Министерство науки и высшего образования Российской федерации
ФГБОУ ВО «Рыбинский государственный авиационный
технический университет имени П. А. Соловьева»

Институт информационных технологий и систем управления

Кафедра математического и программного обеспечения
электронных вычислительных средств

Направление подготовки
09.04.01 Информатика и вычислительная техника

КУРСОВАЯ РАБОТА

по дисциплине

СОВРЕМЕННЫЕ ТЕХНОЛОГИИ ПРОМЫШЛЕННОЙ РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

на тему:

ПРОЕКТИРОВАНИЕ СИСТЕМЫ СТРУКТУРИЗАЦИИ И ХРАНЕНИЯ ДАННЫХ ПЕРСОНАЛА ПРЕДПРИЯТИЯ

Пояснительная записка

Студент группы ИВМ-22 Щербаков М.И.

“___” _____ 2022 г.

Руководитель, доцент: Волков М. Л.

Петров Н. С

“___” _____ 2022 г.

Рыбинск 2022

Содержание

Введение

На всех современных предприятиях ведется учет сотрудников. При этому получаемый в результате массив данных требует жесткой структуризации. Бумажный документооборот постепенно заменяется электронным в виду эффективности, надежности и быстродействия последнего.

Но вместе с этим появляется потребность организации безопасного хранения оцифрованных документов. Для этих целей разрабатываются электронные хранилища – базы данных. Все материалы в базе данных взаимодействуют определенным образом: за изменением одной строчки следуют изменения других данных. Это упрощает работу с большим объемом информации [1].

Целью выполнения курсового проекта является знакомство с системами хранения и обработки баз данных и создание приложения, демонстрирующего возможности электронной системы структуризации.

В виду особенностей выбранного программного обеспечения, разрабатываемая система может дополнятся и видоизменяться, усложняясь и адаптируюсь под нужды конкретного предприятия.

1 Анализ технического задания

1.1 Выбор и обоснование программного обеспечения

В качестве основы серверной части в рамках проходимой дисциплины был выбран объектно-ориентированный язык программирования *Java* [2].

Приложения *Java* обычно транслируются в специальный байт-код, поэтому они могут работать на любой компьютерной архитектуре, для которой существует реализация виртуальной *Java*-машины – программой, обрабатывающей байтовый код и передающей инструкции оборудованию как интерпретатор. Другой важной особенностью технологии *Java* является гибкая система безопасности, в рамках которой исполнение программы полностью контролируется виртуальной машиной. Любые операции, которые превышают установленные полномочия программы (например, попытка несанкционированного доступа к данным или соединения с другим компьютером), вызывают немедленное прерывание.

Данный язык достаточно широко используется в сфере проектирования серверных приложений и веб-приложений.

В качестве системы управления базой данных (СУБД) выбрана *PostgreSQL* [3], поддерживающая БД неограниченного размера, надежных механизмов транзакций, поддержки наследования и легкой расширяемости, что позволит в дальнейшем расширить разрабатываемую систему или использовать ее в качестве интегрируемой подсистемы. Доступ к таблицам базам данных *PostgreSQL* можно получить с помощью утилиты *pgAdmin*, предоставляющей мощный графический интерфейс для создания, обслуживания и использование объектов базы данных. Для этого необходимо создать *docker*-контейнер [3] с запущенной службой *pgAdmin* и выполнить подключение. Интерфейс программы представлен на рисунке 1.1. Порт для подключения БД установлен по умолчанию.

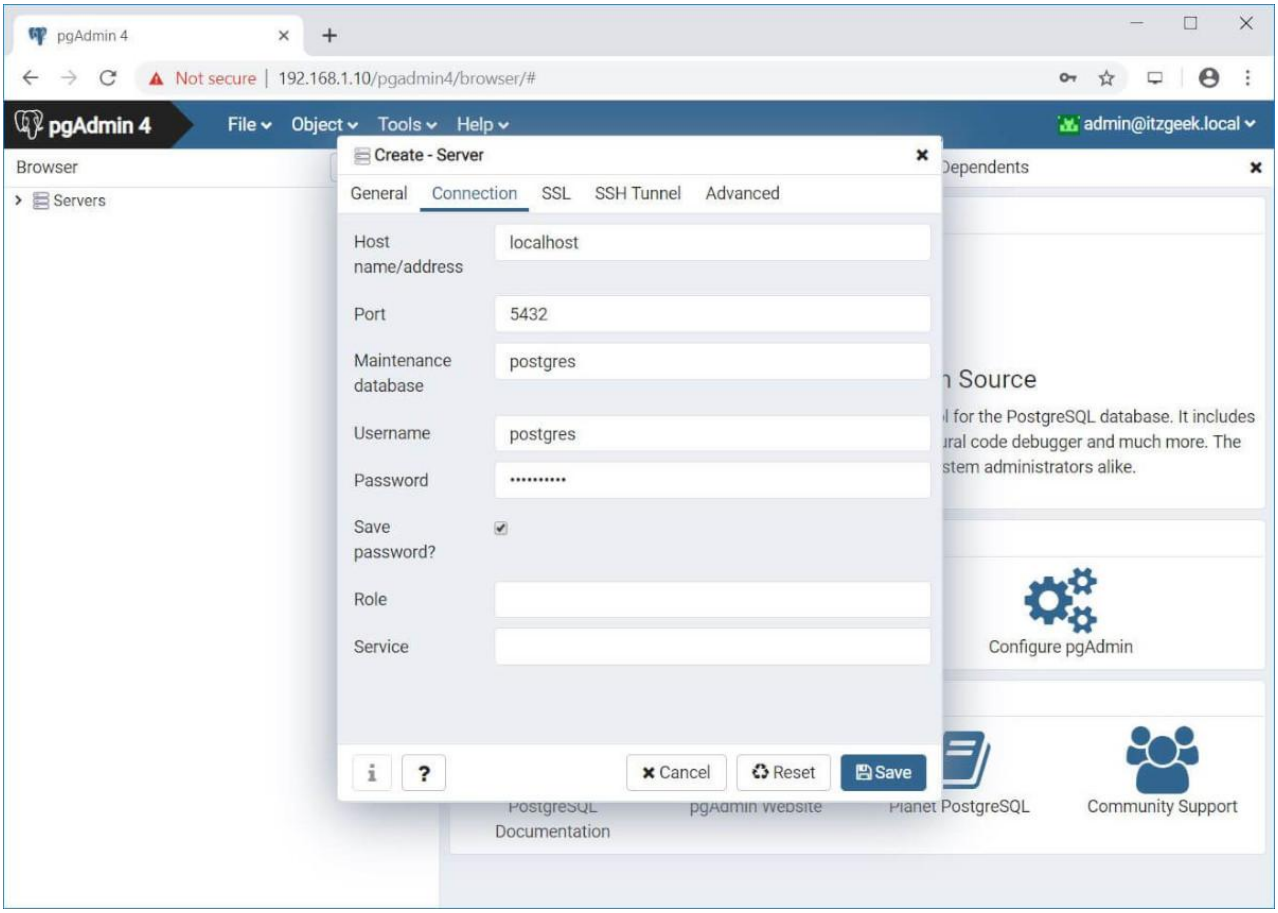

Рисунок 1.1 – Подключение БД с помощью утилиты *pgAdmin*

Разработка серверной части проводилась в среде *IntelliJ IDEA* с использованием фреймворков *Maven* [5] и *Quarkus* [6].

Maven – инструмент для автоматизации сборки проектов. Одна из главных особенностей фреймворка – декларативное описание проекта. Ещё одно достоинство проекта – гибкое управление зависимостями. *Maven* умеет подгружать в свой локальный репозиторий сторонние библиотеки, выбирать необходимую версию пакета, обрабатывать транзитивные зависимости. *Maven* выполняет сборку последовательными фазами, иначе называемыми жизненным циклом проекта:

1) Проверка — *validate*. Фреймворк проверяет, корректен ли проект и предоставлена ли вся необходимая для сборки информация;
2) Компиляция — *compile*. Maven компилирует исходники проекта;
3) Тест — *test*. Проверка скомпилированных файлов;

4) Сборка проекта — *package*. По умолчанию осуществляется в формате *JAR*. Этот параметр можно изменить, добавив в *project* тег *packaging*;
5) Интеграционное тестирование — *integration-test*. *Maven* обрабатывает и при необходимости распаковывает пакет в среду, где будут выполняться интеграционные тесты;
6) Верификация — *verify*. Артефакт проверяется на соответствие критериям качества;
7) Инсталляция — *install*. Артефакт попадает в локальный репозиторий;
8) Размещение проекта в удалённом репозитории — *deploy*, — финальная стадия работы.

Quarkus – это *Kubernetes*-нативный *Java*-фреймворк, созданный для виртуальных *Java*-машин (*JVM*), поэтому он предоставляет средства разработки компонентов *Java*-приложений, реализующих преимущества *Kubernetes*, системы оркестрации контейнеров с открытым кодом.

Quarkus работает как на серверной, так и на клиентской стороне (на уровне пользовательских приложений). Он обеспечивает быстрое время загрузки приложения и низкое потребление памяти. На рисунке 1.2 демонстрируется выбор зависимостей проекта. Результатом является сгенерированный шаблон приложения.

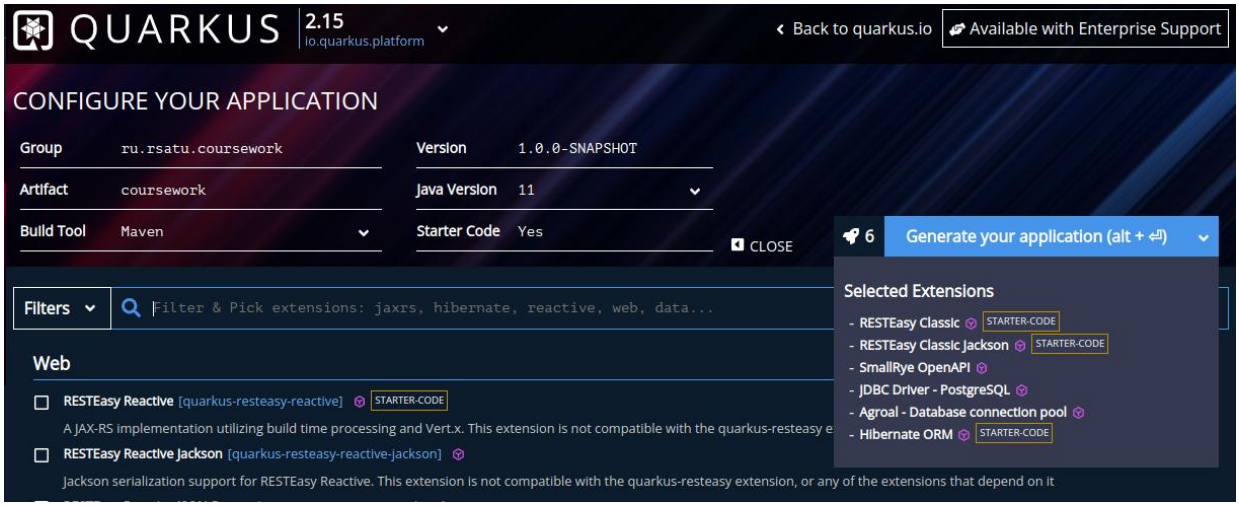

Рисунок 1.2 – Выбор зависимостей проекта на *Quarkus*

Клиентская часть выполнена в виде веб-страницы на *HTML* с использованием *JavaScript* и *Vue.js*.

JavaScript – встраиваемый язык для программного доступа к объектам приложений. Наиболее широкое применение находит в браузерах как язык сценариев для придания интерактивности веб-страницам.

Vue.js [7] — это фреймворк для создания пользовательских интерфейсов. В отличие от фреймворков-монолитов, *Vue* создан пригодным для постепенного внедрения. Его ядро в первую очередь решает задачи уровня представления (*view*), что упрощает интеграцию с другими библиотеками и существующими проектами. С другой стороны, *Vue* полностью подходит и для создания сложных одностраничных приложений (*SPA, Single-Page Applications*), если использовать его совместно с современными инструментами и дополнительными библиотеками.

В любом серьезном проекте подобного типа требуется реализации контроля доступа, тем самым обеспечив защиту данных предприятия. Организация контроля доступа выполнена с помощью *Keycloak* [8]. Это продукт с открытым кодом для реализации *single sign-on* с возможностью управления доступом, нацелен на современные приложения и сервисы. Целью этого инструмента является сделать создание безопасных приложений и сервисов с минимальным написанием кода для аутентификации и авторизации. *Keycloak* состоит из сервера и адаптера к приложению. Для сервера, как остальным компонентам проекта, выделяется собственный *docker*-контейнер. На рисунке 1.3 представлен веб-интерфейс программы.

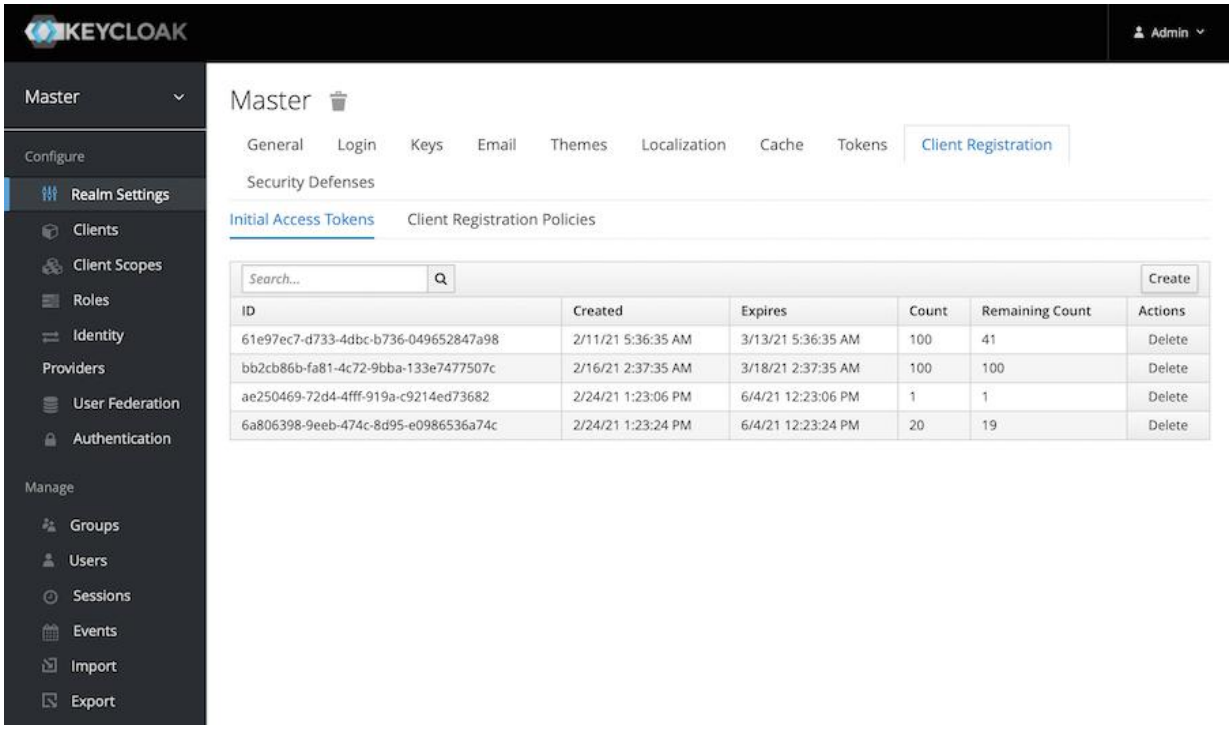

Рисунок 1.3 – Интерфейс *Keycloak*. Вкладка настройки проекта

1.2 Описание Java-сущностей

Сущности в *Java* – это простой объект, не обремененный, какими либо логическими обязательствами, хранящий и представляющий свои данные для других объектов. Не все объекты в *Java* могут быть сущностями, их отличительная черта в том, что их можно сериализовать (сериализация - процесс записи состояния объекта в поток; такой подход очень удобен, когда идет работа со сложными объектами) и десериализовать (десериализация соответственно – процесс извлечения или восстановления состояния объекта из потока). При использовании модели *Object Relation Mapping* (хранение объектов в табличной БД) сущности (*Entity*) — сохраняемые таким образом объекты.

Описание сущностей разрабатываемого проекта представлено в таблице 3.1. В качестве основной сущности выбран объект *Users* –пользователи системы или сотрудники предприятия.

Таблица 3.1 – Описание сущностей проекта

Сущность	Назначение
Users	Список сотрудников и их данные: ИД, имя, внутренний телефон и номер сектора, где работает сотрудник

В таблице 3.2 представлено описание полей сущностей *Users*. Поля могут ссылаться на другие сущности.

Таблица 3.2 – Описание полей сущности *Users*

Имя поля	Тип	Назначение
id	*Long*	Идентификационный номер пользователя
name	*String*	ФИО сотрудника
phone	*Long*	Дополнительная информация (внутренний телефон)
sector_id	*Long*	Номер сектора (отдела), к которому привязан сотрудник

2 Back-end разработка приложения

Задача *back-end* разработки сводится к проектированию серверной части. Для этого был реализован ряд классов, осуществляющих доступ к БД. Модели данных описываются в пакете *pojo* («*plain-old-Java-object*», т.е. простые классы), классы для работы с БД находятся в пакете *service*. В качестве примера в Приложении А представлена реализация класса

Описание таблиц производится с помощью классов *Entity*. Взаимодействие с клиентской частью приложения осуществляют классы *Dto*. В Приложении Б приводится пример *Entity*–класса.

Бэкенд не может отдавать данные из репозитория как есть, поэтому *Entity* необходимо преобразовать в *Dto*, для чего разрабатываются *mapper*-классы. Пример такого *mapper*-класса представлен в Приложении В.

Получение и обработку данных из БД берут на себя класс *repository*, листинг которого приведен в Приложении Г. Класс, реализующий интерфейсы и взаимодействующие с клиентской частью, помещен в пакет *resource*. Листинг данного класса приведен в Приложении Д.

Дерево проекта представлено на рисунке 2.1.

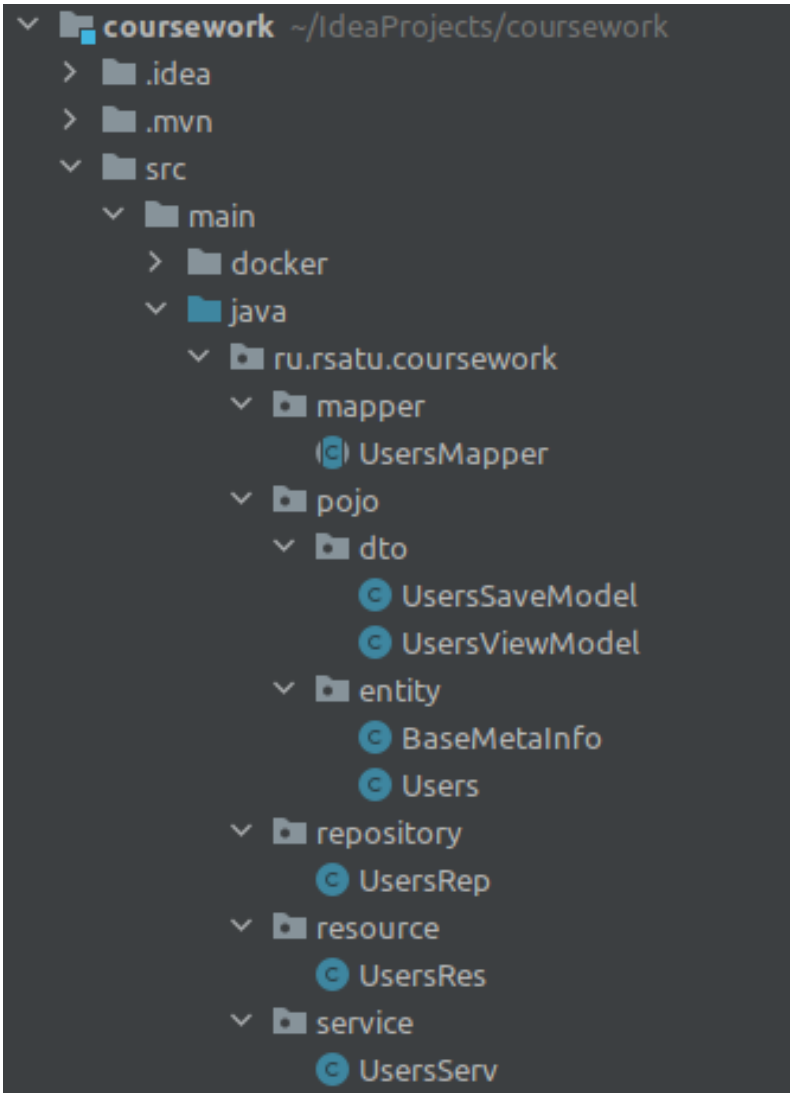

Рисунок 2.1 – Каталог проекта

Параметры соединения с базой данных и параметры подключения к *Keycloak* описываются в *application.properties* – рисунок 2.2.

Рисунок 2.2 – Параметры подключения к БД

Сборка проекта выполняется с помощью фреймворка *Maven*. Использования автоматизированного сборщика ускоряет и упрощает *back-end*, поскольку в таком случае весь проект выстраивается на моделях зависимостей (*dependencies*), описанных в *pom*-файле.

Необходимые для работы проекта зависимости были внедрены ранее с помощью *Quarkus*.

3 Front-end разработка приложения

Front-end разработка сводится к проектированию клиентского приложения, в данном случае реализованного в виде веб-страницы средствами *HTML*, *JavaScript* и *Vue*.

После авторизации пользователь получает доступ к чтению содержимого БД (и записи, если пользователь авторизован как администратор).

На вкладке сотрудники расположена таблица с выводом некоторого количества строк. Навигация по остальной части таблицы осуществляется путем выбора страницы.

3.1 Организация контроля доступа

На рисунке 3.1 представлены настройки *realm*, примененные к проекту.

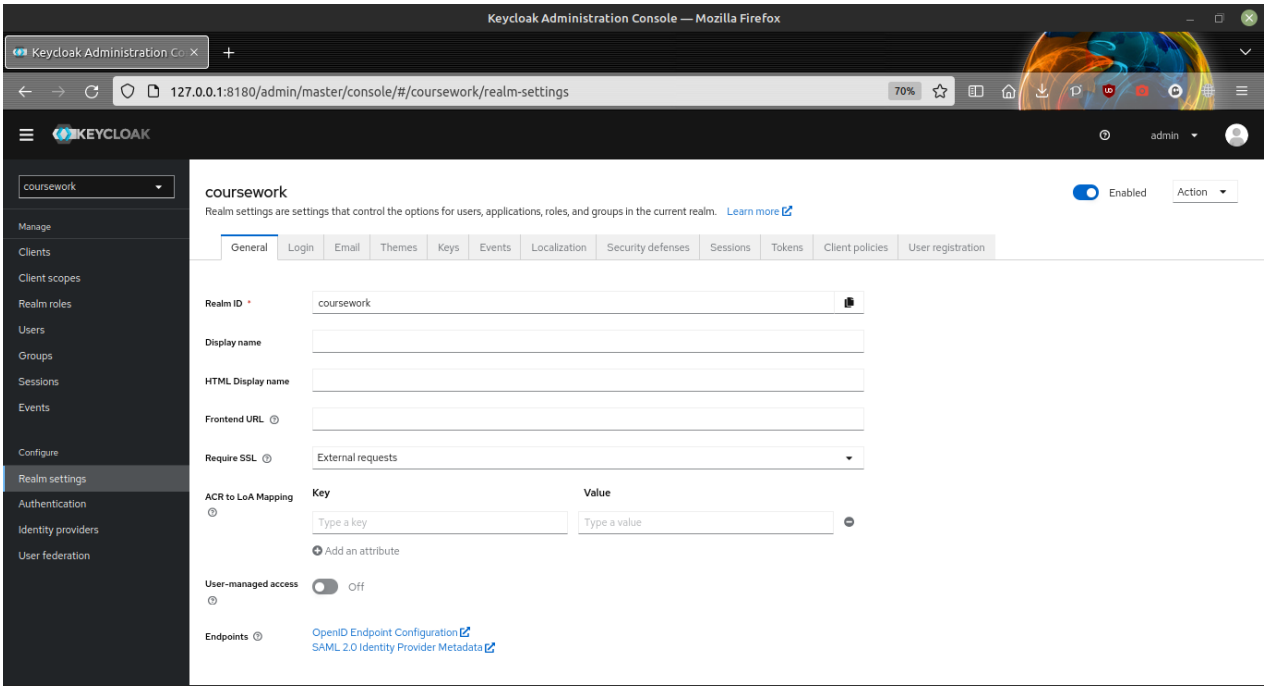

Рисунок 3.1 – Настройка *realm*

На рисунках 3.2 – 3.4 представлены список ролей пользователей и доступные им действия (чтение и запись данных).

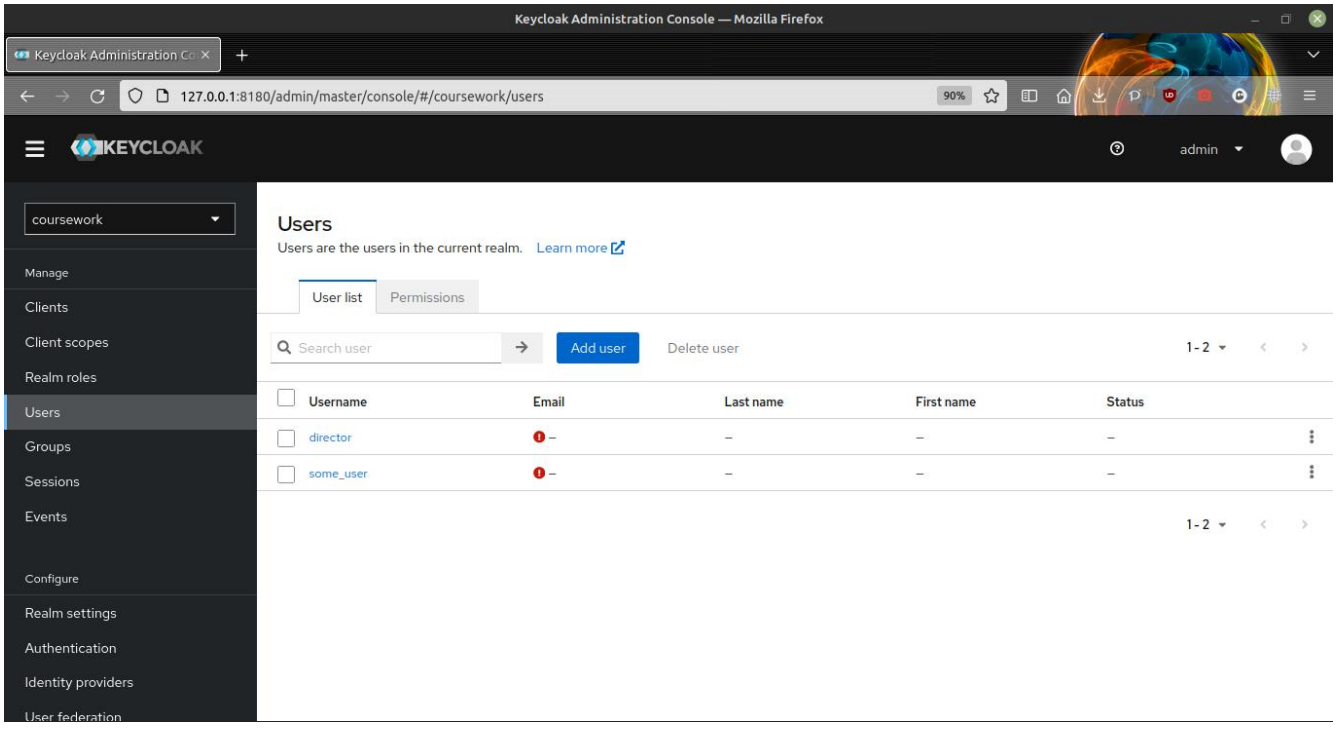

Рисунок 3.2 – Роли пользователей

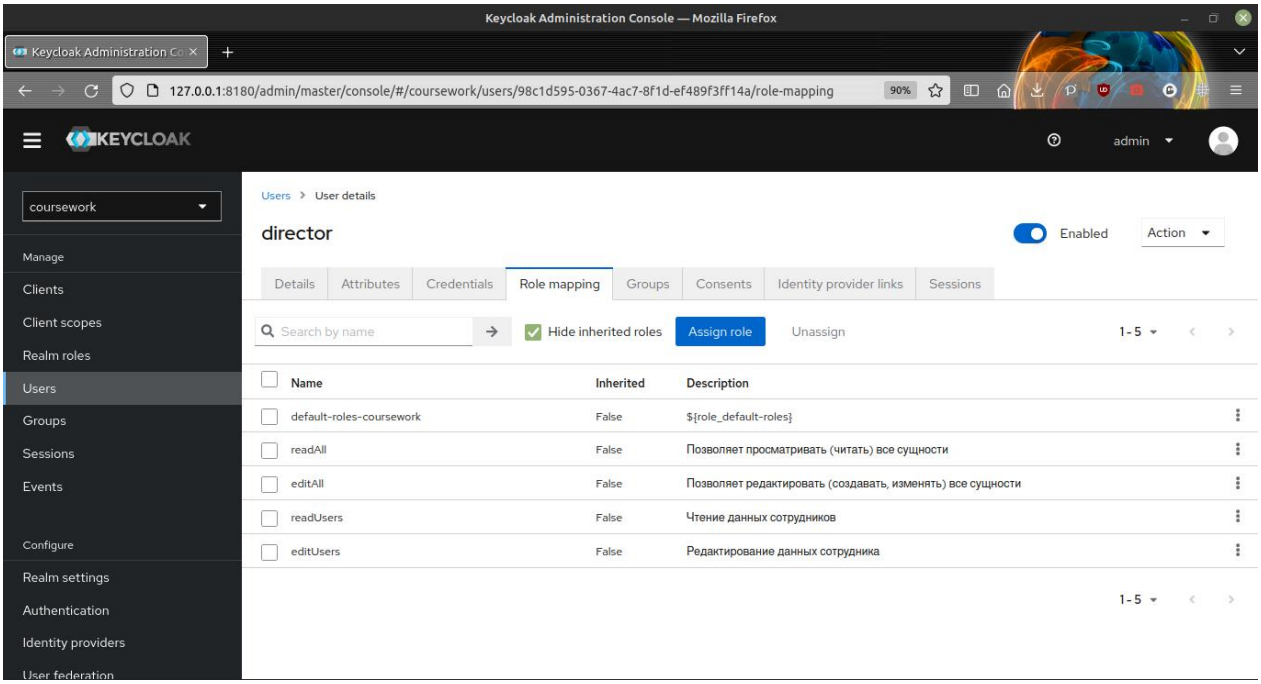

Рисунок 3.3 – Действия, доступные пользователю «*director*» (чтение и запись)

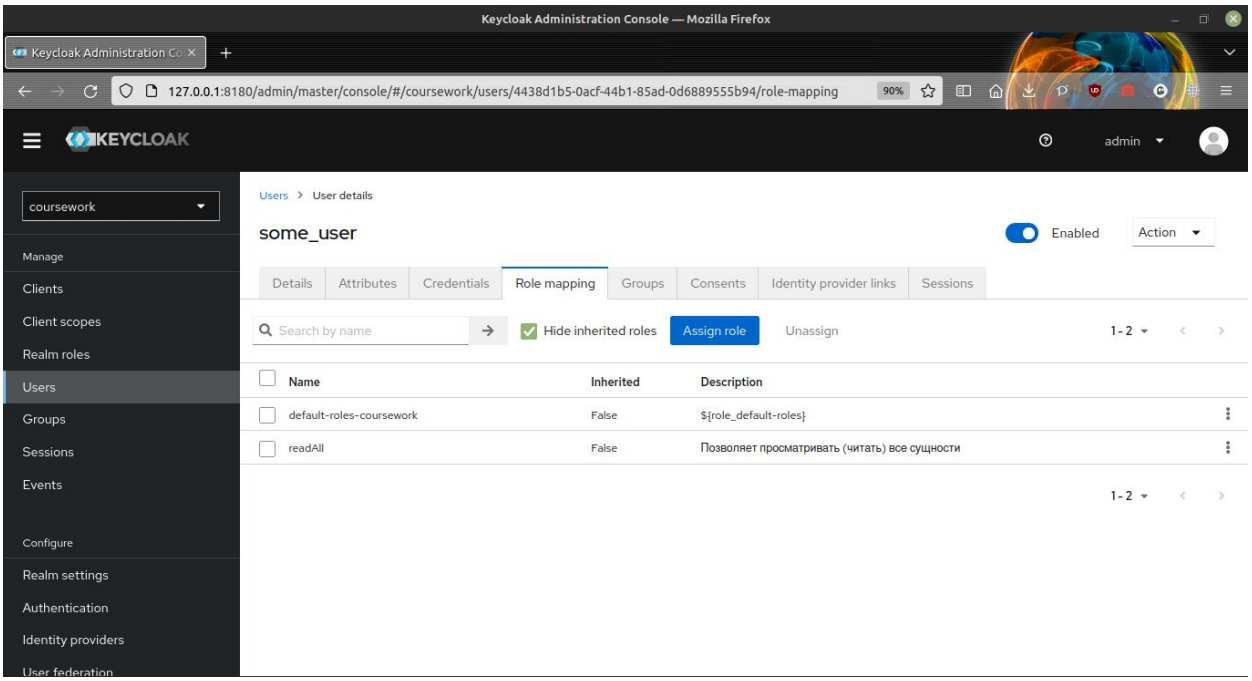

Рисунок 3.4 – Действия, доступные пользователю «*some_user*» (чтение)

Таким образом, с помощью настроек ролей *Keycloak* можно ограничить доступ пользователей к определенным сущностям проекта.

4 Демонстрация работы

Для работы проекта необходимо запустить докер-контейнер с базой данных на порту, прописанном в параметрах подключения сервера, контейнер *keycloak*, сервер приложения и клиентскую часть приложения (рисунок 4.1).

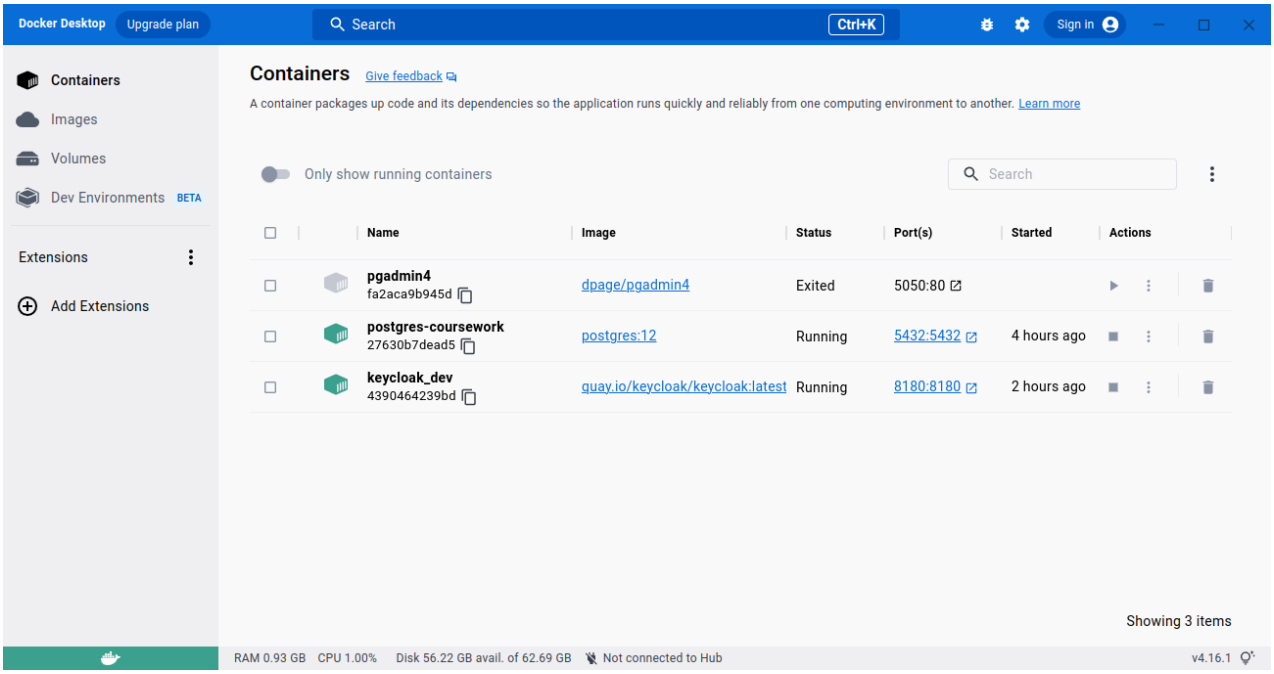

Рисунок 4.1 – запуск *docker*-контейнеров

На рисунке 4.2 представлена вкладка авторизации *Keycloack*. На рисунке 4.3 представлена вкладка Список сотрудников.

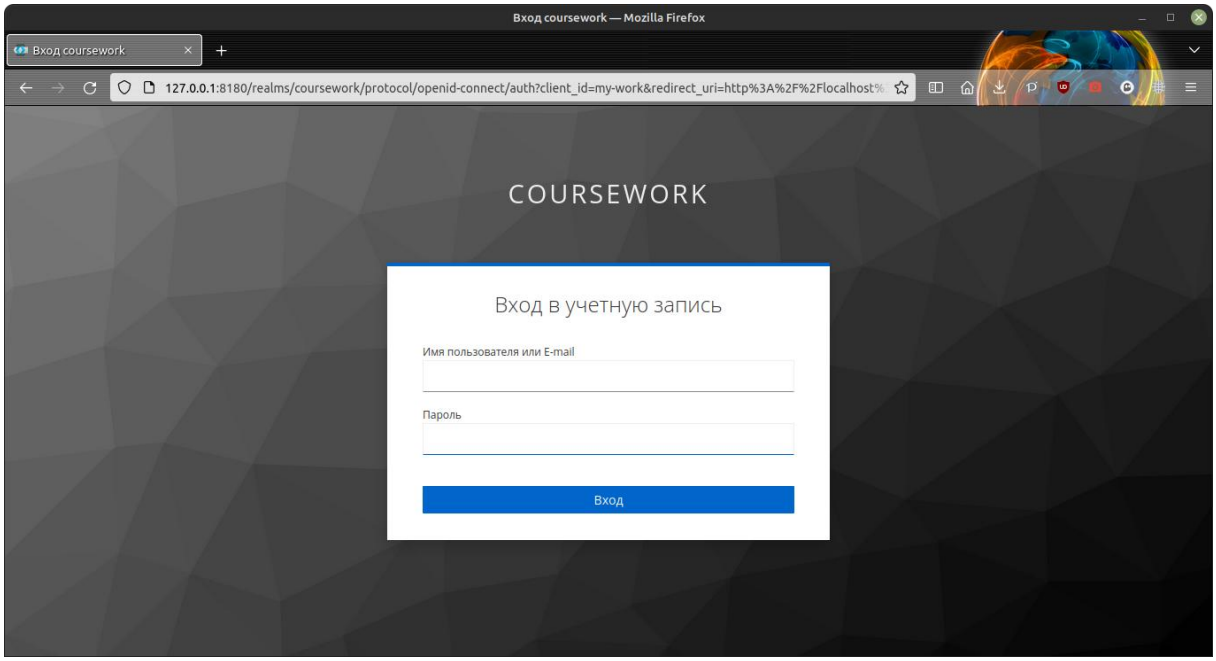

Рисунок 4.2 - Вкладка авторизации *Keycloack*

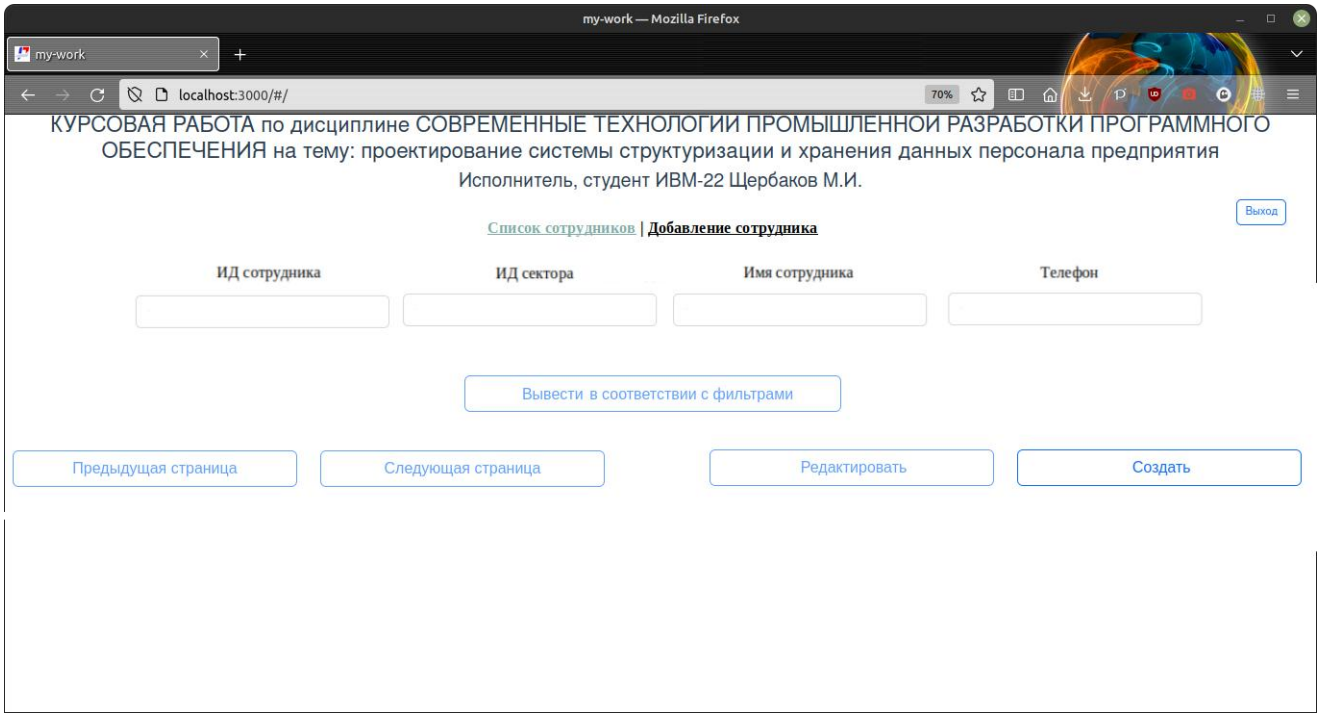

Рисунок 4.3 - Вкладка «Список сотрудников» с возможностью провести фильтрацию, вывести список согласно установленному фильтру, редактировать и создать записи

Заключение

В процессе выполнения курсовой работы были получены навыки в организации серверного приложения, проведения *front-end* и *back-end* разработки.

Были изучены методы работы с базами данных и контейнерами. Изучены методы построения системы авторизации и контроля доступа. Получены навыки в разработки веб-приложений.

В дальнейшем разработанное приложение может дополняться в зависимости от потенциальной сферы применения. Также данное приложение может стать вызываемым модулем другой системы структуризации данных.

Список использованных источников

1. ООО «ГикБреинс» Что такое база данных: принципы работы, лучшие СУБД [электронный ресурс]. – URL: https://gb.ru/blog/chto-takoe-baza-dannykh/ (дата обращения 22.12.2022)

2. Oracle Java [электронный ресурс]. – URL: https://www.java.com/ru/ (дата обращения 10.12.2022)

3. The PostgreSQL Global Development Group [электронный ресурс]. – URL: https://www.postgresql.org/ (дата обращения 12.12.2022)

4. Docker Inc. [электронный ресурс]. – URL: https://www.docker.com/ (дата обращения 10.12.2022)

5. Apache Software Foundation. Apache Maven [электронный ресурс]. – URL: https://maven.apache.org/ (дата обращения 22.12.2022)

6. Quarkus - Supersonic Subatomic Java [электронный ресурс]. – URL: https://quarkus.io/ (дата обращения 22.12.2022)

7. Vue.js [электронный ресурс]. – URL: https://ru.vuejs.org/index.html (дата обращения 19.12.2022)

8. Open Source Identity and Access Management [электронный ресурс]. – URL: https://www.keycloak.org/ (дата обращения 22.12.2022)

Приложение А

Листинг service-класса *UsersServ*

```
package ru.rsatu.coursework.service;
import ru.rsatu.coursework.mapper.UsersMapper;
import ru.rsatu.coursework.pojo.dto.UsersSaveModel;
import ru.rsatu.coursework.pojo.dto.UsersViewModel;
import ru.rsatu.coursework.repository.UsersRep;
import javax.enterprise.context.ApplicationScoped;
import javax.inject.Inject;
import java.util.List;
import java.util.stream.Collectors;
@ApplicationScoped
public class UsersServ {
    @Inject
    UsersMapper mapper;
    @Inject
    UsersRep rep;
    //Получение списка сотрудников
    public List<UsersViewModel> load() {
        return rep.load().stream()
                .map(mapper::toUsersViewModel)
                .collect(Collectors.toList());
    }
    //Сохранить данные сотрудника
    public void save(UsersSaveModel model) {
        rep.save(mapper.toUsers(model));
    }
}
```

Приложение Б

Листинг *Entity*–класса *User*s

```
package ru.rsatu.coursework.pojo.entity;
import lombok.Getter;
import lombok.Setter;
import org.hibernate.annotations.Comment;
import javax.persistence.*;
//Таблица "Сотрудники"
@Getter
@Setter
@Entity
@Table(name = "Users")
public class Users extends BaseMetaInfo {
    // ИД сотрудника
    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE,
            generator = "users_id_gen")
    @SequenceGenerator(name = "users_id_gen",
            sequenceName = "users_id_gen_seq",
            initialValue = 10,
            allocationSize = 10)
    @Column(name = "id")
    @Comment(value = "Идентификатор сотрудника")
    private Long id;
    //Сектор сотрудника
    @JoinColumn(name = "sector_id")
    @Comment(value = "Номер сектора")
    private Long sector_id;
```

```
    // Имя сотрудника
    @Column(name = "name")
    @Comment(value = "ФИО сотрудника")
    private String name;
    //Дополнительная информация о сотруднике
    @Column(name = "phone")
    @Comment(value = "Внутренний телефон сотрудника")
    private Long phone;
}
```

Приложение В

Листинг *mapper*-класса *UsersMapper*

```
package ru.rsatu.coursework.mapper;
import io.quarkus.security.identity.SecurityIdentity;
import org.mapstruct.AfterMapping;
import org.mapstruct.Mapper;
import org.mapstruct.Mapping;
import org.mapstruct.MappingTarget;
import ru.rsatu.coursework.pojo.dto.*;
import ru.rsatu.coursework.pojo.entity.*;
import javax.inject.Inject;
import java.sql.Timestamp;
@Mapper(componentModel = "cdi")
public abstract class UsersMapper {
    @Inject
    SecurityIdentity securityIdentity;
    @Mapping(target = "id", source = "id")
    @Mapping(target = "sector_id", source = "sector_id")
    @Mapping(target = "name", source = "name")
    @Mapping(target = "phone", source = "phone")
    public abstract UsersViewModel toUsersViewModel(Users from);
    @Mapping(target = "id", source = "id")
    @Mapping(target = "sector_id", source = "sector_id")
    @Mapping(target = "name", source = "name")
    @Mapping(target = "phone", source = "phone")
    public abstract Users toUsers(UsersSaveModel from);
    @AfterMapping
    protected void updateUsersAfterMapping(@MappingTarget Users db_model) {
```

```
db_model.setRecordChangeAuthor(securityIdentity.getPrincipal().getName());
            db_model.setRecordChangeTS(new
Timestamp(System.currentTimeMillis()));
        }
    }
```

Приложение Г

Листинг *repository*-класса *UsersRep*

```
package ru.rsatu.coursework.repository;
import ru.rsatu.coursework.pojo.entity.Users;
import javax.enterprise.context.ApplicationScoped;
import javax.inject.Inject;
import javax.persistence.EntityManager;
import javax.transaction.Transactional;
import java.util.List;
@ApplicationScoped
public class UsersRep {
    @Inject
    EntityManager entityManager;
    //Получить список сотрудников
    public List<Users> load() {
        return entityManager.createQuery("from Users tbl order by tbl.number ASC",
Users.class)
                .getResultList();
    }
    //Сохранение данных сотрудника в базу данных
    @Transactional
    public void save(Users db_model) {
        if (db_model.getId() != null) {
            entityManager.merge(db_model);
        } else {
            entityManager.persist(db_model);
        }
        entityManager.flush();    }}
```

Приложение Д

Листинг *resource*-класса *UsersRes*

```
package ru.rsatu.coursework.resource;
import ru.rsatu.coursework.pojo.dto.UsersSaveModel;
import ru.rsatu.coursework.pojo.dto.UsersViewModel;
import ru.rsatu.coursework.service.UsersServ;
import javax.annotation.security.RolesAllowed;
import javax.inject.Inject;
import javax.ws.rs.*;
import javax.ws.rs.core.MediaType;
import java.util.List;
@Path("/coursework/api/v1/Users")
public class UsersRes {
    @Inject
    UsersServ serv;
    //Получение списка view-моделей
    @GET
    @Path("/loadUsers")
    @RolesAllowed("readUsers")
    public List<UsersViewModel> load() {
        return serv.load();
    }
    //Добавить сотрудника
    @POST
    @Consumes(MediaType.APPLICATION_JSON)
    @Path("/createUsers")
    @RolesAllowed("editUsers")
    public void create(UsersSaveModel model) {
        serv.save(model);
```

```
    }
    //Обновление данных сотрудника
    @PUT
    @Consumes(MediaType.APPLICATION_JSON)
    @Path("/updateUsers")
    @RolesAllowed("editUsers")
    public void update(UsersSaveModel model) {
        serv.save(model);
    }
}
```