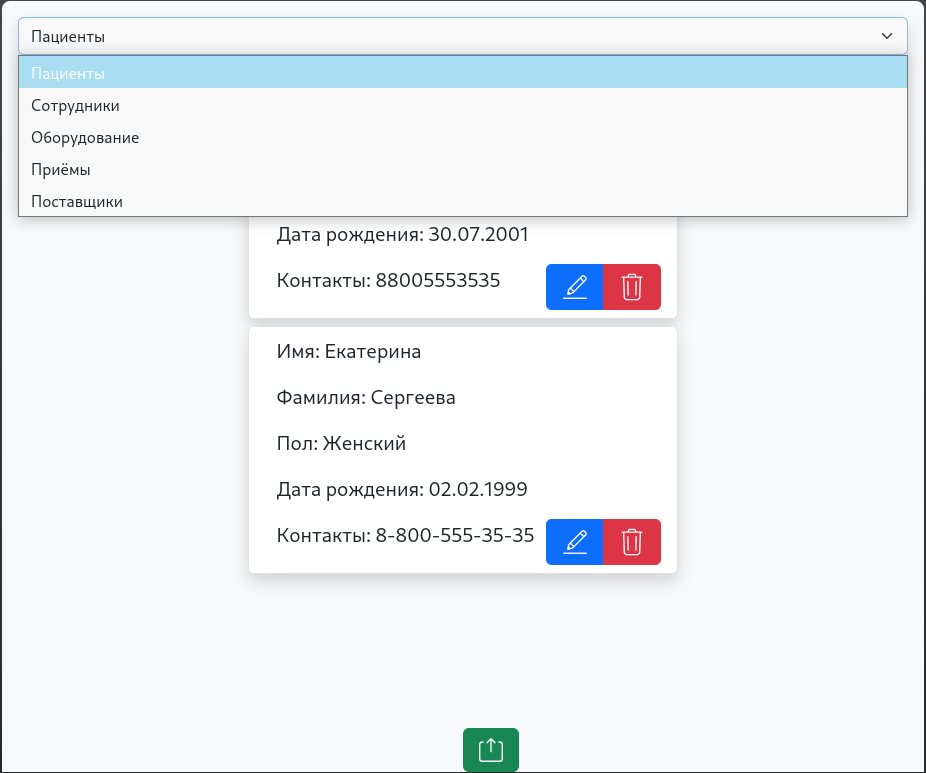


рис. 6.1 – Выбор сущности «Пациенты»

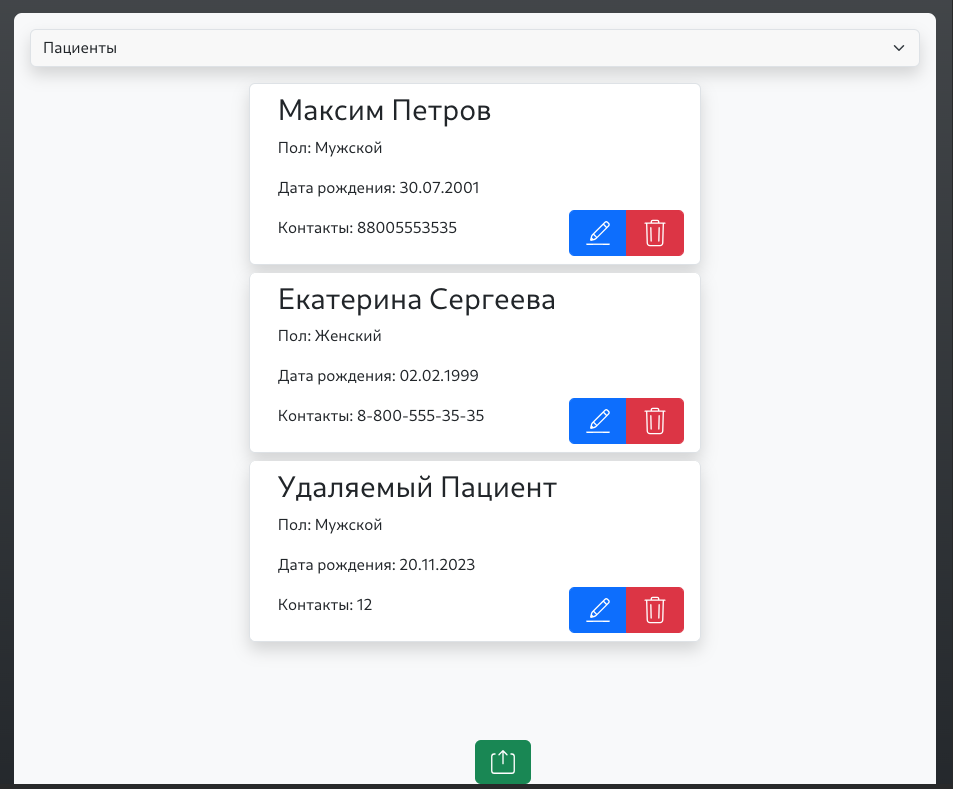


рис. 6.2 – Список всех пациентов

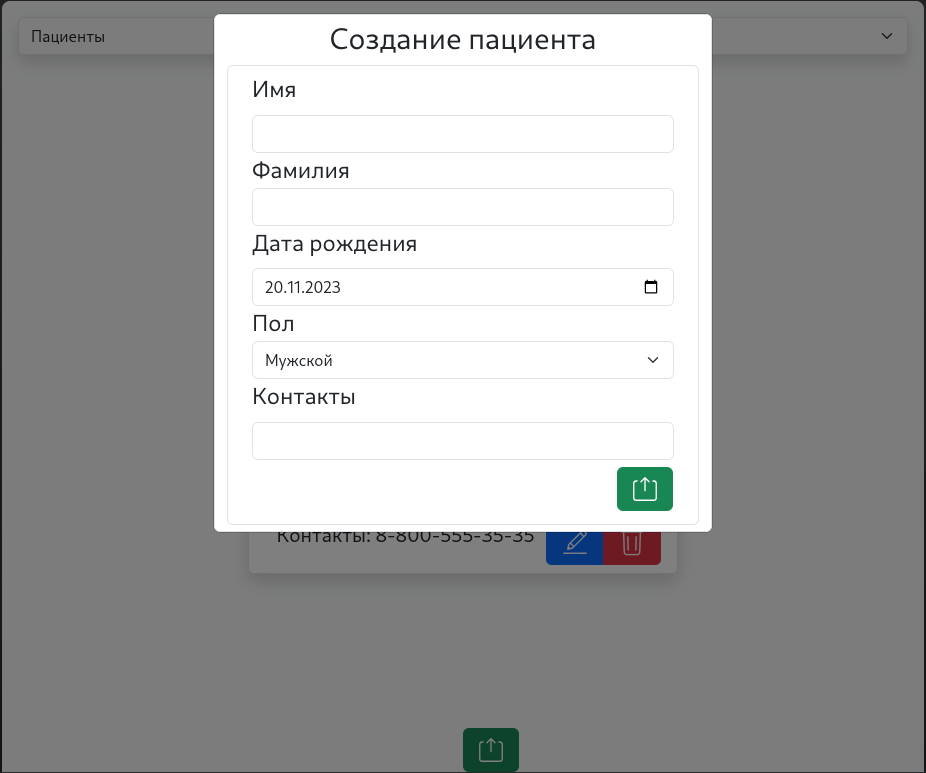


рис. 6.3 – Форма создания пациента

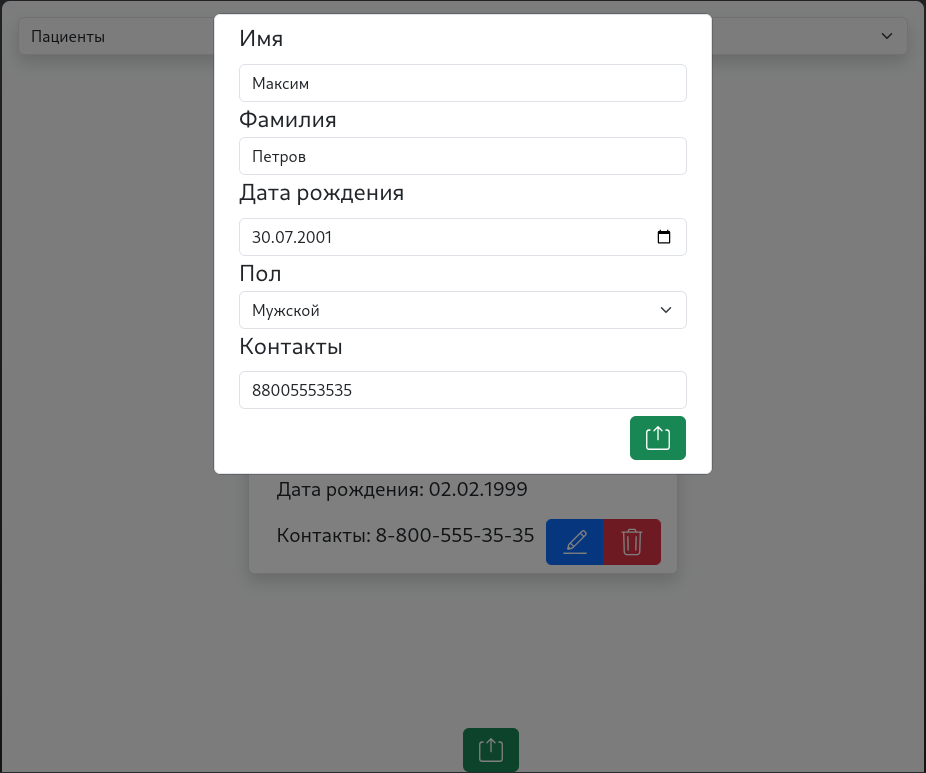


рис. 6.4 – форма редактирования пациента

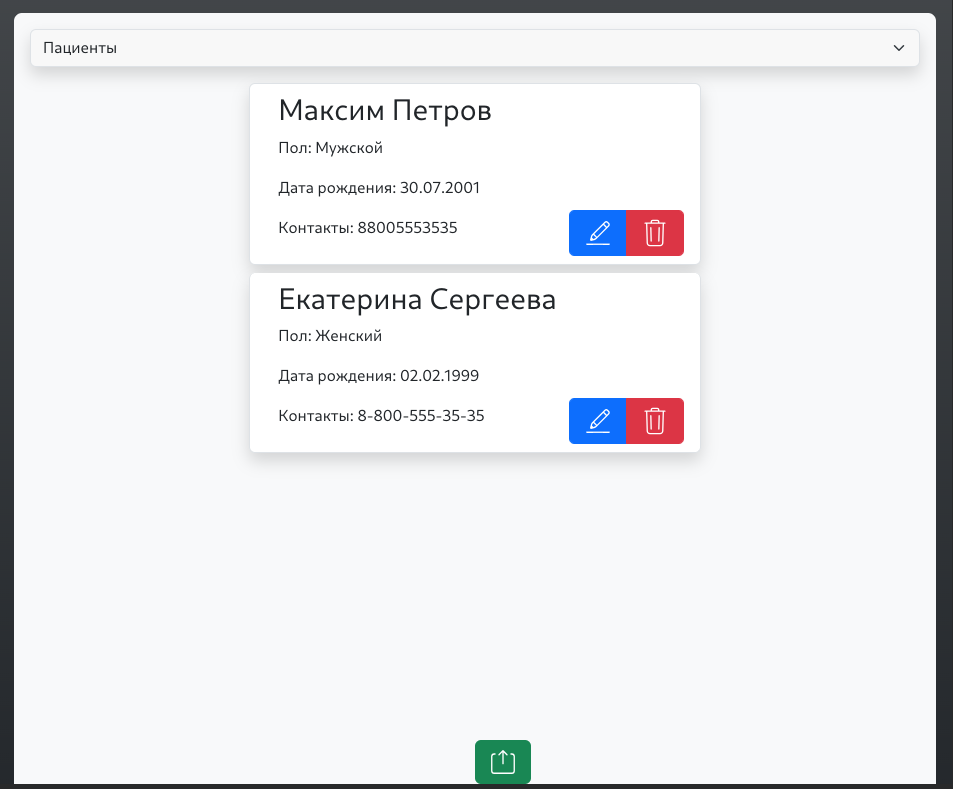


рис. 6.5 – удаление пациента

На рис. 6.5 представлено последствие удаления после нажатия на красную кнопку с иконкой мусорного бака.

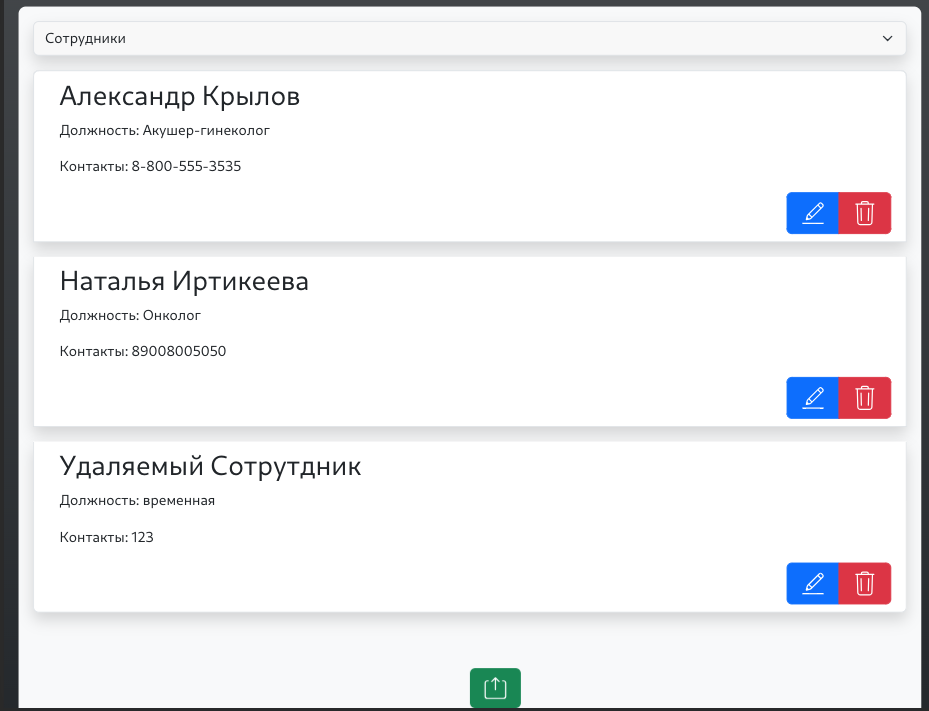


рис. 6.6 – Выбор сущности «Сотрудники»

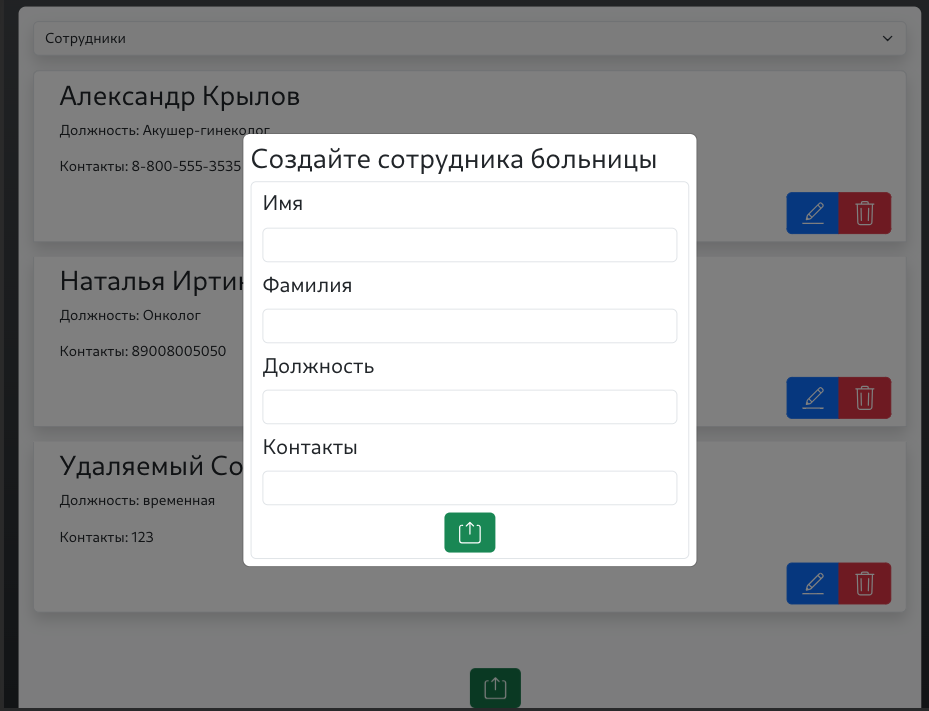


рис. 6.7 – Форма создания сотрудника

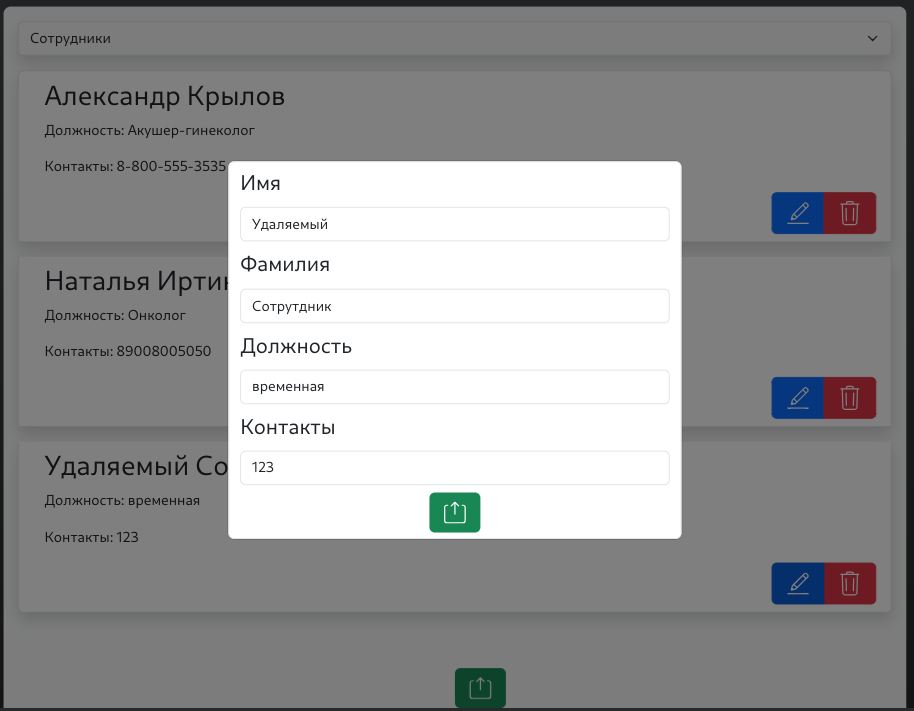


рис. 6.8 – Форма редактирования сотрудника

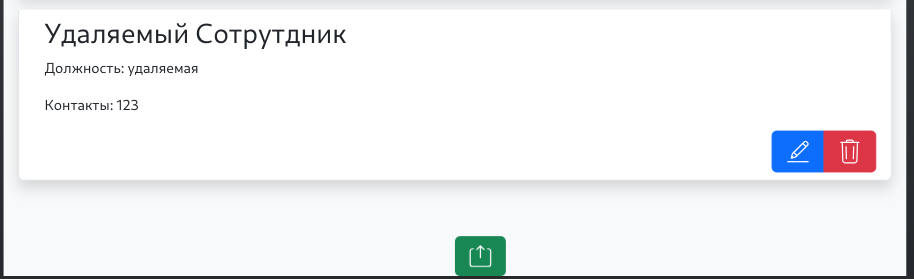


рис. 6.9 – Отредактированный удаляемый сотрудник

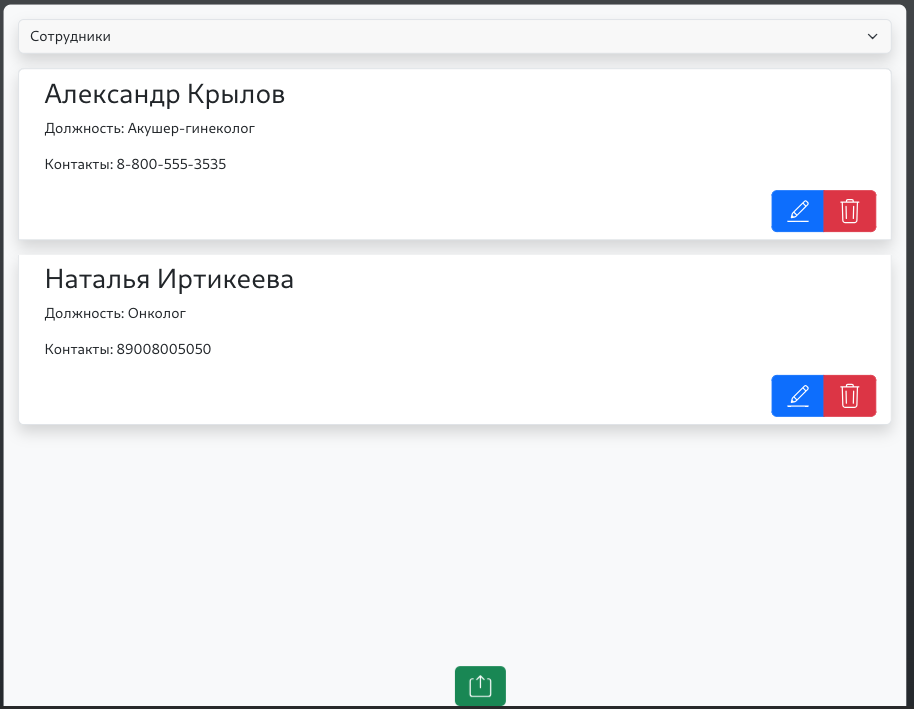


рис. 6.10 – Список сотрудников после удаления удаляемого сотрудника

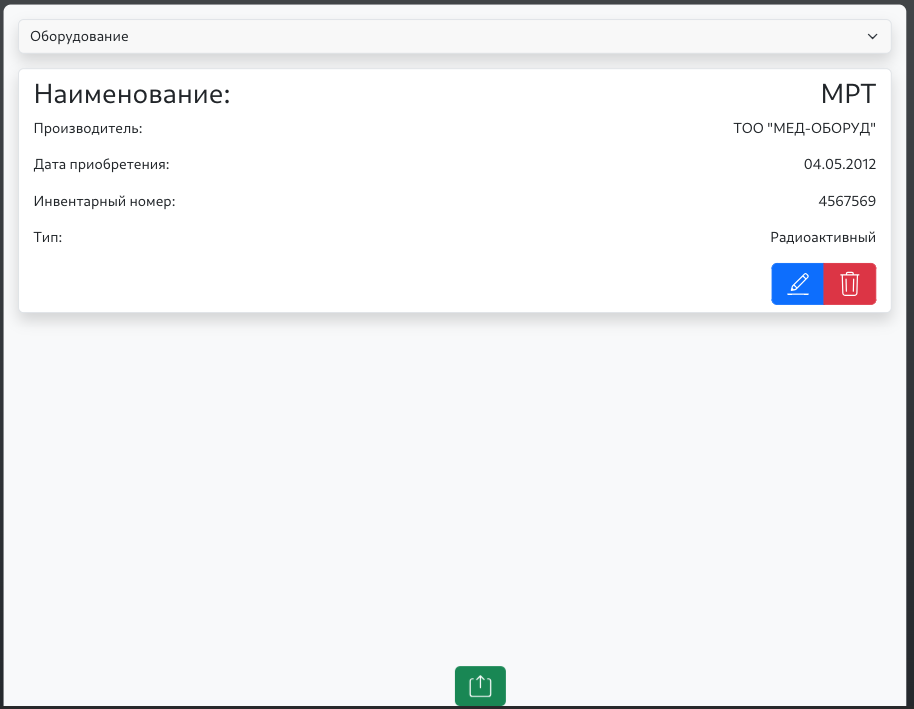


рис. 6.11- Выбор сущности «Оборудование»

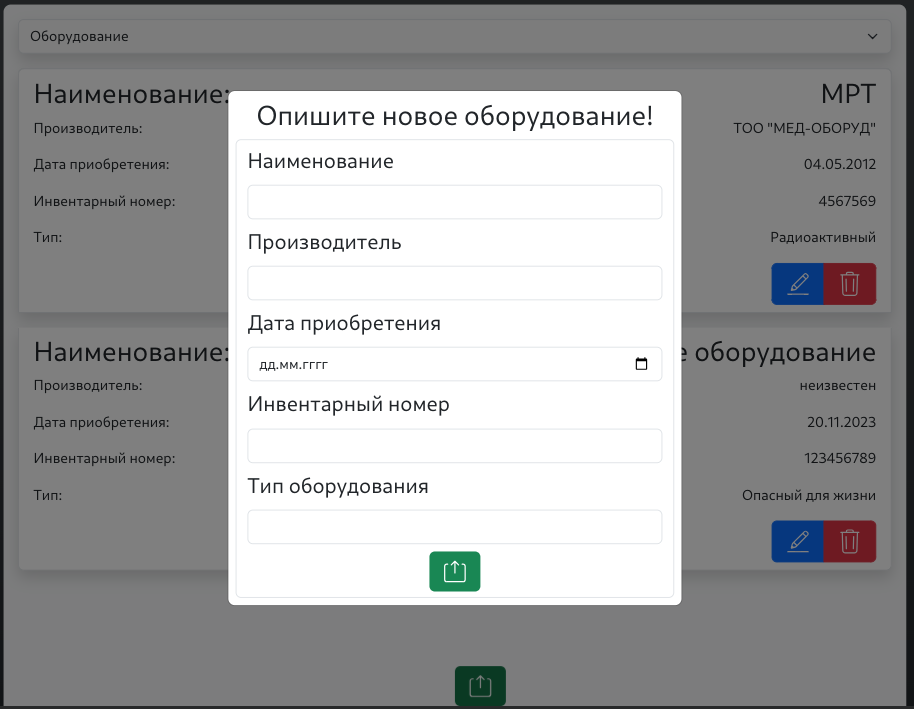


рис. 6.12 – Форма создания оборудования

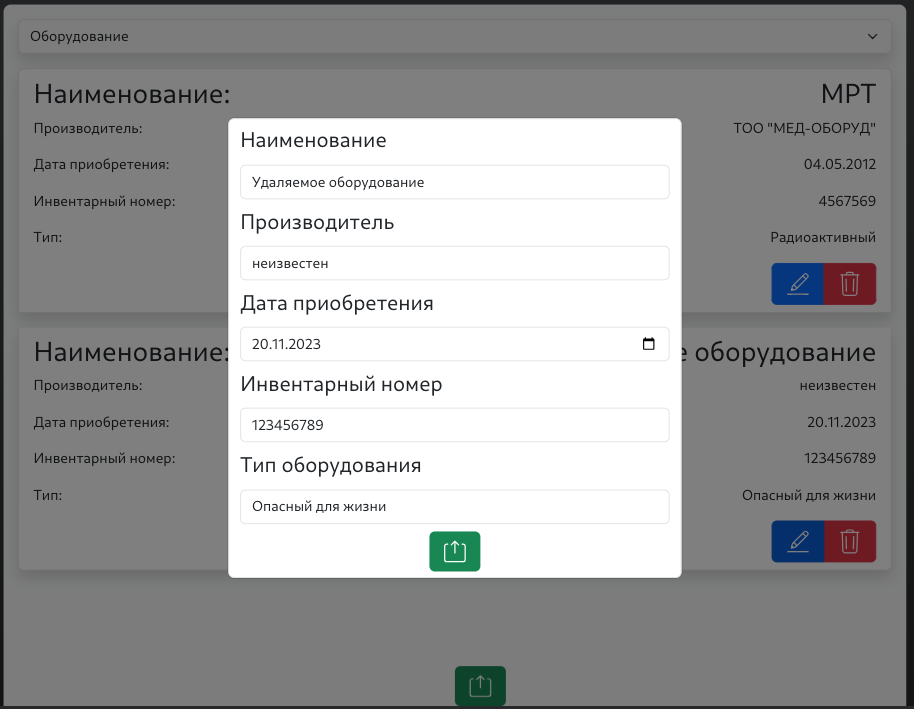


рис. 6.13 – Форма редактирования оборудования

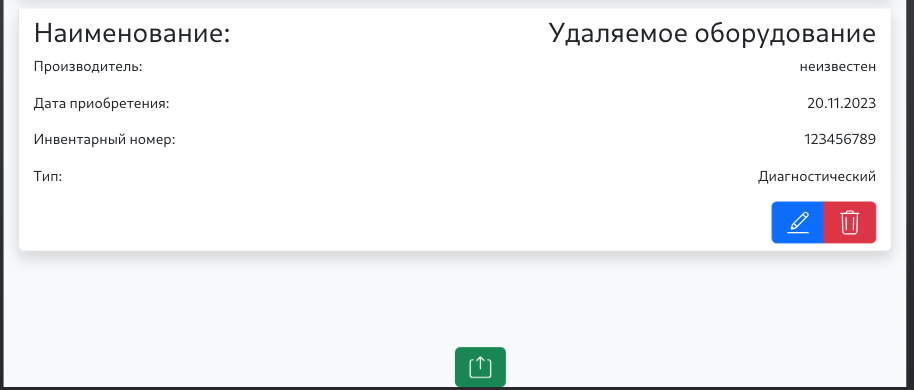


рис. 6.14 – Оборудование после редактирования

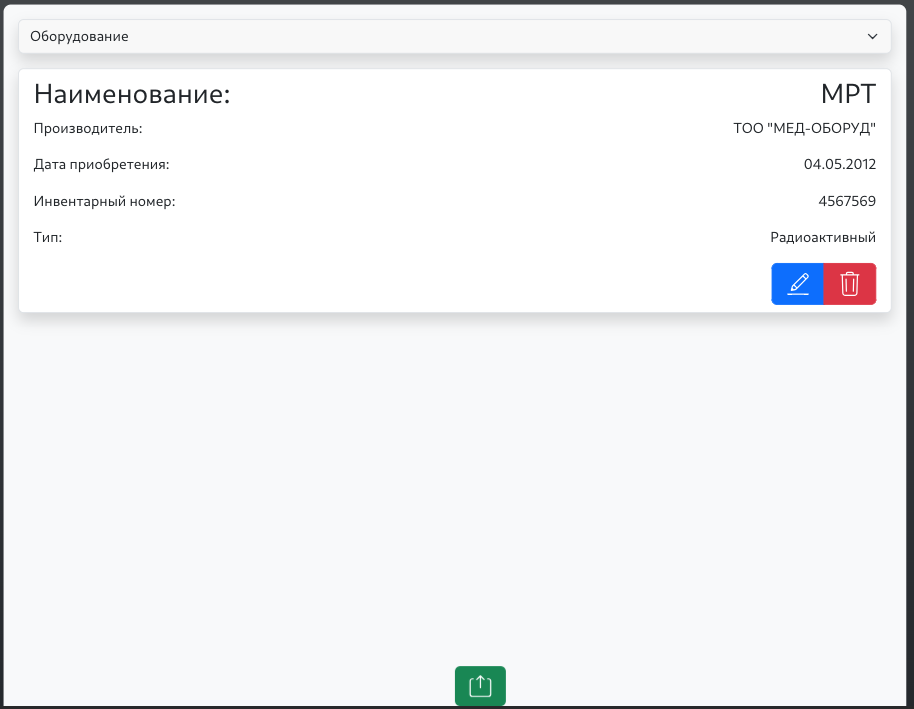


рис. 6.15 – Удаление оборудования

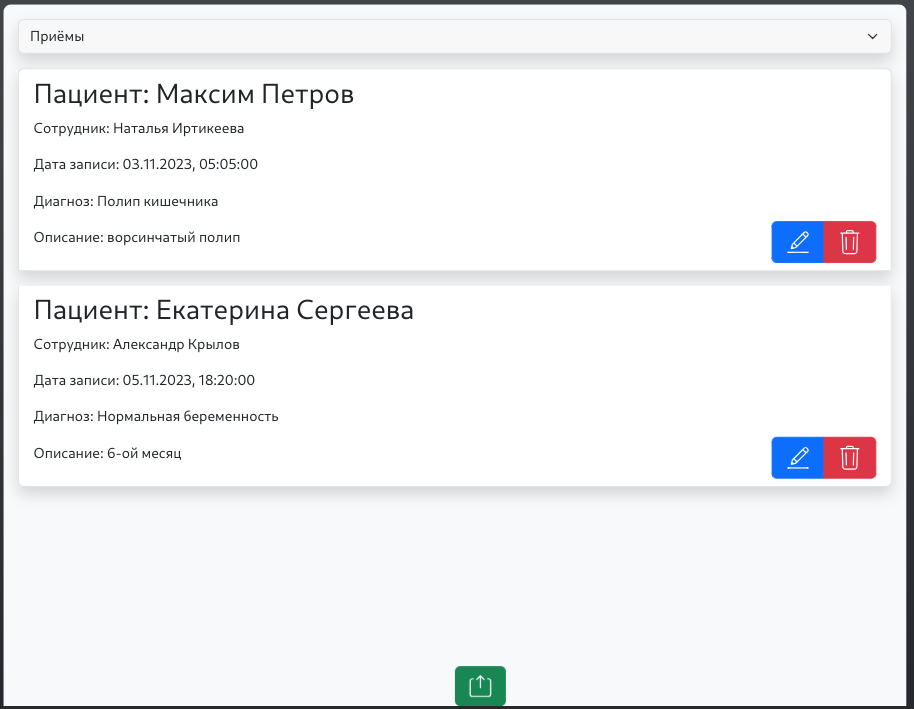


рис. 6.16 – Выбор сущности «Приёмы»

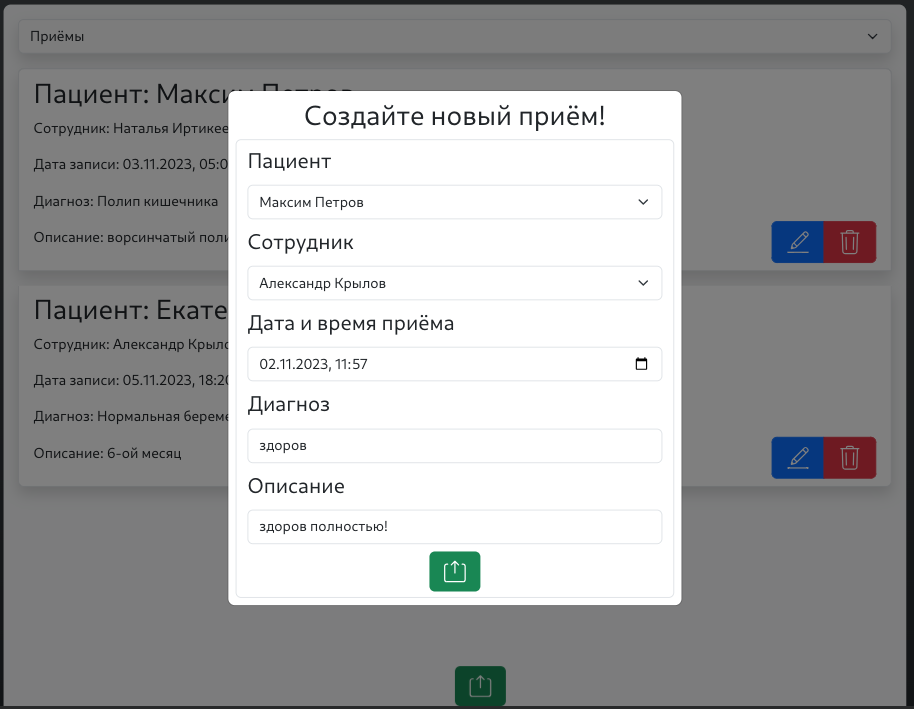


рис. 6.17 – Создание тестового приёма

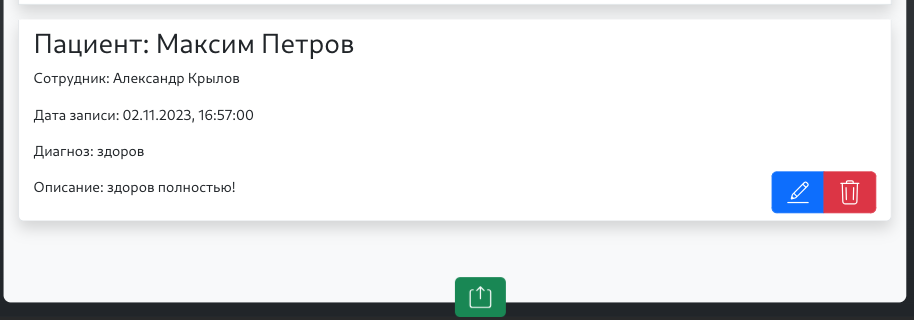


рис. 6.18 – Созданный тестовый приём

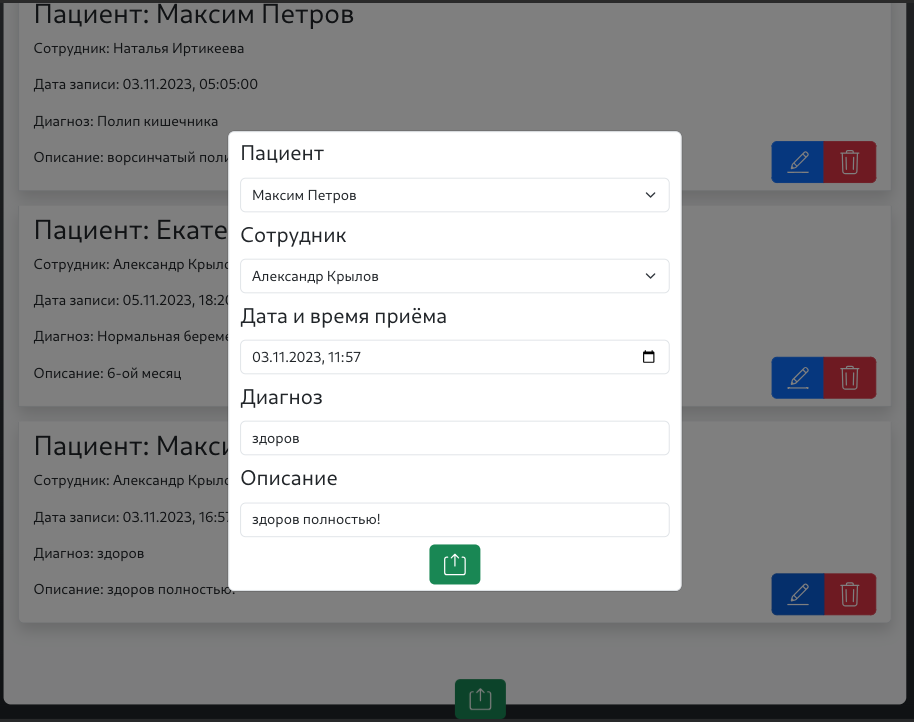


рис. 6.19 – Редактирование тестового приёма

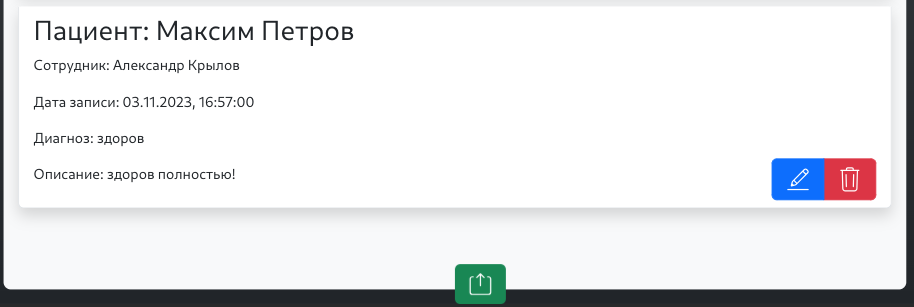


рис. 6.20 – Отредактированный приём

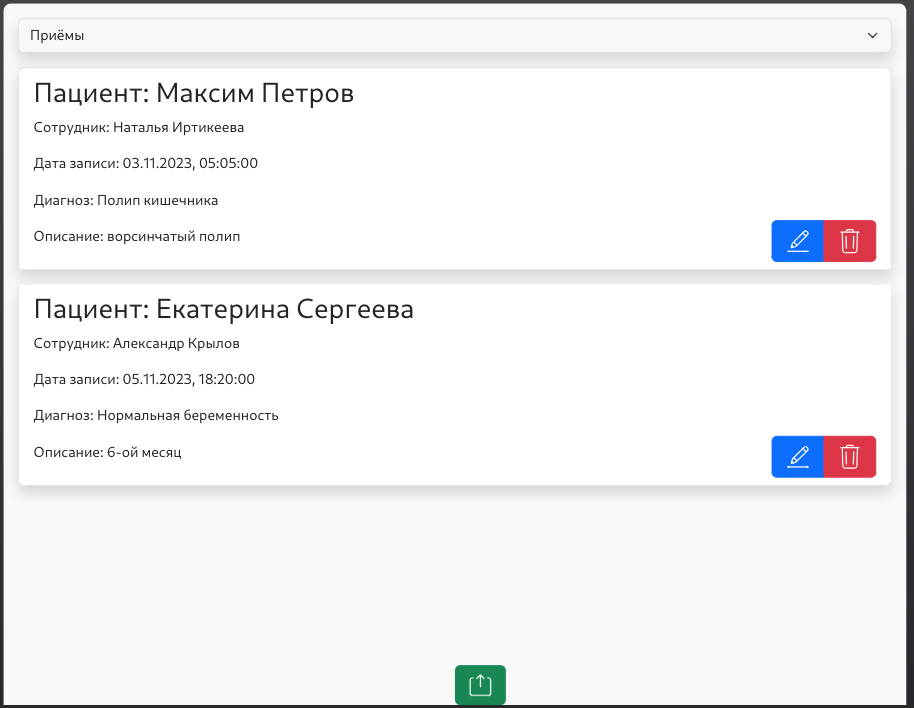


рис. 6.21 – Удаление тестового приёма

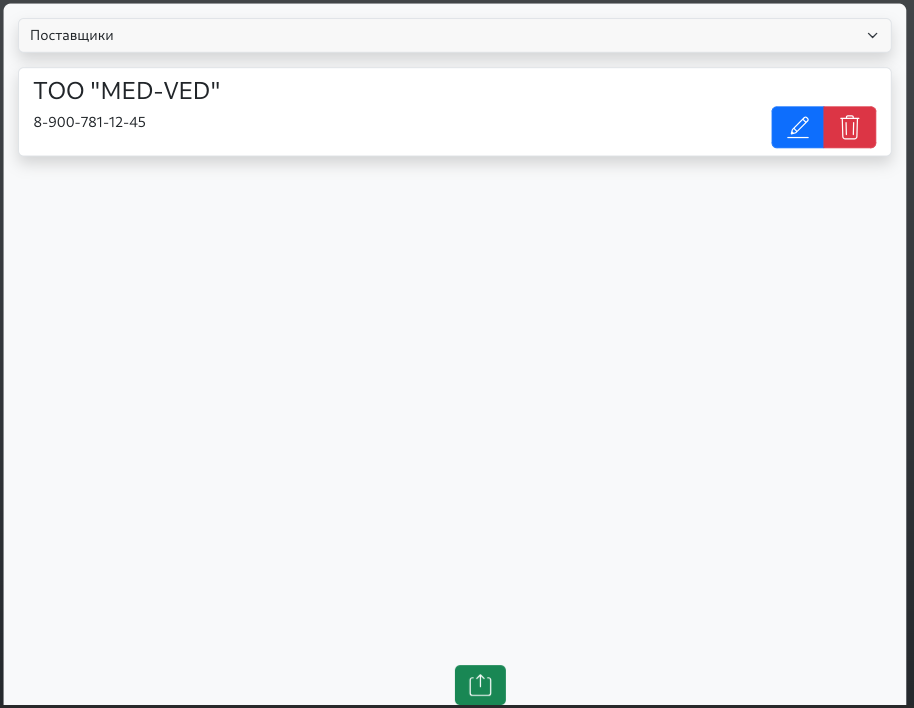


рис. 6.22 – Выбор сущности «Поставщики»

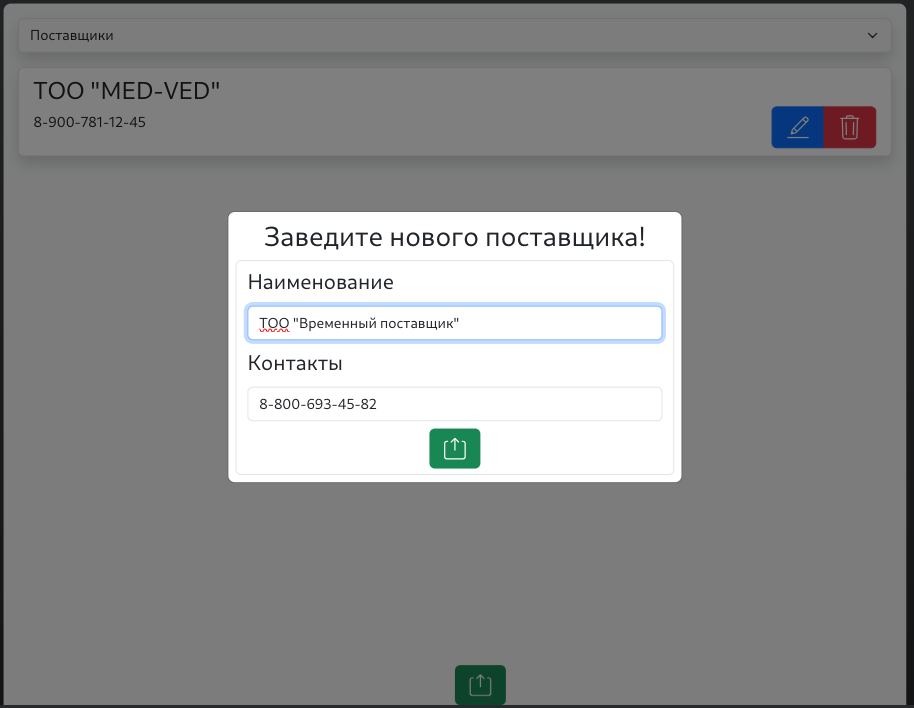


рис. 6.23 – Создание временного поставщика

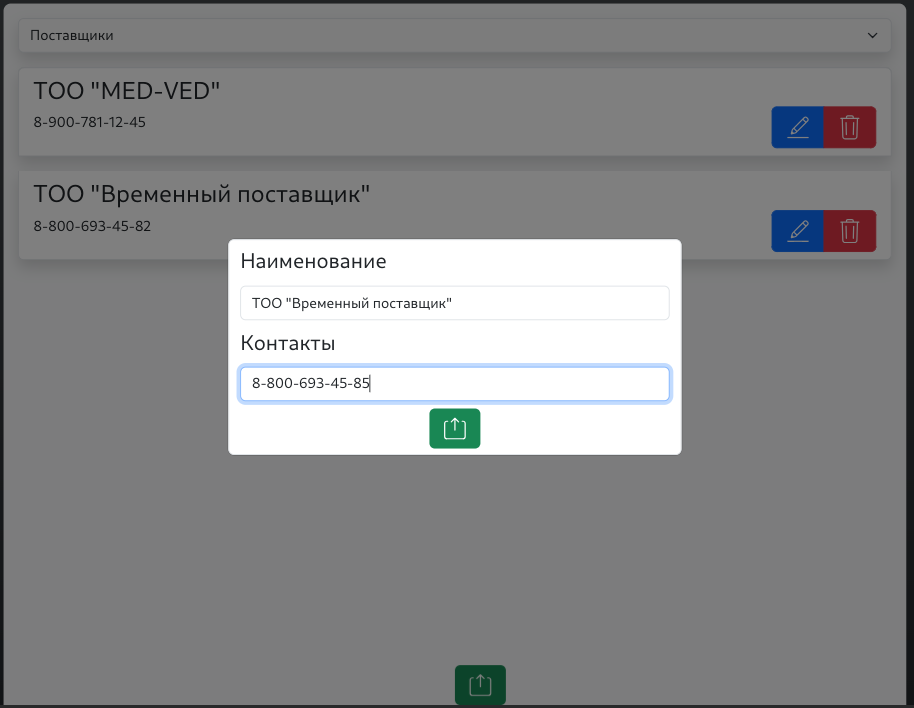


рис. 6.24 – Редактирование временного поставщика

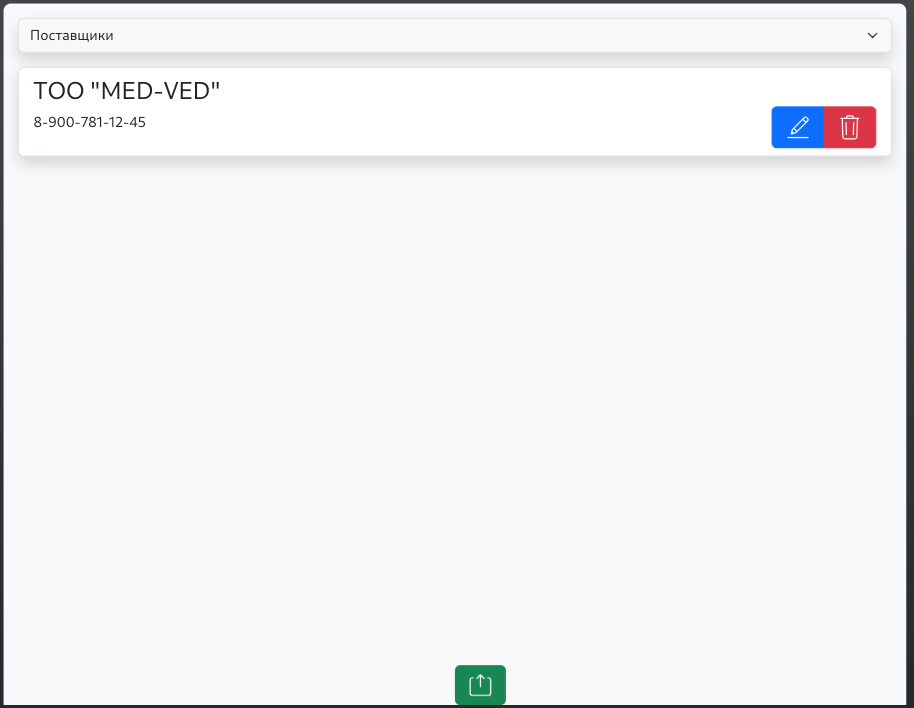


рис. 6.25 – Удаление временного поставщика

# 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}));  
 }  
  
 fetchItems = async () => {  
 return await fetch("http://localhost:8080/equipments")  
 .then(resp => {  
 if (!resp.ok) {  
 throw new Error({message: ***JSON***.stringify(resp.body)});  
 }  
 return resp.json();  
 })  
  
 }  
  
 render() {  
  
 let {items, isLoaded, error} = this.state;  
 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 container-xl d-flex flex-column justify-content-between mt-3 shadow"  
 key={i.\_links.self.href}>  
 <div className="d-flex flex-column ">  
 <div className="d-flex justify-content-between text-break">  
 <h2>Наименование:</h2>  
 <h2>{i.name}</h2>  
 </div>  
 <div className="d-flex justify-content-between">  
 <p>Производитель:</p>  
 <p>{i.manufacturer}</p>  
 </div>  
 <div className="d-flex justify-content-between">  
 <p>Дата приобретения:</p>  
 <p>{new ***Date***(Date.parse(i.acquisitionDate)).toLocaleDateString()}</p>  
 </div>  
 <div className="d-flex justify-content-between">  
 <p>Инвентарный номер:</p>  
 <p>{i.inventoryNumber}</p>  
 </div>  
 <div className="d-flex justify-content-between">  
 <p>Тип:</p>  
 <p>{i.type}</p>  
 </div>  
 </div>  
 <div className="btn-group align-self-end">  
 <button data-bs-target={"#update-equipment-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.equipments],  
 isLoaded: true  
 })  
 })  
 .catch(error => this.setState({error}))  
 }}>  
 <CircumIcon name="trash"/>  
 </button>  
 </div>  
 <div className="modal fade" id={"update-equipment-form" + + i.\_links.self.href.split('/')[i.\_links.self.href.split('/').length-1]}>  
 <div className="modal-dialog-centered modal-dialog">  
 <div className="modal-content">  
 <EquipmentForm  
 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-equipment-form">  
 <FixedSaveBtn/>  
 </div>  
 <div className="modal fade" id="create-equipment-form">  
 <div className="modal-dialog modal-dialog-centered">  
 <div className="modal-content p-2">  
 <h2 className="text-center">Опишите новое оборудование!</h2>  
 <EquipmentForm onSubmit={(data => {  
 fetch("http://localhost:8080/equipments",  
 {  
 method: "POST",  
 headers: {  
 "Content-Type": "application/json"  
 },  
 body: ***JSON***.stringify(data)  
 }  
 ).then(resp => {  
 return this.fetchItems()  
 })  
 .then(json => {  
 this.setState({  
 items: [...json.\_embedded.equipments],  
 isLoaded: true  
 })  
  
 })  
 .catch(error => this.setState({error}))  
 })}/>  
 </div>  
 </div>  
 </div>  
 </>  
 );  
 }  
}

import {PatientCreator} from "../form/patient/PatientCreator";  
import PatientsViewer from "../view/PatientsViewer";  
import 'bootstrap/dist/css/bootstrap.css';  
import React from "react";  
import FixedSaveBtn from "../util/FixedSaveBtn";  
export default class Patient extends React.Component {  
  
 constructor(props) {  
 super(props);  
 this.state = {}  
 }  
  
 render() {  
 return (  
 <div className="min-vh-100 d-flex flex-column align-items-center justify-content-between">  
 <div className="mt-2">  
 <PatientsViewer/>  
 </div>  
 <div data-bs-toggle="modal" data-bs-target="#creator">  
 <FixedSaveBtn/>  
 </div>  
 <div className="modal fade" id="creator">  
 <div className="modal-dialog">  
 <div className="modal-content">  
 <PatientCreator/>  
 </div>  
 </div>  
 </div>  
 </div>  
 );  
 }  
  
  
}

import React from "react";  
import Error from "../util/Error";  
import Loading from "../util/Loading";  
import {FormSaveBtn} from "../form/FormSaveBtn";  
import CircumIcon from "@klarr-agency/circum-icons-react";  
import StaffForm from "../form/staff/StaffForm";  
import FixedSaveBtn from "../util/FixedSaveBtn";  
export default class Staff extends React.Component {  
  
 constructor(props) {  
 super(props);  
 this.state = {  
 isLoaded: false  
 }  
 }  
  
 componentDidMount() {  
 this.fetchItems().then(  
 json => {  
 this.setState({  
 items: [...json.\_embedded.staff],  
 isLoaded: true  
 })  
  
 }  
 )  
 .catch(error => this.setState({error}));  
 }  
  
 fetchItems = async () => {  
 return await fetch("http://localhost:8080/staff")  
 .then(resp => {  
 if (!resp.ok) {  
 throw new Error({message: ***JSON***.stringify(resp.body)});  
 }  
 return resp.json();  
 })  
  
 }  
  
 render() {  
  
 const {items, isLoaded, error} = this.state;  
 if (error) {  
 return <Error message={`${error.message}`}/>  
 } else if (!isLoaded) {  
 return <Loading/>  
 }  
  
 return (  
 <>  
 <ul className="list-group min-vh-100">  
 {  
 items.map(  
 i => (  
 <li className="list-group-item d-flex flex-column justify-content-between mt-3 shadow" key={i.\_links.self.href}>  
 <div className="container">  
 <h2>{i.firstname} {i.lastname}</h2>  
 <p>Должность: {i.position}</p>  
 <p>Контакты: {i.contacts}</p>  
 </div>  
 <div className="btn-group align-self-end">  
 <button data-bs-target={"#update-staff-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.staff],  
 isLoaded: true  
 })  
 })  
 .catch(error=>this.setState({error}))  
 }}>  
 <CircumIcon name="trash"/>  
 </button>  
 </div>  
 <div className="modal fade" id={"update-staff-form"+ i.\_links.self.href.split('/')[i.\_links.self.href.split('/').length-1]}>  
 <div className="modal-dialog-centered modal-dialog">  
 <div className="modal-content">  
 <StaffForm initData={i} onSubmit={data=>{  
 fetch(i.\_links.self.href,{  
 method:"PUT",  
 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-stuff-form">  
 <FixedSaveBtn/>  
 </div>  
 <div className="modal fade" id="create-stuff-form">  
 <div className="modal-dialog modal-dialog-centered">  
 <div className="modal-content p-2">  
 <h2>Создайте сотрудника больницы</h2>  
 <StaffForm onSubmit={(data => {  
 fetch("http://localhost:8080/staff",  
 {  
 method: "POST",  
 headers: {  
 "Content-Type": "application/json"  
 },  
 body: ***JSON***.stringify(data)  
 }  
 ).then(resp => {  
 return this.fetchItems()  
 })  
 .then(json => {  
 this.setState({  
 items: [...json.\_embedded.staff],  
 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 SupplierForm from "../form/SupplierForm";  
export default class Supplier extends React.Component{  
  
 constructor(props) {  
 super(props);  
 this.state = {  
 isLoaded: false  
 }  
 }  
  
 componentDidMount() {  
 this.fetchItems().then(json => {  
 this.setState({  
 items: [  
 ...json.\_embedded.suppliers  
 ]  
 }, () => {  
 ***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/suppliers")  
 .then(resp => {  
 if (!resp.ok) {  
 throw new Error({message: ***JSON***.stringify(resp.body)});  
 }  
 return resp.json();  
 })  
  
 }  
  
  
  
 render() {  
  
 let {items, isLoaded, error} = this.state;  
 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="">  
 <h3>{i.name}</h3>  
 <p>{i.contacts}</p>  
 </div>  
 <div className="btn-group align-self-end">  
 <button data-bs-target={"#update-supplier-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.suppliers],  
 isLoaded: true  
 })  
 })  
 .catch(error => this.setState({error}))  
 }}>  
 <CircumIcon name="trash"/>  
 </button>  
 </div>  
 <div className="modal fade" id={"update-supplier-form" + i.\_links.self.href.split('/')[i.\_links.self.href.split('/').length-1] }>  
 <div className="modal-dialog-centered modal-dialog">  
 <div className="modal-content">  
 <SupplierForm  
 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-supplier-form">  
 <FixedSaveBtn/>  
 </div>  
 <div className="modal fade" id="create-supplier-form">  
 <div className="modal-dialog modal-dialog-centered">  
 <div className="modal-content p-2">  
 <h2 className="text-center">Заведите нового поставщика!</h2>  
 <SupplierForm onSubmit={(data => {  
 fetch("http://localhost:8080/suppliers",  
 {  
 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>  
 </>  
 );  
 }  
}