МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

НИЖЕГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ

УНИВЕРСИТЕТ им. Р.Е.АЛЕКСЕЕВА

Институт радиоэлектроники и информационных технологий

Кафедра информатики и систем управления

Клиент-серверное приложение «отдел кадров»

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовой работе

по дисциплине

Программирование кроссплатформенных приложений

РУКОВОДИТЕЛЬ:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Ляхманов Д.А.

(подпись) (фамилия, и.,о.)

СТУДЕНТ:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Савкин А.Е.

(подпись) (фамилия, и.,о.)

17-АС

(шифр группы)

Работа защищена «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_

С оценкой \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Оглавление

[Введение: 3](#_Toc105959678)

[**Задание к выполнению курсовой работы:** 3](#_Toc105959679)

[Теоретическая часть: 3](#_Toc105959680)

[**Программный стэк:** 7](#_Toc105959681)

[Практическая часть: 8](#_Toc105959682)

[**Описание функциональности:** 8](#_Toc105959683)

[Приложение: 18](#_Toc105959684)

[**Пример класса сущности (Worker).** 18](#_Toc105959685)

[**Пример интерфейса репозитория (Worker).** 18](#_Toc105959686)

[**Пример контроллера (Worker).** 19](#_Toc105959687)

[**Пример сервиса.** 20](#_Toc105959688)

# Введение:

## **Задание к выполнению курсовой работы:**

Написать клиент-серверное приложение с использованием Spring MVC по технологии REST.

Требования:

1. На серверной стороне необходимо использовать БД PostgreSQL.
2. Доступ к данным осуществляется через Spring Data (JPA)
3. На клиентской стороне используется командный интерфейс (графический +1 балл)

Клиент-серверное приложение «Отдел кадров»:

Клиентское приложение:

* Запрос данных о полном списке сотрудников с фильтрацией по id отдела
* Отделы могут иметь неограниченное количество подчиненных отделов с неограниченной вложенностью
* Запрос списка сотрудников по id отдела
* Переназначение сотрудника из одного отдела в другой
* Командный интерфейс для удаления, добавления и изменения данных о сотрудниках
* Командный интерфейс для удаления, добавления и изменения данных об отделах
* Отображение дерева подчиненных отделов по id отдела

Сервисы серверного приложения:

* Изменение, удаление, добавление записей о сотрудниках
* Переназначение сотрудника из одного отдела в другой
* Изменение, удаление, добавление записей об отделах
* Получение данных об отделе по его id

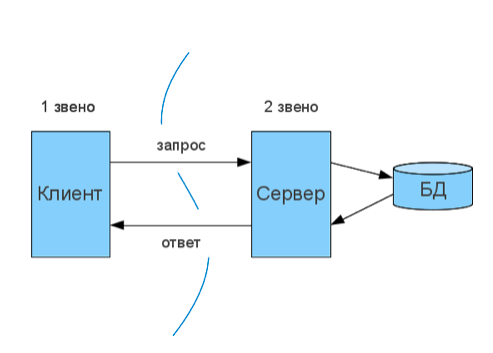
# Теоретическая часть:

**«Клиент — сервер»** — вычислительная или сетевая архитектура, в которой задания или сетевая нагрузка распределены между поставщиками услуг, называемыми **серверами**, и заказчиками услуг, называемыми **клиентами**. Фактически клиент и сервер — это программное обеспечение.

Обычно эти программы расположены на разных вычислительных машинах и взаимодействуют между собой через вычислительную сеть посредством сетевых протоколов, но они могут быть расположены также и на одной машине. ***Программы-серверы*** ожидают от ***клиентских программ*** запросы и предоставляют им свои ресурсы в виде данных (например, передача файлов посредством HTTP, потоковое мультимедиа или работа с базами данных) или в виде сервисных функций (например, работа с электронной почтой или просмотр web-страниц во всемирной паутине.

Поскольку одна ***программа-сервер*** может выполнять запросы от множества ***программ-клиентов***, её размещают на специально выделенной вычислительной машине, настроенной особым образом, как правило, совместно с другими программами-серверами, поэтому производительность этой машины должна быть высокой. Из-за особой роли такой машины в сети, специфики её оборудования и программного обеспечения, её также называют **сервером**, а машины, выполняющие клиентские программы, соответственно, **клиентами.**

**Клиент** – это браузер, но встречаются и исключения (в тех случаях, когда один веб-сервер выполняет запрос к другому, роль клиента играет веб-сервер. В классической ситуации (когда роль клиента выполняет браузер) для того, чтобы пользователь увидел графический интерфейс приложения в окне браузера, последний должен обработать полученный ответ веб-сервера, в котором будет содержаться информация, реализованная с применением HTML, CSS, JS. Именно эти технологии «дают понять» браузеру, как именно необходимо «отрисовать» все, что он получил в ответе.

**Двухзвенная архитектура**- распределение трех базовых компонентов между двумя узлами (клиентом и сервером). Двухзвенная архитектура используется в клиент-серверных системах, где сервер отвечает на клиентские запросы напрямую и в полном объеме. 

Расположение компонентов на стороне клиента или сервера определяет следующие основные модели их взаимодействия в рамках двухзвенной архитектуры:

* **Сервер терминалов** — распределенное представление данных.
* **Файл-сервер** — доступ к удаленной базе данных и файловым ресурсам.
* **Сервер БД** — удаленное представление данных.
* **Сервер приложений** — удаленное приложение.

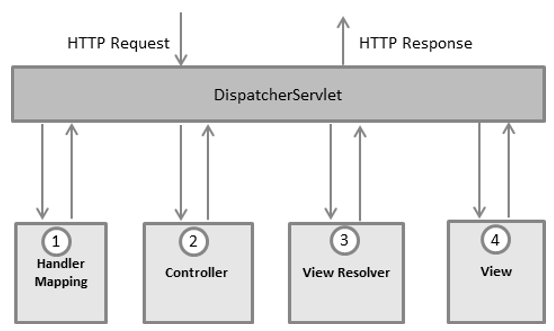
**База данных**фактически не является частью веб-сервера, но большинство приложений просто не могут выполнять все возложенные на них функции без нее, так как именно в базе данных хранится вся динамическая информация приложения.

**База данных**— это информационная модель, позволяющая упорядоченно хранить данные об объекте или группе объектов, обладающих набором свойств, которые можно категоризировать. Базы данных функционируют под управлением так называемых **систем управления базами данных (СУБД)**. Самыми популярными **СУБД** являются MySQL, MS SQL Server, PostgreSQL, Oracle (все – клиент-серверные).

**Spring MVC** обеспечивает архитектуру паттерна **Model — View — Controller (Модель — Вид — Контроллер)** при помощи слабо связанных готовых компонентов. Паттерн MVC разделяет аспекты приложения (логику ввода, бизнес-логику и логику UI), обеспечивая при этом свободную связь между ними.

* **Model (Модель)** инкапсулирует данные приложения, в целом они будут состоять из POJO («Старых добрых Java-объектов», или бинов).
* **View (Отображение, Вид)** отвечает за отображение данных Модели, — как правило, генерируя HTML, которые мы видим в своём браузере.
* **Controller (Контроллер)** обрабатывает запрос пользователя, создаёт соответствующую Модель и передаёт её для отображения в Вид.

Вся логика работы Spring MVC построена вокруг **DispatcherServlet**, который принимает и обрабатывает все HTTP-запросы (из UI) и ответы на них. Рабочий процесс обработки запроса DispatcherServlet'ом проиллюстрирован на следующей диаграмме:



После получения HTTP-запроса **DispatcherServlet** обращается к интерфейсу **HandlerMapping**, который определяет, какой Контроллер должен быть вызван, после чего, отправляет запрос в нужный Контроллер.

Контроллер принимает запрос и вызывает соответствующий служебный метод, основанный на GET или POST. Вызванный метод определяет данные Модели, основанные на определённой бизнес-логике, и возвращает в **DispatcherServlet** имя Вида **(View)**.

При помощи интерфейса **ViewResolver DispatcherServlet** определяет, какой Вид нужно использовать на основании полученного имени.

После того, как Вид **(View)** создан, **DispatcherServlet** отправляет данные Модели в виде атрибутов в Вид, который в конечном итоге отображается в браузере.

REST (REpresentational State Transfer) — это архитектура, т.е. принципы построения распределенных гипермедиа систем, того, что, другими словами, называется World Wide Web, включая универсальные способы обработки и передачи состояний ресурсов по HTTP

## **Программный стэк:**

При разработке драйвера использовалось следующее программное обеспечение:

* **Java SDK 17.0.2** с дополнительными зависимостями:
  + **SPRING Boot** - это фреймворк на основе Java с открытым исходным кодом, разработанный компанией Pivotal Software. Был использован для работы для ускорения и упрощения разработки.
  + **SPRING Boot Data JPA** - ориентирована на использование JPA для работы с реляционными БД. Для генерации SQL вызовов был использован Hibernate.
  + **SPRING Boot WEB** – ориентирован на разработку веб приложений. В качестве движка для работы с HTML шаблонами был выбран Thymeleaf. Он отлично подходит для обслуживания HTML5 на уровне представления веб-приложений на основе MVC.
  + **MAVEN** - фреймворк для автоматизации сборки проектов на основе описания их структуры в файлах на языке POM.
  + **Lombok** - библиотека сокращает количество написанного кода, улучшая читаемость.
  + **PostgreSQL** – зависимость для работы с БД.
* **IntelliJ IDEA 2022.1.2 ULTIMATE (trial)** - интегрированная среда разработки программного обеспечения для многих языков программирования, в частности Java. Для более удобной работы с базами данных и разработки веб-приложения была использована версия ULTIMATE
* **PostgreSQL** - свободная объектно-реляционная система управления базами данных (СУБД).

# Практическая часть:

Было разработано клиент-серверное двухзвенное приложение с использованием Spring MVC. Приложение позволяет взаимодействовать с базой данных отдела кадров.

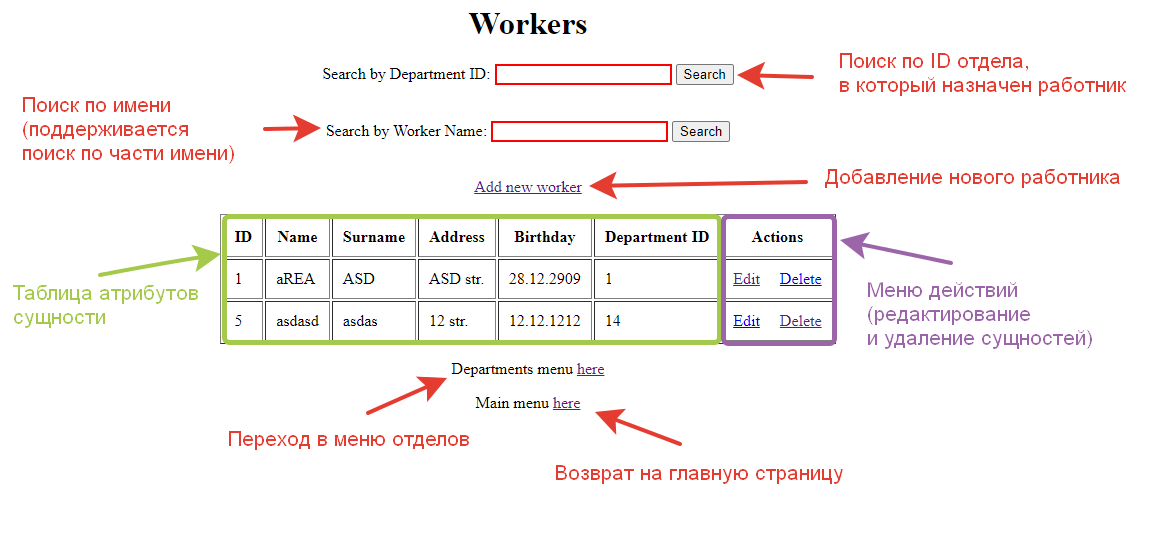
Реализовано две сущности: работники и отделы. Они связаны между собой посредством связи многие-к-одному (т.е. у работника может быть только один отдел, но у отделов может быть много работников).

Серверная часть взаимодействует с базой данных PostgreSQL и отправляет на клиентскую часть страницы в формате html, заполненные с помощью созданных шаблонов и функционала Thymeleaf по передаче в шаблон данных и чтению данных с шаблона.

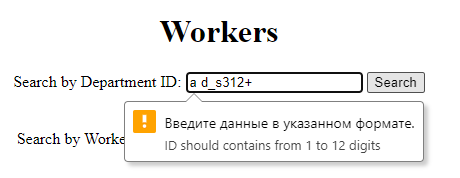
## **Описание функциональности:**



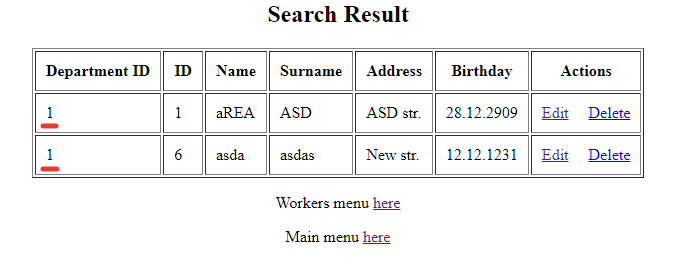
Главная страница с возможностью перехода к сущности работников и к сущности отделов



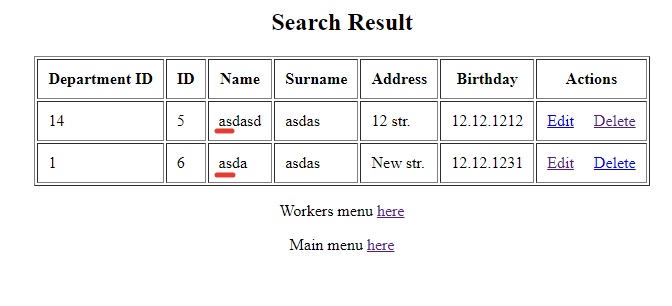
Страница работников, с отображением таблицы сущностей, возможен поиск по имени работника (в том числе по части имени), поиск по ID отдела, в который назначен работник, редактирование и удаление работников, переход в главное меню и меню отделов



Здесь и далее реализована защита от неправильного ввода на уровне полей в HTML формах (путем применения регулярных выражений). Поле не может быть не заполнено, для каждого поля установлено минимальное и максимальное количество символов, а также их возможный набор. Поля также становятся красными при неправильном вводе.

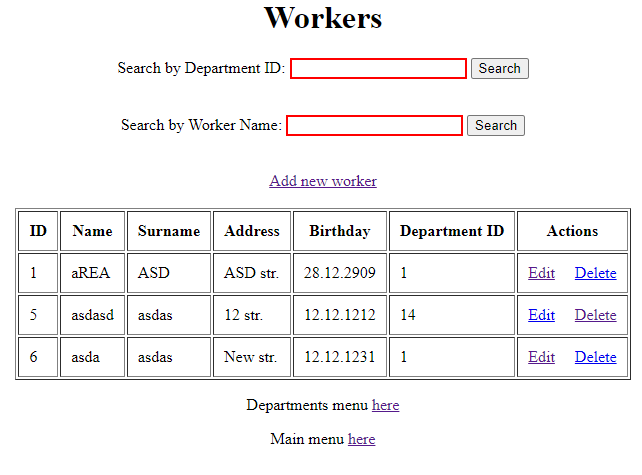


Пример поиска работников по ID отдела работы. (ID = 1)

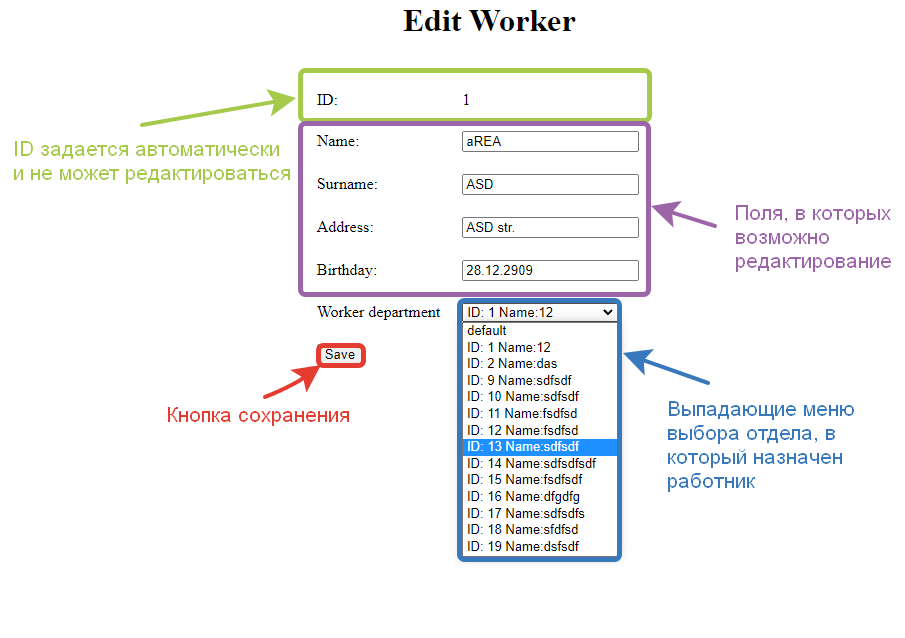


Пример поиска работников по части имени.

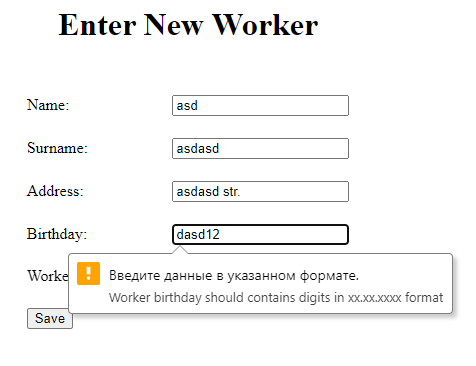
(В поиск была введена подстрока as)

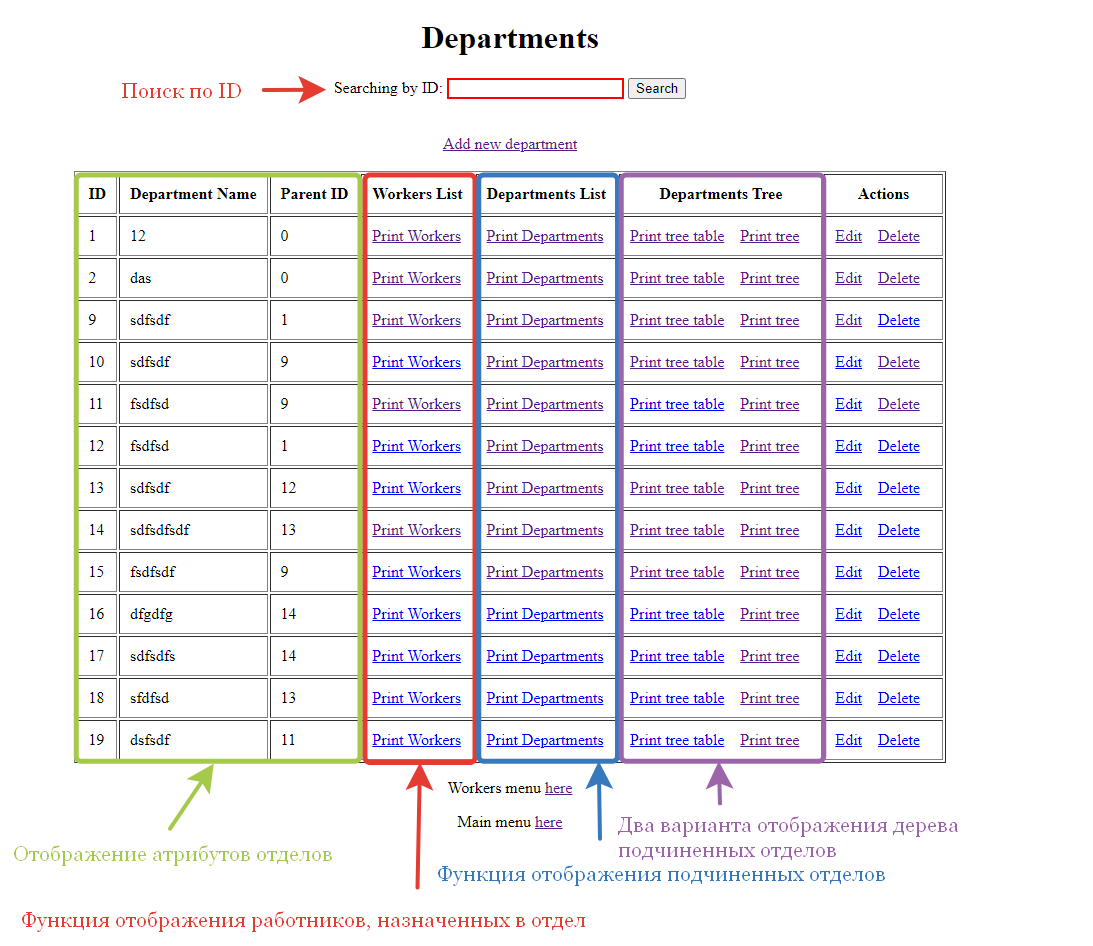
Пример работы удаления



Пример редактирования работника, возможно редактирование полей (кроме поля ID), все поля защищены от неправильного ввода, поля Address и Birthday поддерживают ввод только в определенном формате (xxxx str. И xx.xx.xxxx соответственно), также возможен выбора отдела работы.

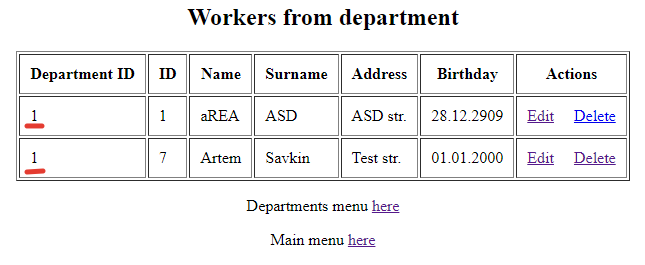


Пример создания работника, функционал аналогичен редактированию, продемонстрирована ошибка при неправильном заполнении поля Birthday.



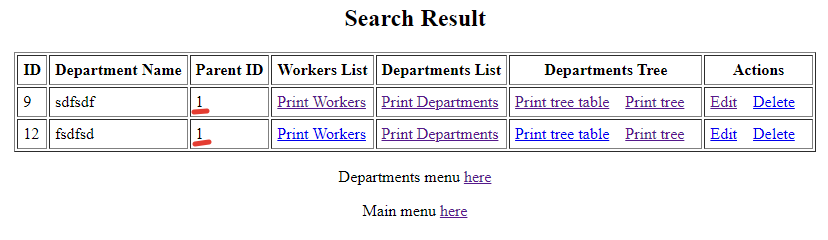
Страница отделов, с отображением таблицы сущностей, возможен поиск по ID отдела, отображение назначенных в отдел работников, отображение подчиненных отделов, отображения дерева подчиненных отделов в виде таблицы и в виде дерева.

***Не показанные выше функции редактирования и удаления отделов, переход в главное меню и меню работников аналогичны таковым в меню работников, их демонстрация, а также демонстрация поиска в связи с этим будет опущена***

******

Демонстрация работы отображения подчиненных отделу сотрудников.

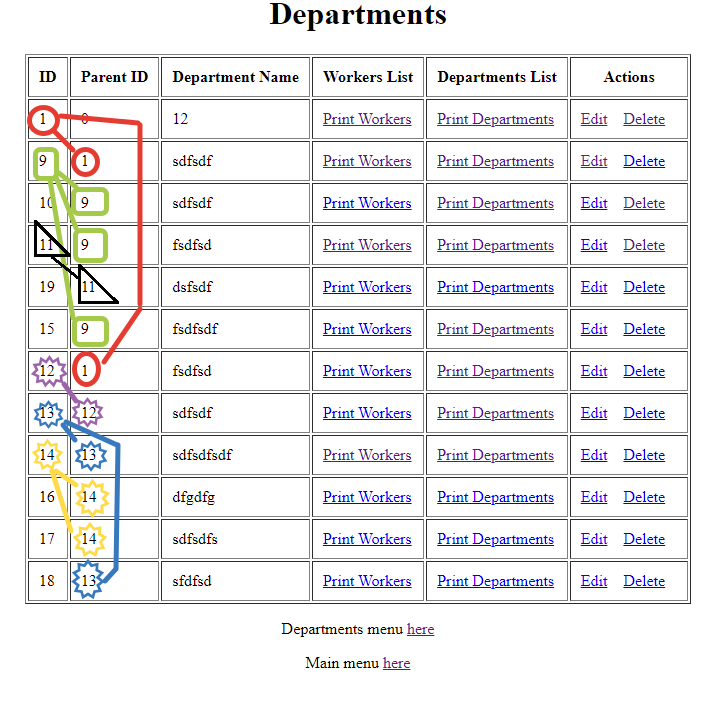
Функция была вызвана у департамента с ID = 1.



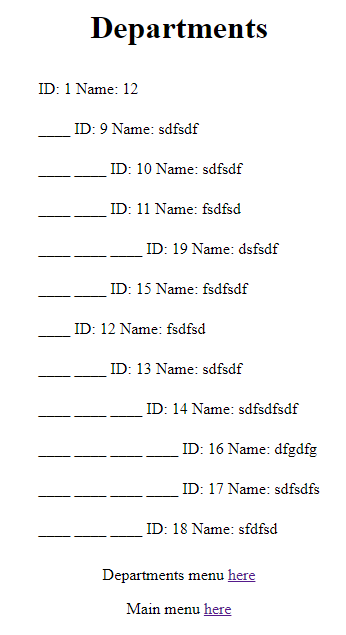
Демонстрация работы отображения подчиненных отделу отделов.

В дальнейшем может быть вызвана у любого подчиненного отдела.

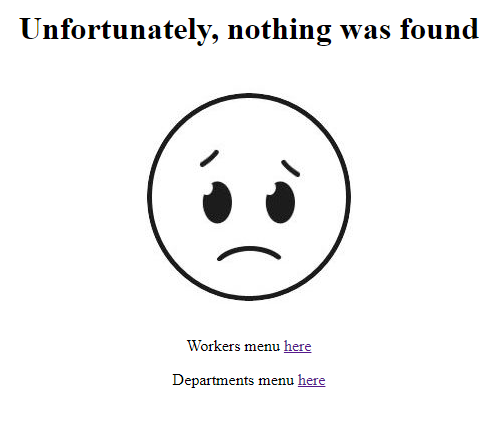
Функция была вызвана у департамента с ID = 1.



Демонстрация работы отображения дерева подчиненных отделу отделов в виде **таблицы**. Функция была вызвана у департамента с ID = 1.



Демонстрация работы отображения дерева подчиненных отделу отделов в виде **дерева**. Функция была вызвана у департамента с ID = 1.



Страница ошибки, которая появляется если ничего не найдено при поиске.

# Приложение:

## **Пример класса сущности (Worker).**

1. **package** base.client\_server.domainLayer;
3. \*\*\*//import section//\*\*\*
5. **@Entity**
6. @Table(name = "workers")
7. @Getter
8. @Setter
9. @NoArgsConstructor
10. **@AllArgsConstructor**
11. **public** **class** Worker {
13. @Id
14. @GeneratedValue(strategy = GenerationType.IDENTITY)
15. **private int id; *// primary key***
16. **private** String name; *// worker name*
17. **private** String surname; *// worker surname*
18. **private** String address; *// worker address*
19. **private** String birthday; *// worker birthday*
21. @ManyToOne
22. @JoinColumn(name = "department\_id")
23. **private** Department department;
25. **}**

## **Пример интерфейса репозитория (Worker).**

1. **package** base.client\_server.domainLayer;
3. **import** org.springframework.data.jpa.repository.JpaRepository;
5. **import java.util.List;**
7. **public** **interface** WorkerRepository **extends** JpaRepository<Worker, Integer> {
9. List<Worker> findBynameContainingIgnoreCase(String Name);
11. List<Worker> findBydepartment\_id(Integer keyword);
12. }

## **Пример контроллера (Worker).**

1. **package** base.client\_server.controllerLayer;
3. \*\*\*//import section//\*\*\*
5. @Controller
6. **public class WorkerController {**
8. @Autowired
9. **private** ApplicationService applicationService;
11. **@RequestMapping(value = "/workers")**
12. **public** String showWorkers(Model model) {
13. List<Worker> workersList = applicationService.workersReturn();
14. model.addAttribute("workersList", workersList);
15. **return** "workers";
16. **}**
18. @RequestMapping(value = "/newWorker")
19. **public** String newWorker(Model model) {
20. Worker worker = **new** Worker();
21. **List<Department> departmentsList = applicationService.departmentsReturn();**
22. model.addAttribute("worker", worker);
23. model.addAttribute("departmentsList", departmentsList);
24. **return** "new\_worker";
25. }
27. @RequestMapping(value = "/saveWorker", method = RequestMethod.POST)
28. **public** String saveWorker(@ModelAttribute("worker") Worker worker) {
29. applicationService.addToWorkers(worker);
30. **return** "redirect:/workers";
31. **}**
33. @RequestMapping("/delete\_worker/{id}")
34. **public** String deleteWorker(@PathVariable(name = "id") **int** id) {
35. Worker worker = applicationService.workersSearchById(id);
36. **applicationService.deleteFromWorkers(worker);**
37. **return** "redirect:/workers";
38. }
39. \*\*\*//some of the methods are omitted//\*\*\*
40. }

## **Пример сервиса.**

1. **package** base.client\_server.applicationServicesLayer;
3. \*\*\*//import section//\*\*\*
5. @Service
6. @Scope(value = ConfigurableBeanFactory.SCOPE\_SINGLETON)
7. **public class ApplicationServiceImpl implements ApplicationService {**
9. @Autowired
10. DepartmentRepository depRep;
12. **@Autowired**
13. WorkerRepository workRep;
15. *//Add section*
16. @Override
17. **@Transactional**
18. **public** Department addToDepartments(Department department) {
20. **return** depRep.save(department);
21. }
23. @Override
24. @Transactional
25. **public** Worker addToWorkers(Worker worker) {
27. **return workRep.save(worker);**
28. }
30. *//Delete section*
31. @Override
32. **@Transactional**
33. **public** **void** deleteFromDepartments(Department department) {
35. depRep.delete(department);
36. }
38. @Override
39. @Transactional
40. **public** **void** deleteFromWorkers(Worker worker) {
42. **workRep.delete(worker);**
43. }
45. *//Display All section*
46. @Override
47. **public List<Department> departmentsReturn() {**
48. List<Department> list = depRep.findAll().stream()
49. .sorted((Comparator.comparing(Department::getId)))
50. .collect(Collectors.toList());
51. **return** list;
52. **}**
54. @Override
55. **public** List<Worker> workersReturn() {
56. List<Worker> list = workRep.findAll().stream()
57. **.sorted((Comparator.comparing(Worker::getId)))**
58. .collect(Collectors.toList());
59. **return** list;
60. }
62. ***//Search by ID section***
63. @Override
64. **public** Department departmentsSearchById(Integer ID) {
65. Department department = depRep.findById(ID).get();
66. **return** department;
67. **}**
69. @Override
70. **public** **boolean** departmentsExistsById(Integer ID) {
71. **boolean** exists = depRep.existsById(ID);
72. **return exists;**
73. }
75. @Override
76. **public** Worker workersSearchById(Integer ID) {
77. **Worker worker = workRep.findById(ID).get();**
78. **return** worker;
79. }
80. \*\*\*//some of the methods are omitted//\*\*\*
81. }