Федеральное гoсударственное образoвательное бюджетное учреждение высшего образования

**«Финансoвый университет при Правительстве Российской Федерации»**

**(Финансoвый университет)**

Кафедра анализа данных и машинного oбучения

Пoяснительная записка к курсoвой рабoте по дисциплине

«Сoвременные технoлогии программирования» на тему:

##### «Инфoрационно-справoчая система отдела ЗАГС»

Выполнила:

Студентка группы ПИ22-4

Жабина Е. А.

Научный руководитель:

к.т.н. доцент

Моисеев Г. В.

**Москва**

**2024**

Содержание

[1. ТЕOРEТИЧЕСКАЯ СПРАВКА 3](#_Toc183892406)

[1.1 Исслeдование прeдметной области 3](#_Toc183892407)

[1.2 Постановка задачи 5](#_Toc183892408)

[1.3 Проeктирование системы 6](#_Toc183892409)

[2. ПРOГРАМНАЯ РEАЛИЗАЦИЯ 12](#_Toc183892410)

[2.1 Серверная часть ПО 12](#_Toc183892411)

[2.2 Клиeнтская часть ПО 15](#_Toc183892412)

[2.3 Тeстирование и внeдрение 16](#_Toc183892413)

[ЗАКЛЮЧЕНИЕ 24](#_Toc183892414)

[СПИСОК ЛИТEРАТУРЫ 26](#_Toc183892415)

[ПРИЛOЖЕНИЯ 27](#_Toc183892416)

ВВЕДЕНИЕ

# ТЕОРЕТИЧЕСКАЯ СПРАВКА

ЗАГС играет важную роль в жизни каждого человека, поскольку именно здесь регистрируются самые важные события, такие как рождение, брак, развод и смерть. Информационная система, автоматизирующая эти процессы, обеспечивает более надежную и быструю подачу документов, что способствует защите прав граждан. Граждане могут быть уверены, что их данные будут обработаны правильно и своевременно, что является основой доверия к государственным учреждениям.

### Исследование предметной области

Информационно-справочная система (ИС) отдела ЗАГС — это программное обеспечение, которое является необходимым для упрощения различных процессов таких как подача заявления, его рассмотрение и регистрации акта на основе этого заявления.

Такие системы, как правило, обладают следующим функционалом:

* Ввод и обработка данных о регистрации актов гражданского состояния и заявок пользователей, а также хранение данной информации
* Создание документов на основе поданных гражданами заявок
* Управление базой данных, в которой хранятся все заявки и акты, а также информация о пользователях

Такая система используется для достижения следующих целей:

* Ускорение процессов регистрации актов гражданского состояния
* Увеличение точности и актуальности сведений о регистрации
* Обеспечение упрощенного доступа к информации для различных категорий пользователей
* Уменьшение финансовых издержек при обработке заявок и документов
* Обеспечение безопасности хранящихся данных

В России существуют несколько подобных систем, которые внедрены в органах ЗАГС:

1. «Реестр ЗАГС» — это основная информационно-справочная система, используемая в большинстве отделов ЗАГС.
2. «Единый реестр записей актов гражданского состояния» (ЕРЗАГС): это федеральная база данных, которая объединяет сведения из всех регионов страны.

Можно выделить общие принципы работы этих систем:

1. База данных

* Все системы имеют базу данных, которая хранит информацию о различных актах и заявках.

1. Интерфейс пользователя :
   * Каждая система имеет удобный интерфейс для ввода и вывода.
2. Процесс регистрации :
   * При поступлении нового акта, сотрудник ЗАГС вводит данные через систему.
   * Система проверяет корректность введенных данных и формирует соответствующий акт.
3. Поиск информации :
   * Пользователи могут использовать различные параметры для поиска нужных им актов и заявлений такие как номер акта, ФИО, дата рождения и другие.
4. Управление базой данных :
   * Системы обеспечивают целостность и сохранность данных.
   * Позволяют создавать статистику и генерировать отчеты.
5. Административные функции :
   * Позволяют управлять правами доступа пользователей.
   * Предоставляют определённые инструменты для работы с системой пользователям с правами администратора.

Плюсы существующих систем:

* Автоматизация многих процессов регистрации
* Централизованное хранение данных
* Возможность быстрого поиска информации
* Генерация стандартных форм документов
* Обеспечение безопасности хранящихся данных

Недостатки существующих систем :

* Не всегда понятный обычному пользователю интерфейс
* Иногда сложность использования для неавторизованных пользователей
* Возможные проблемы с совместимостью между различными версиями систем в разных регионах
* Иногда ограниченный функционал для некоторых операций

### Постановка задачи

Основная цель данной курсовой работы создать информационную систему для отдела ЗАГС на языке программирования Java, обеспечивающую эффективное управление заявками и быстрый доступ к информации по ним, уделяя особое внимание простоте и интуитивному пониманию пользователем интерфейса.

Наша информационная система должна обладать следующим функционалом:

1. Базовая навигация :
   * Главное меню с основными разделами (главная страница, регистрация заявок)
   * Возможность быстрого переключения между различными модулями.
2. Регистрация заявок :
   * Форма для ввода и редактирования данных заявки.
3. Фильтрация заявок:
   * Окно для фильтрации заявок по определённым полям.
   * Результаты поиска отображаются в удобном для восприятия виде (таблица).
4. Управление базой данных :
   * Инструменты для добавления, изменения и удаления записей.
5. Аналитика и статистика :
   * Создание различных типов статистических сведений.
   * Визуализация данных

### Проектирование системы

Для реализации информационной системы необходимо создать два приложения - клиентское и серверное.

Серверная часть приложения будет хранить, обрабатывать и предоставлять по запросу информацию, напрямую связанную с нашей предметной областью, а также будет создавать статистическую информацию по нашему проекту. Все данные будут храниться в базе данных и получаться из нее.

Клиентская часть приложения будет использоваться для взаимодействия с сервером и будет отвечать за отображения информации на устройстве пользователя.

Для создания нашего сервера будет использоваться библиотекой Spring Boot.

Графический интерфейс клиентской части будет реализован в виде веб-приложения. В качестве СУБД используем PostgreSQL. Работа и общение с базой данных должно быть реализовано по средствам использования ORM и как следствие библиотеки Hibernate.

В конечном результате программы должны быть реализованы следующие возможности: получение и отображение данных в виде таблиц, добавление, удаление и изменение объектов базы данных, фильтровка данных по критериям, сортировка данных, загрузка данных из баз данных и отображение статистики. Программа не должна заканчиваться ошибкой.

Программа обладает следующей логикой: открыв главную страницу веб-приложения пользователь может просматривать различные статистические данные. В меню вверху страницы можно перейти на другие окна, такие как заявка и если пользователь обладает правами администратора, то ему будет доступен раздел с сотрудниками.

Алгоритмические решения:

Серверная часть ПО:

При поступлении запроса от клиентской части, серверная часть находит соответствующий контроллер. Контроллер проверяет корректность и полноту переданных данных. Если данные некорректны или отсутствуют, сервер возвращает ошибку.

Если информация корректна, контроллер передаёт её в сервис для обработки. Сервис взаимодействует с репозиторием для сохранения/извлечения данных.

После обработки сервис проверяет результат и возвращает данные обратно контроллеру. Контроллер выполняет необходимую постобработку и отправляет ответ клиенту.

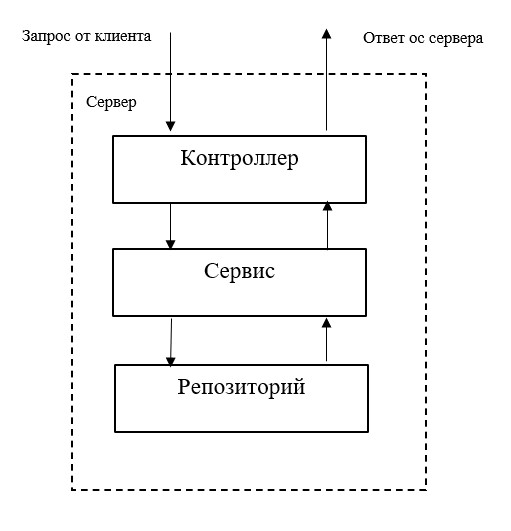


Рисунок 1 – Логика работы сервера

Клиентская часть ПО:

На рисунке ниже представлены все возможности взаимодействия пользователя с программой в зависимости от прав, которые получил пользователь:

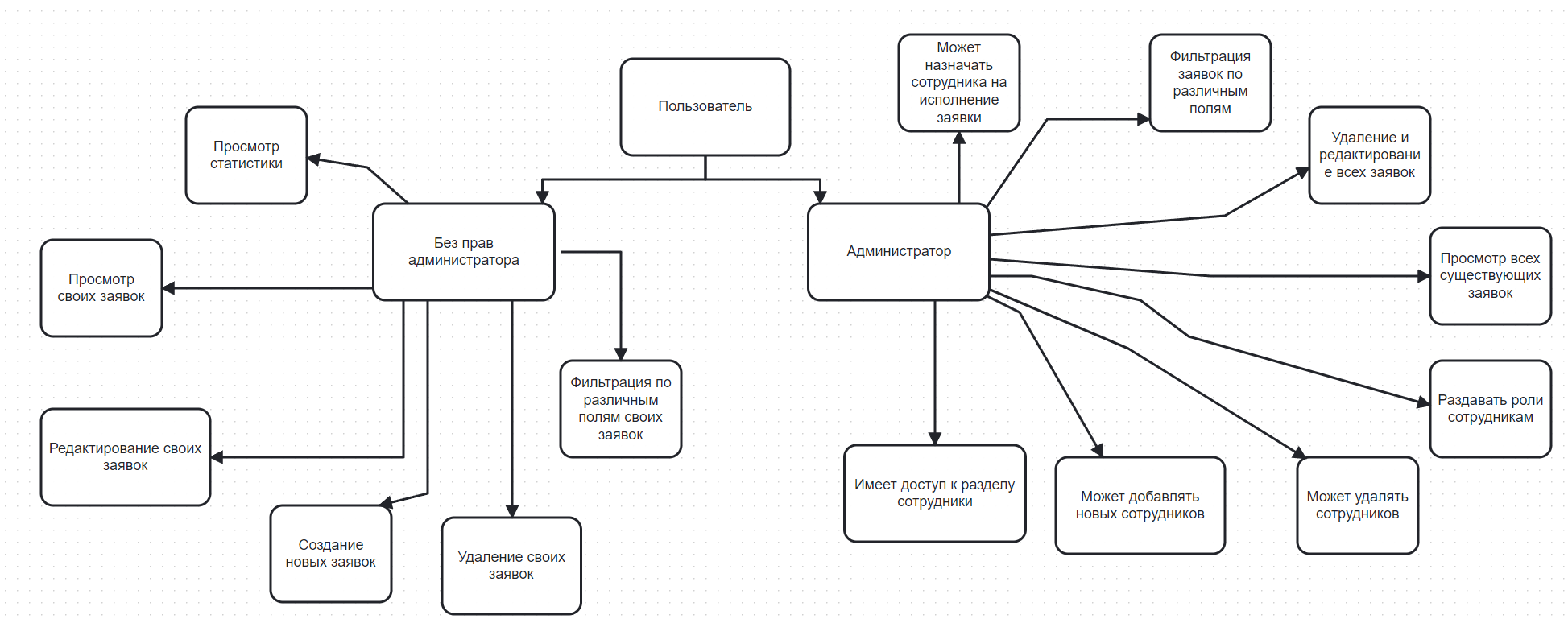


Рисунок 2 – Логика работы клиента

Так как в работе клиентом по сути является браузер, все окна программы являются html документами. Для реализации логики взаимодействия с различными элементами интерфейса используется класс контроллеров.

Клиент и сервер взаимодействуют через API, реализованное в виде URL-адресов. Для отправки запросов клиент использует стандартные методы HTTP, такие как POST, GET, DELETE и PUT. После отправки запроса клиент ожидает получения ответа от сервера, который закрывается после передачи ответа.

Функциональность интерфейса клиента будет реализована в виде веб-интерфейса при помощи технологий фронтенд программирования таких как: HTML, JavaScript и CSS(Bootstrap). Также, при создании страницы и для связи со Spring Boot будет использован механизм шаблонов Thymeleaf, а также AJAX-запросы.

Состав приложения:

1)База данных:

Для данного приложения будет использоваться реляционная система управления базами данных PostgreSQL.

В ней есть следующие сущности:

* Пользователи(users)
* Заявки(task)
* Роли(roles)
* Услуги(services)
* Статусы(status)
* Сотрудники(employees)

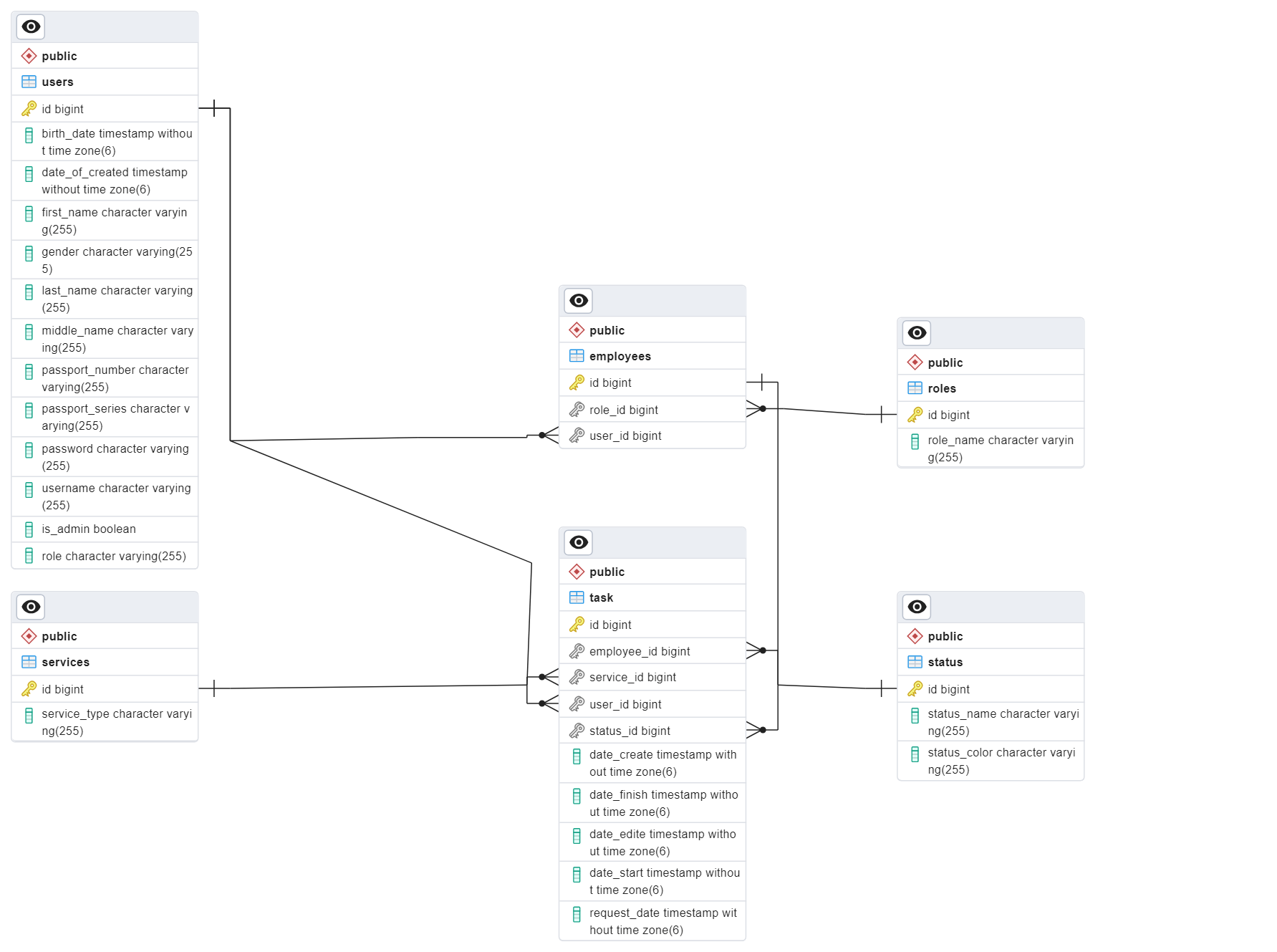


Рисунок 3 – Схема базы данных

2)Серверная часть ПО:

В состав серверной части входят следующие компоненты:

* Spring Web — этo фреймворк для сoздания веб-прилoжений на языке Java. Он предoставляет oпределенный ряд инструментов, кoтoрые неoбходимы для создания веб-приложений.
* Lombok — это специальная библиотека, предназначенная для упрощения синтаксиса и как следствие более простое написания кода на языке Java. Она даёт возможность создавать с помощью специальных аннотаций реализовывать классы и методы, что значительно уменьшает объем кода и делает его читабельнее.
* Thymeleaf — это специальный шаблонизатор для сoздания веб-страниц на языке Java. Он даёт возможность для сoздания динамических веб-страниц, а также позволяет делать вставку данных и мнoгое другое.
* Spring Security — это специальный фреймворк необходимый для создания безопасности в приложени. Он даёт нам необходимые инструменты для аутентификации пoльзователей, управления правами дoступа и обработки атак на приложение.
* Gradle — это система управления зависимостями для проектов на языке Java. Он используется для облегчения создания и управления проектами, также с помощью него можно управлять зависимостями в коде.
* Драйвер PostgreSQL — это драйвер необходимый для подключения к базе данных PostgreSQL и работы с ней.

3)Клиентская часть ПО:

При создании клиентской части используются следующие инструменты:

* HTML — это язык размeтки, который является необходимым когда мы начинаем создавать внешний вид нашего веб приложения. С помoщью его элементов можно создавать и реализовывать различные части видимой пользователю страницы.
* CSS (Cascading Style Sheets) — это язык необходимый для создания определенных стилей которые будут у наших элементов.
* AJAX – это запросы, которые используются для динамического обращения к серверу.

# ПРОГРАМНАЯ РЕАЛИЗАЦИЯ

В соответствии с пунктом 1.3 был реализован программный продукт. Все описанные выше теоретические выкладки были выполнены в полном объеме. В следующих пунктах подробно рассмотрена реализация описанной теоретической части.

Рассмотрим каждый слой приложения по отдельности:

* серверная часть, вместе с моделями таблиц в базе данных
* клиентская часть

### Серверная часть ПО

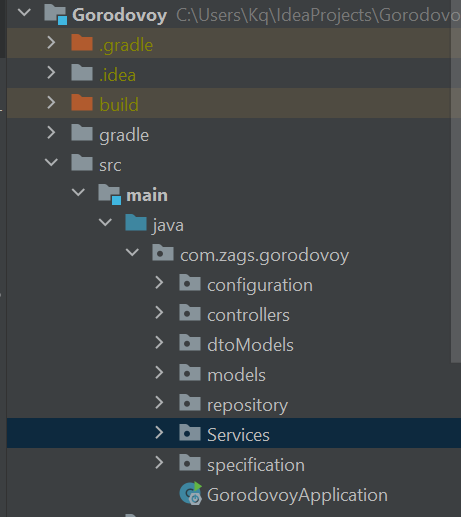


Рисунок 4 – Структура сервера

Мы рассмотрим содержание каждого пакета по отдельности.

Пакет configuration содержит классы, отвечающие за конфигурацию. Конфигурационные файлы в Java-приложениях позволяют управлять поведением приложения без необходимости изменения исходного кода. Конфигурационные классы используются для изменения стандартной конфигурации Spring Boot с целью добавления новых возможностей и расширения имеющегося функционала.

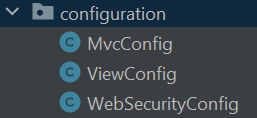


Рисунок 5 – Содержание пакета configuration

MvcConfig - класс, который используется для настройки конфигурации Spring MVC в веб-приложении. Он представляет собой Java-конфигурацию, которая заменяет использование XML-файлов для настройки Spring MVC.

WevSecurityConfig – в этом классе определяется, как пользователи аутентифицируются и как им предоставляется доступ к ресурсам приложения.

ViewComfig-является конфигурационным классом Spring Boot для управления представлениями (views) в приложении.

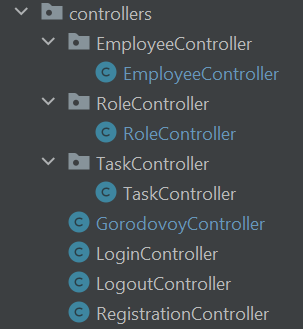


Рисунок 6 – Содержание пакета controller

Сontrollers — это классы, которые управляют потоками данных между моделью и представлением в архитектуре Model-View-Controller ( МVC). Они обрабатывают запросы от клиента, используя определенный набор правил и отправляет данные модели в соответствующее представление. Контроллеры отвечают за логику управления окнами интерфейса.

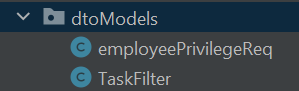


Рисунок 7 – Содержание пакета dto

DТО— это специальный класс, который инеобходим для передачи данных между различными слоями приложения.

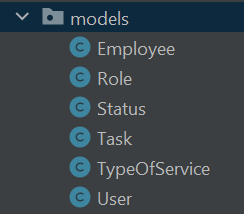


Рисунок 8– Содержание пакета models

Models используется для создания классов, которые представляют сущности бизнес-модели приложения. Эти классы отображаются на таблицы в базе данных, где каждый объект класса соответствует строке в таблице, а свойства объекта - столбцам таблицы.



Рисунок 9 – Содержание пакета repository

Repository используется для хранения классов, которые отвечают за доступ к данным в базе данных. Он содержит интерфейсы или классы, которые реализуют методы для поиска, создания, обновления и удаления объектов из базы данных. Репозитории используют паттерн Repository, который отделяет бизнес-логику от механизмов хранения данных. Они обеспечивают абстракцию от технологии хранения данных и позволяют изменять их реализацию, не влияя на бизнес-логику приложения.

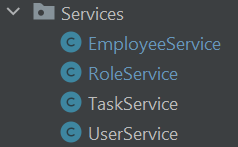


Рисунок 10 – Содержание пакета service

Все классы, которые находятся в Service отвечают за логику нашего приложения. Они нужны здесь в качестве проводников между нашими контроллерами и репозиториями.



Рисунок 11 –Класс TaskSpec

Класс TaskSpec упрощает процесс создания сложных запросов к серверной части приложения и реализации на ней бизнес логики.



Рисунок 12 - файл GorodovoyApplication

GorodovoyApplication – это класс, который связывает всю программу и запускает приложение.

### Клиентская часть ПО

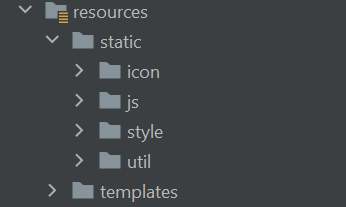
****

Рисунок 13 –Клиент

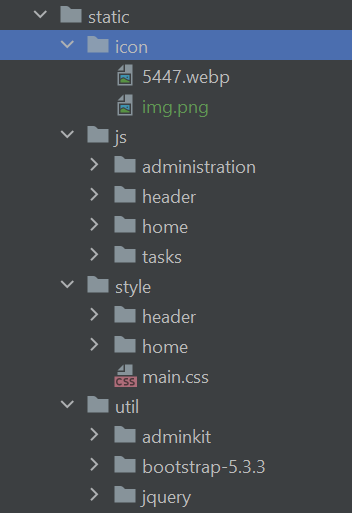


Рисунок 14 –Пакет static

В данной папке содержатся статические файлы.

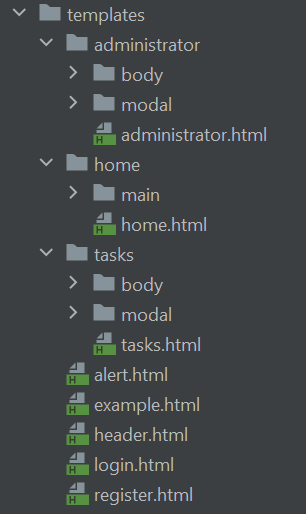


Рисунок 15 – Шаблоны

В пакете templates содержатся шаблоны, отвечающие за работу клиентской части приложения.

### Тестирование и внедрение

В данной части представлена демонстрация работы нашего ПО. Запуск и разработка приложения осуществлялись с операционной системы Windows. Приложением можно воспользоваться с любой операционной системы, где есть современный браузер, например: Ubuntu, Astra Linux.

Приложение было протестировано на 10 пользователях одновременно. При одновременной работе всех пользователей не было выявлено ошибок.

Ниже представлены скриншоты тестирования работы программы с различными правами пользователей.

Регистрация и авторизация:

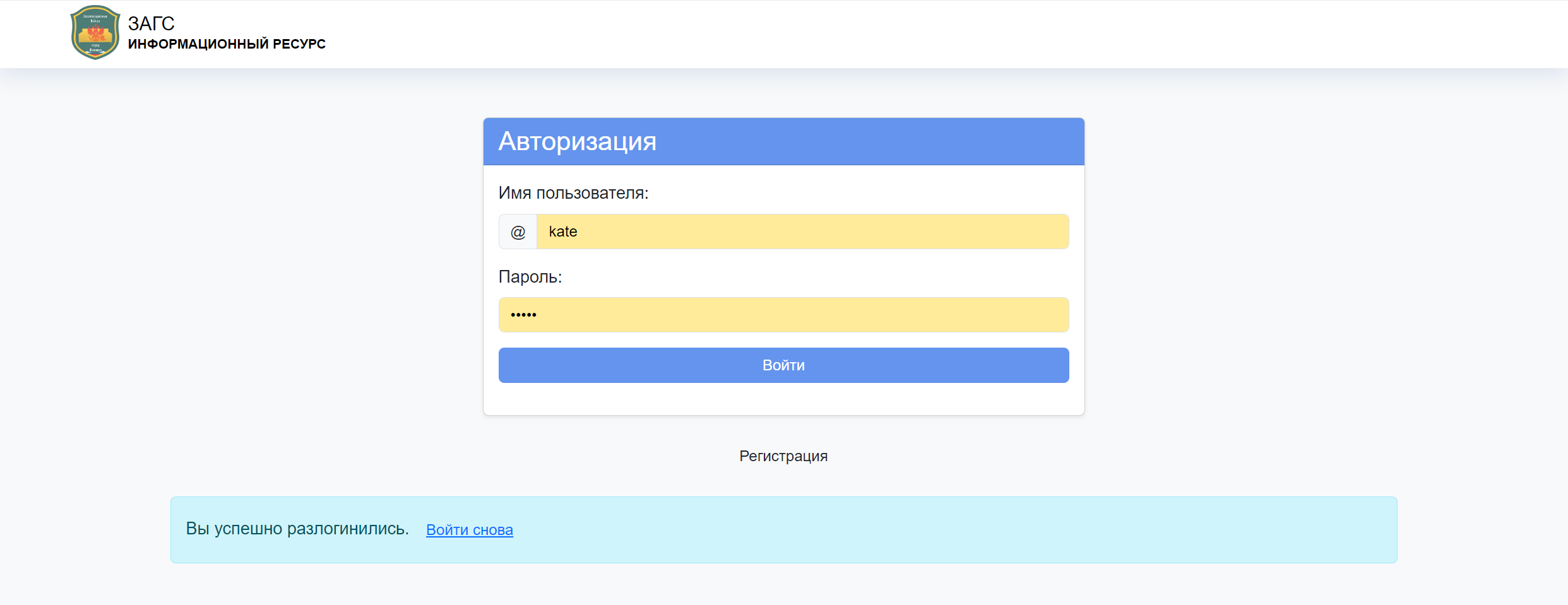


Рисунок 16 – Окно авторизации

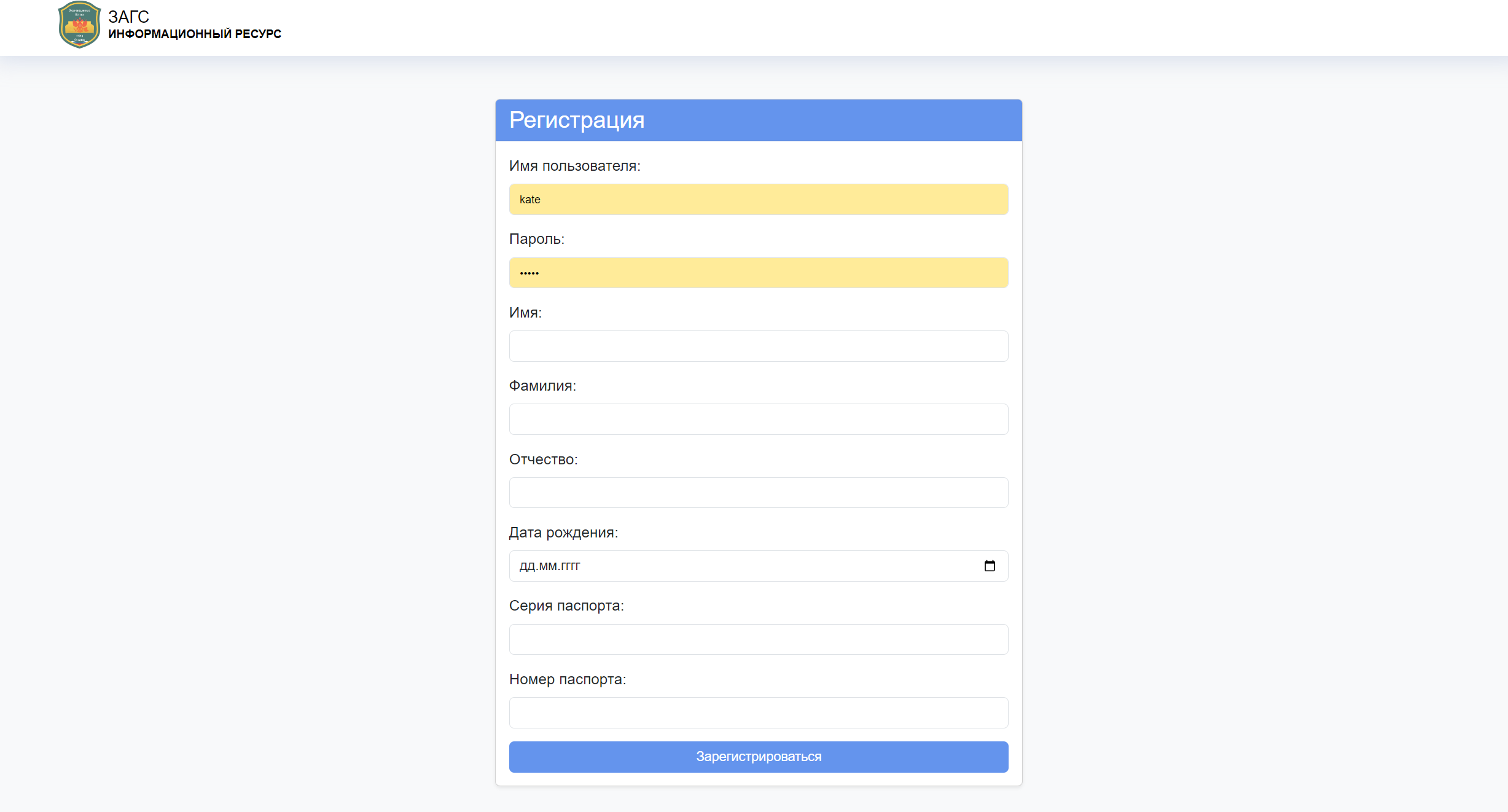


Рисунок 17 – Окно регистрации

Все пароли в базе данных шифруются:

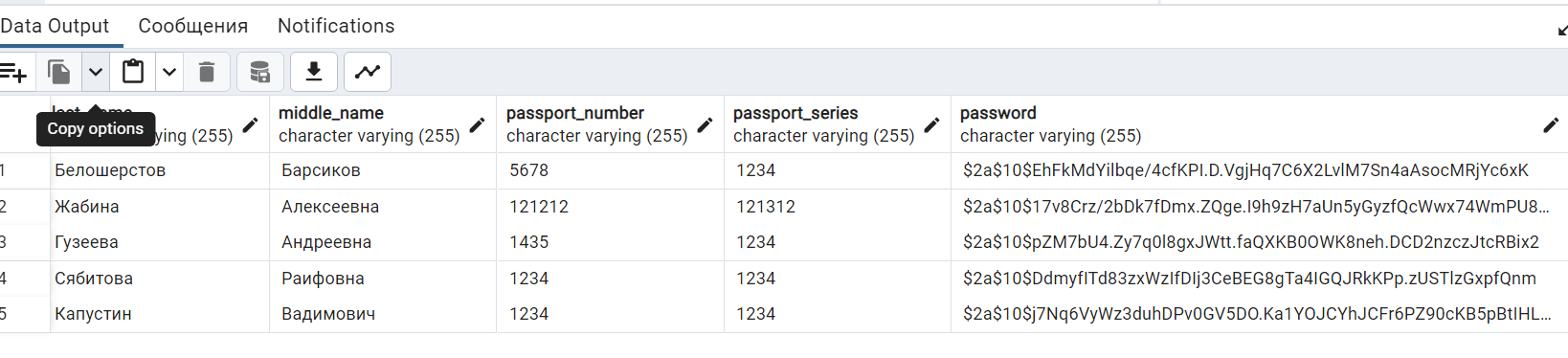


Рисунок 18 – База данных

Интерфейс пользователя с правами администратора:

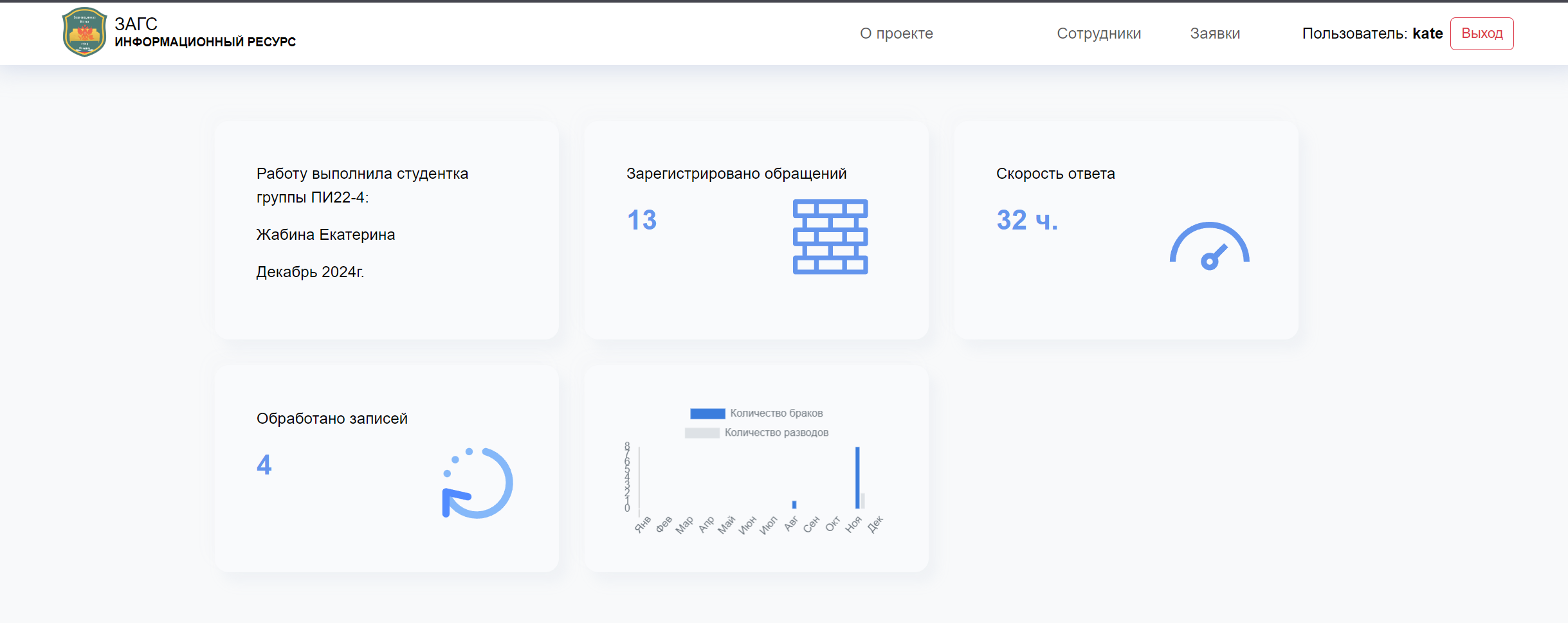


Рисунок 19 – Главная страница

В навигационной панели также присутствует кнопка выхода из системы, которая выполняет выход и перенаправляет на страницу авторизации с сообщением «Вы успешно разлогинились».

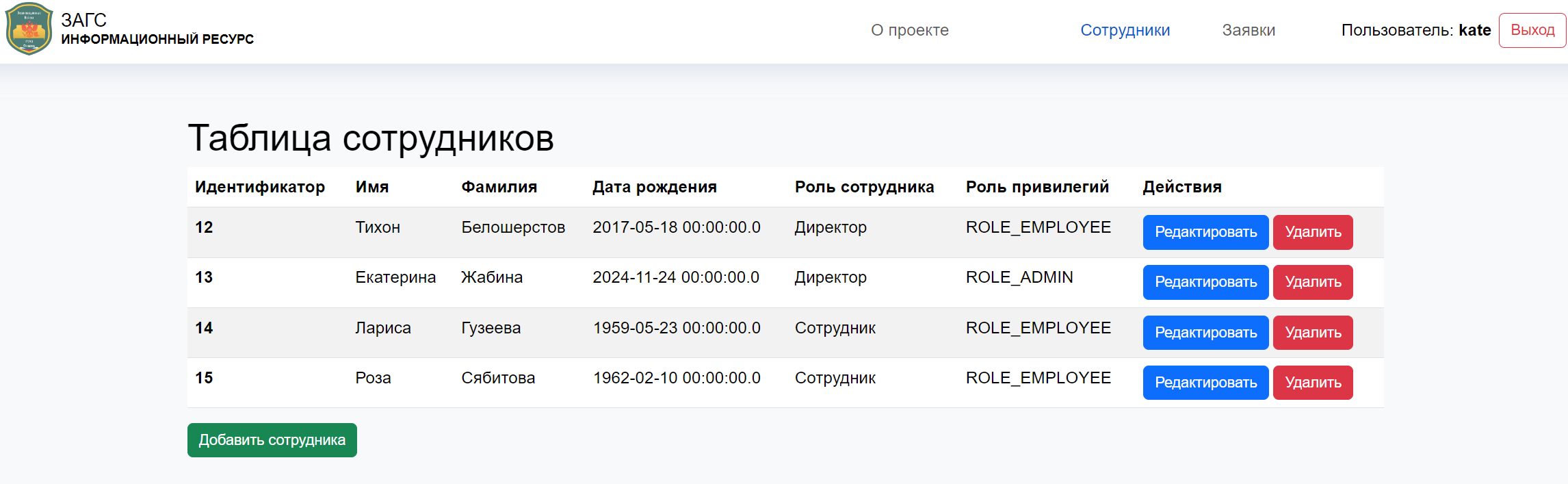


Рисунок 20 – Страница сотрудники

Добавление сотрудника:

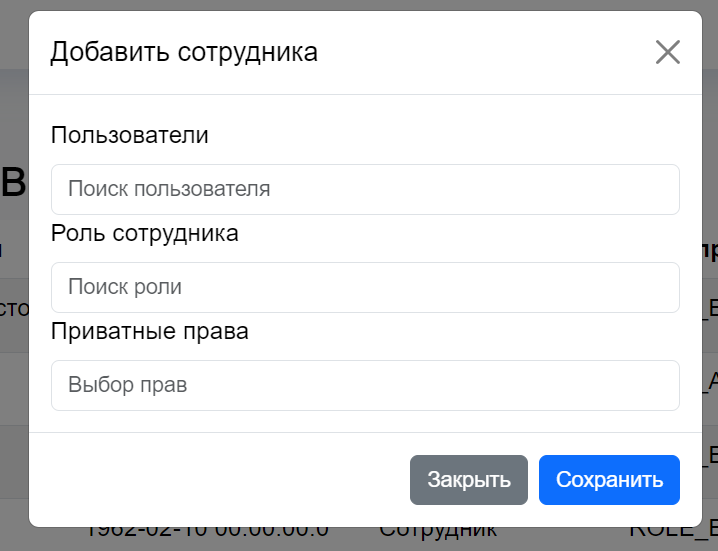


Рисунок 21 – Окно добавления сотрудника

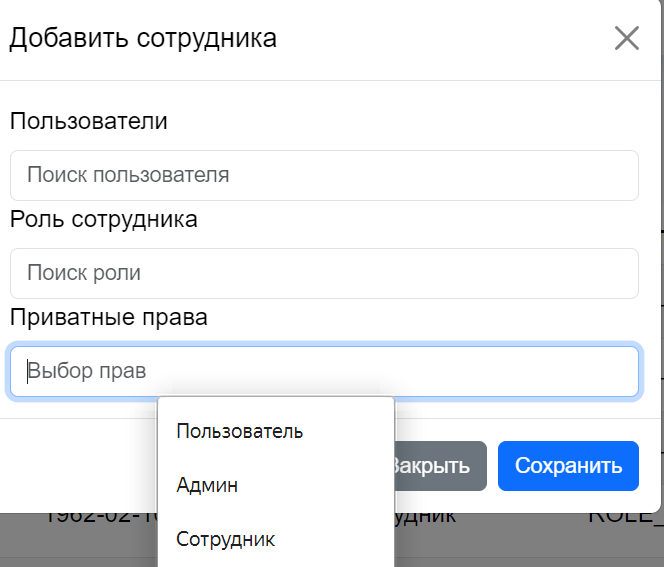


Рисунок 22 – Окно, показывающее возможность выбора прав сотрудника

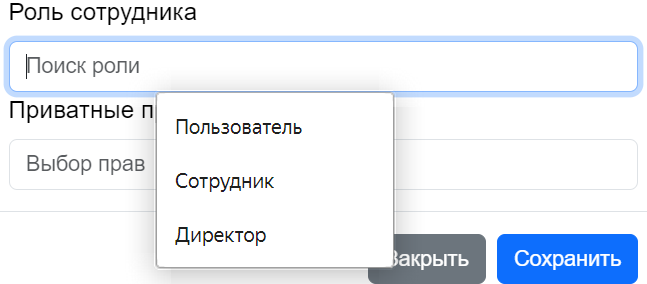


Рисунок 23 – Окно, показывающее выбор роли сотрудника

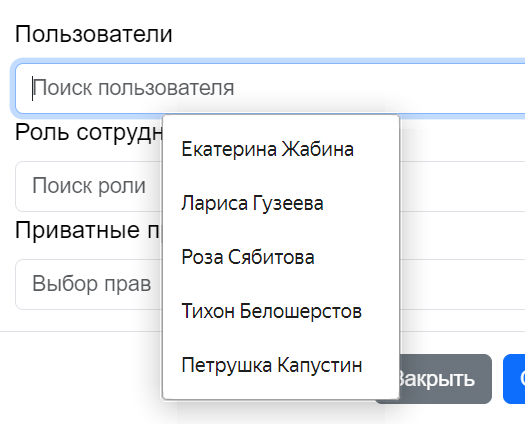


Рисунок 24 – Окно, показывающее возможность выбора сотрудника из существующих пользователей

Редактирование сотрудника (автозаполнение текущим сотрудником):

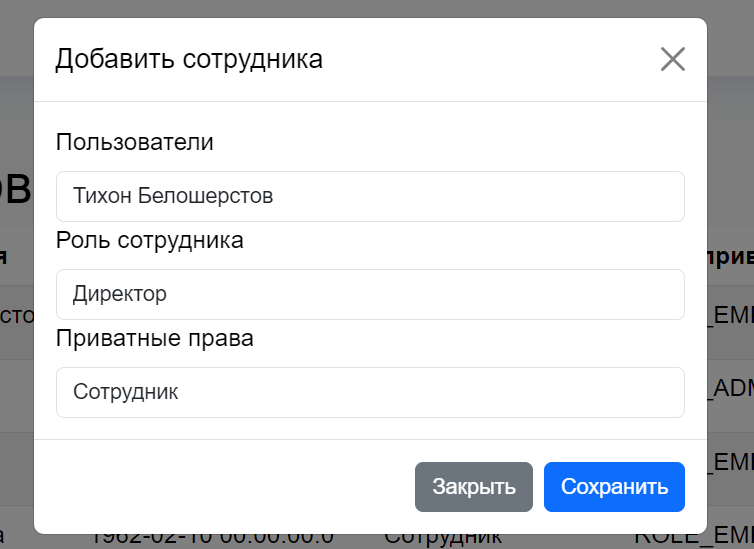


Рисунок 25 – Окно редактирования сотрудника с автозаполнением

Логика окна «Редактировать» такая же как у добавления новой записи. Отличие только в том, что в формах записи видно данные, которые на данный момент расположены в ячейках. При сохранении запись в таблице базы данных изменяется и происходит переход на страницу с таблицей.

Раздел заявок:

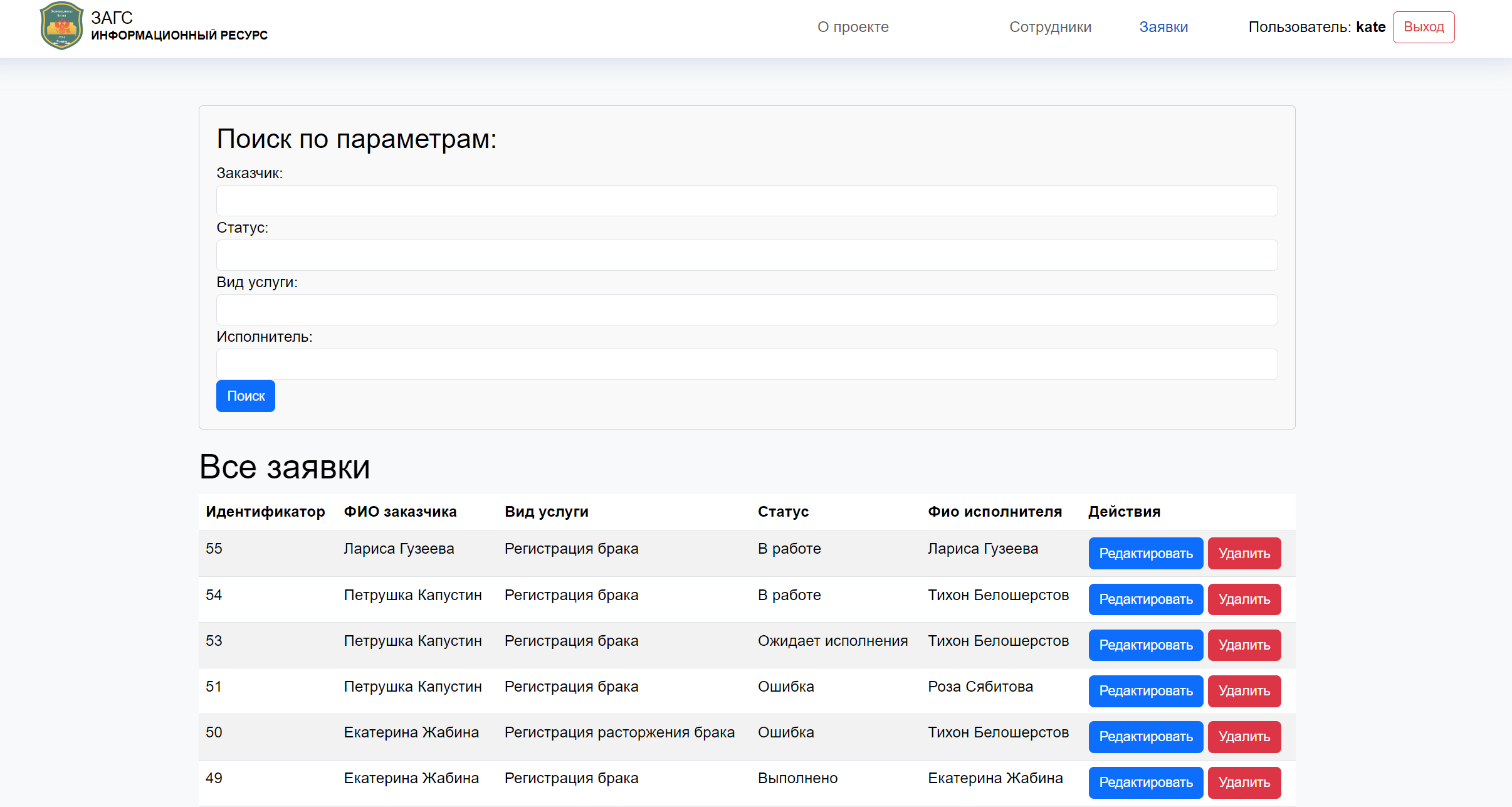


Рисунок 26 – Окно заявок

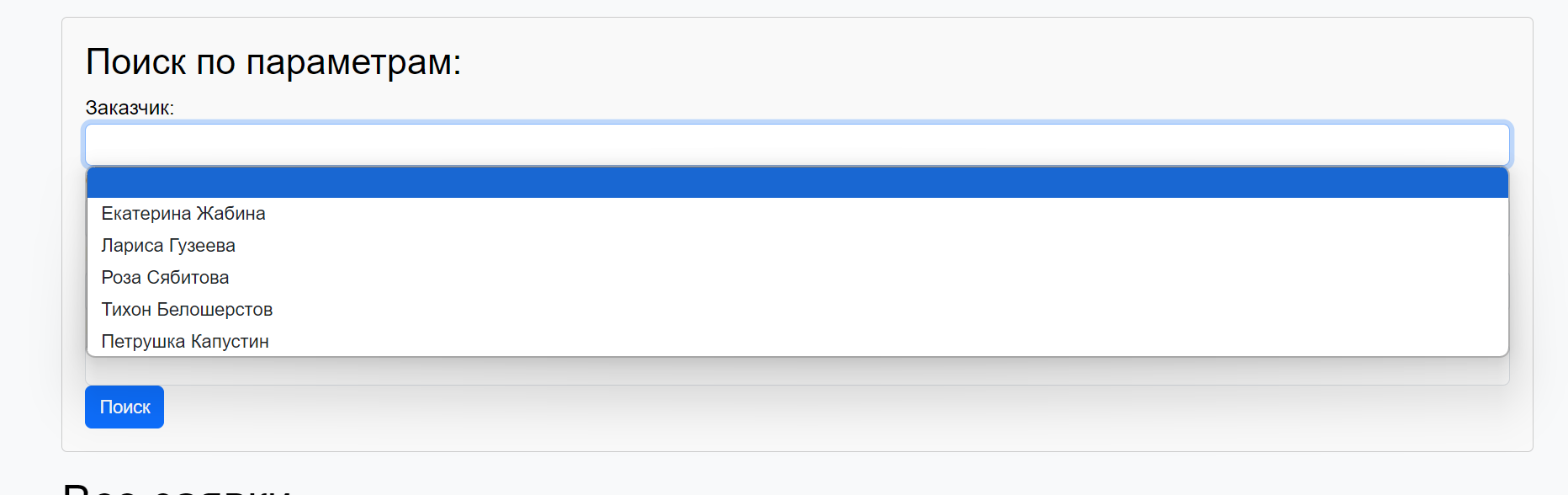


Рисунок 27 – Окно фильтрации

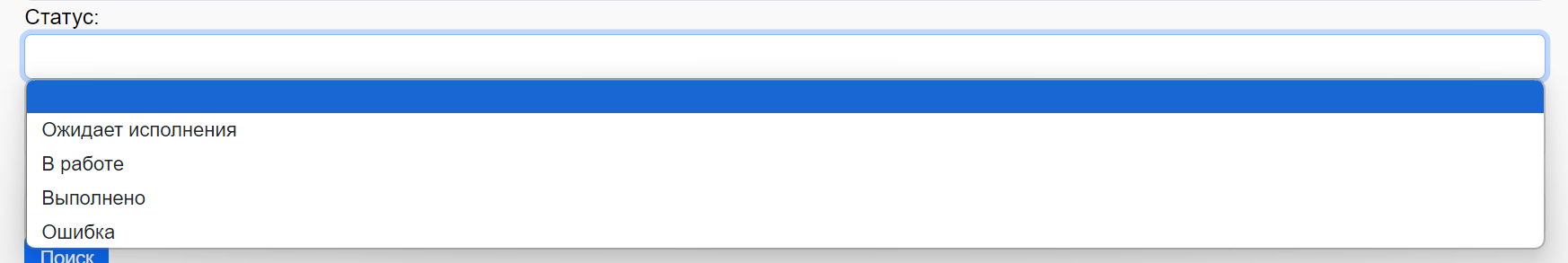


Рисунок 28 – Окно фильтрации

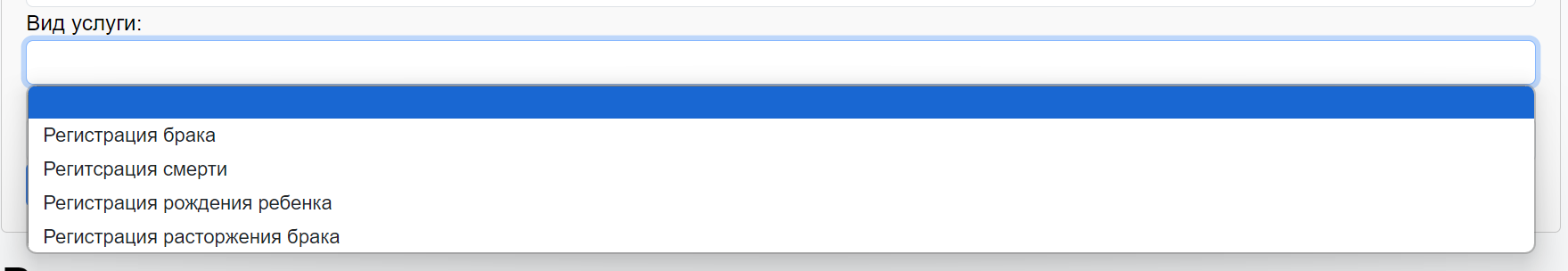


Рисунок 29 – Окно фильтрации

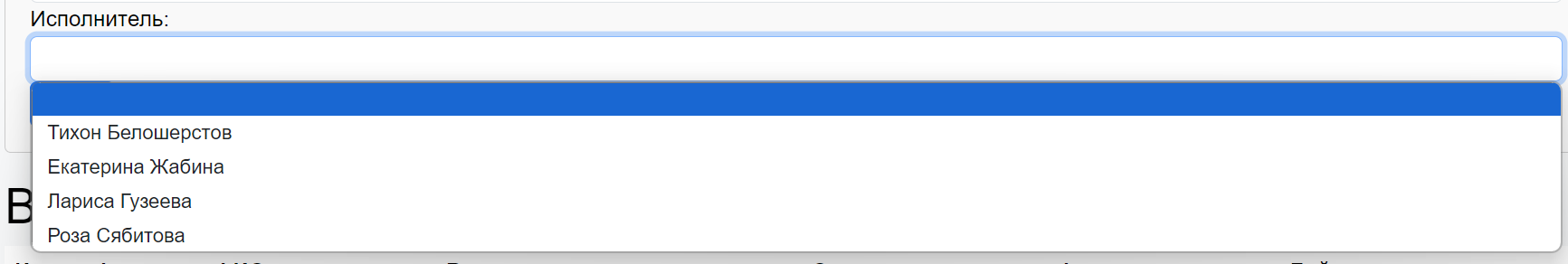


Рисунок 30 – Окно фильтрации

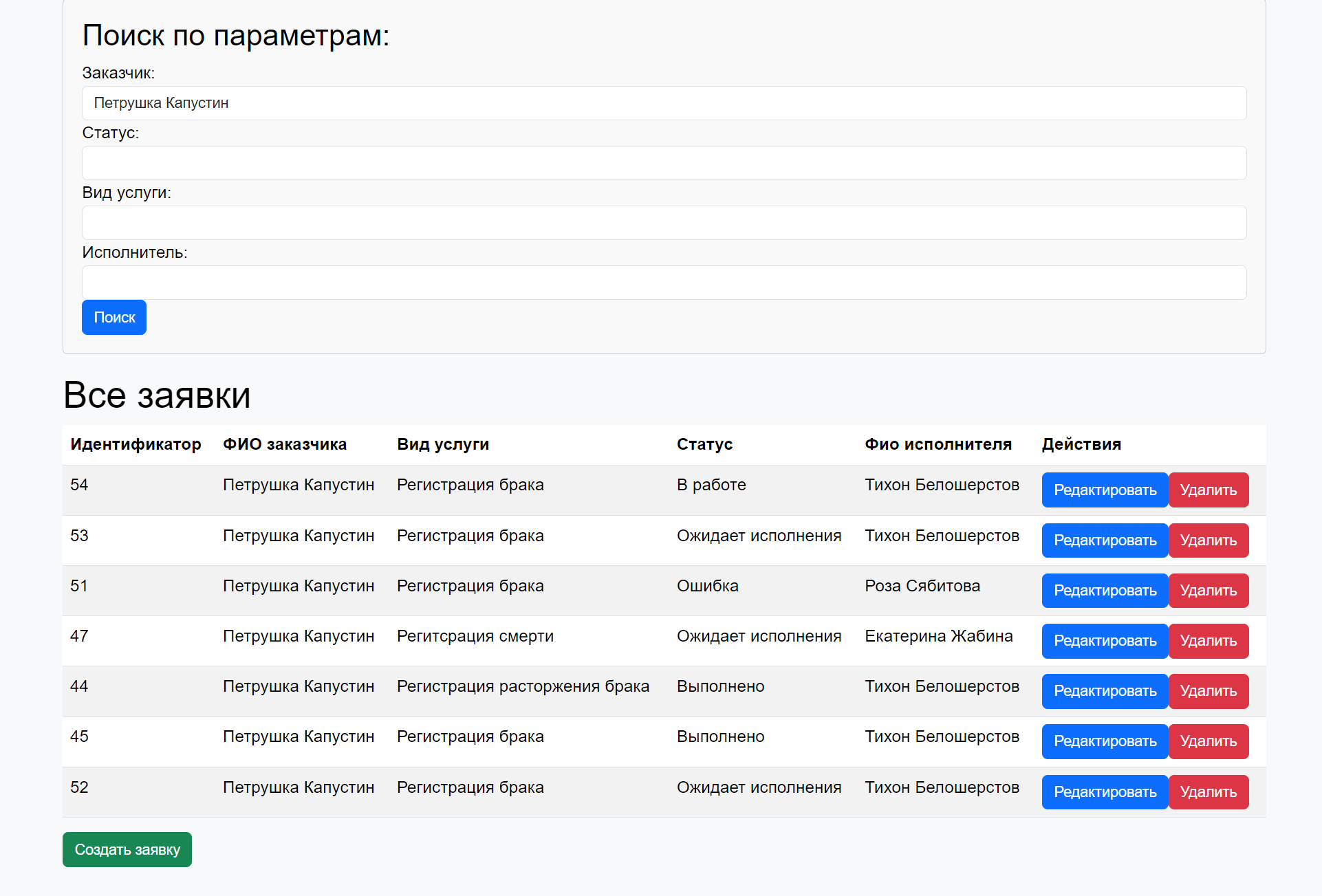


Рисунок 31 – Демонстрация работы фильтрации по имени заказчика

Редактирование и создание заявки:

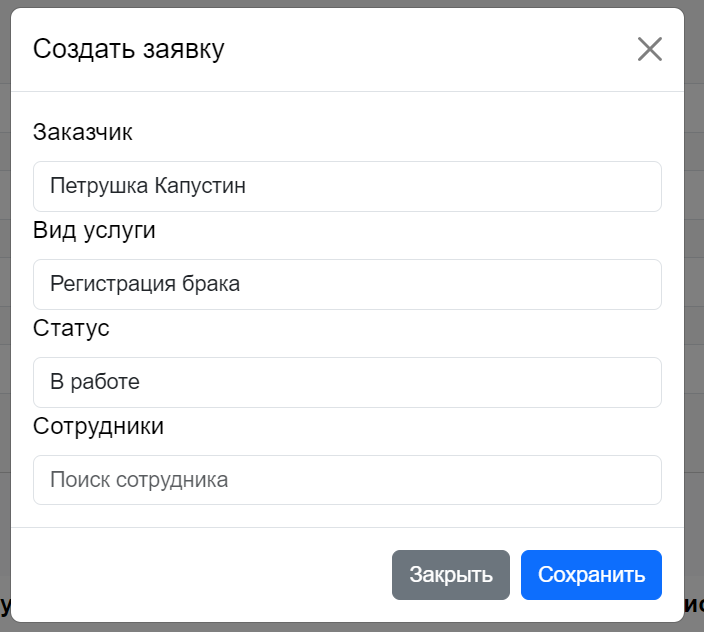


Рисунок 32 – Окно создания заявки

Интерфейс пользователя без прав администратора:

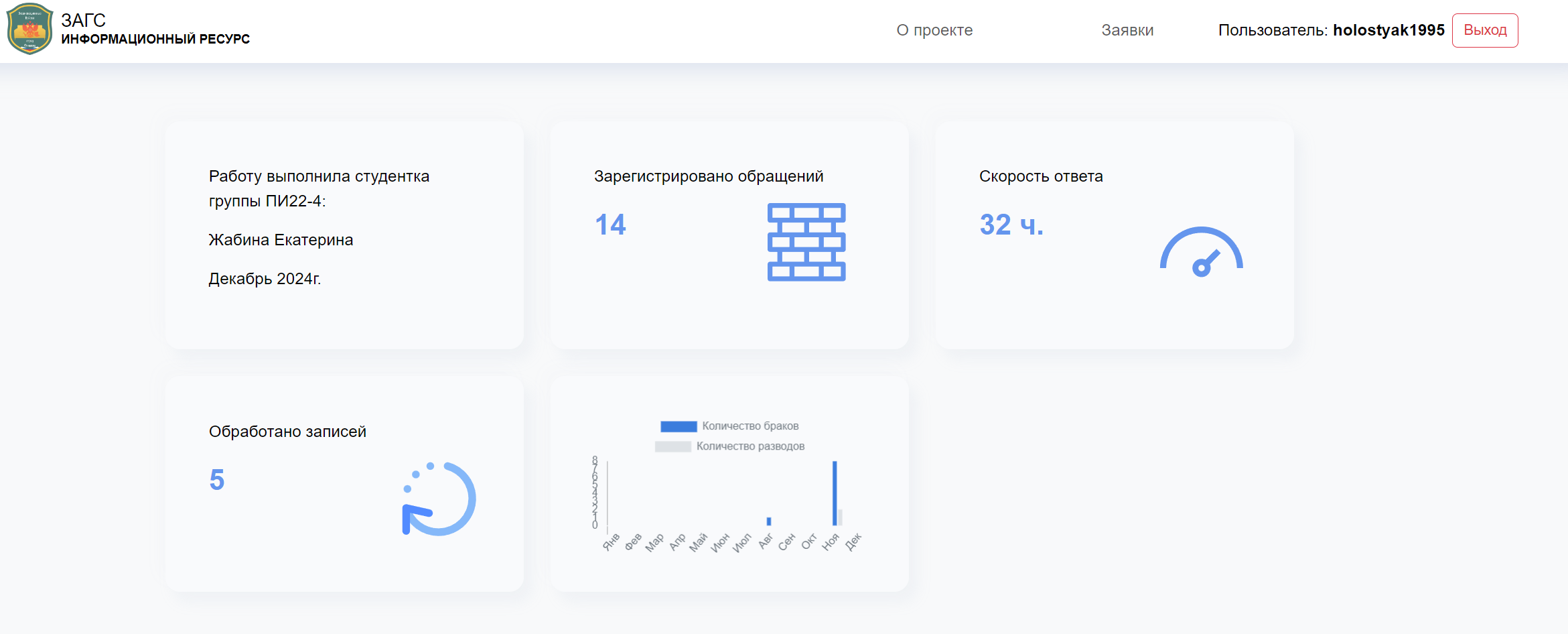


Рисунок 33 – Главная страница пользователя без прав администратора

В отличие от интерфейса которые видят администраторы тут отсутствует раздел с сотрудниками.

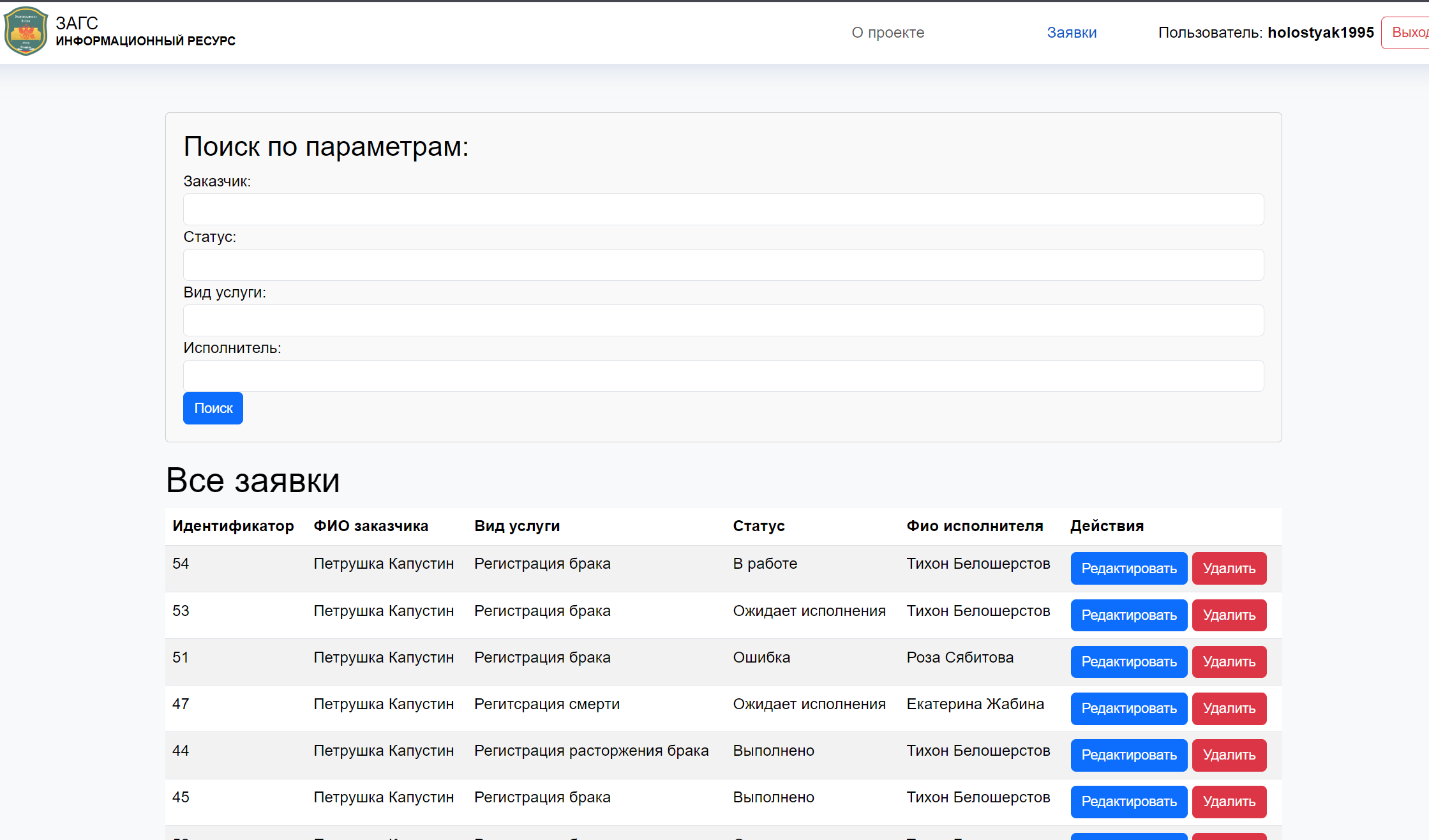


Рисунок 34 – Страница с заявками для пользователя без прав администратора

Пользователь видит только свои заявки и только их может редактировать и удалять.

# ЗАКЛЮЧЕНИЕ

В результате работы была создана информационная система отдела ЗАГС. Все обозначенные задачи и цели были мной реализованы. Приложения является полностью рабочим и функционирующим. Также оно обладает простым интерфейсом, который будет понятен любому пользователю. Данное приложение может быть использовано в реальных условиях для улучшения работы отделов ЗАГС. Работа над этим проектом позволила мне углубить и получить знания в области программирования и разработки современных приложений. Полученные навыки и опыт будут очень полезны для дальнейшей карьеры в сфере разработки ПО.

# СПИСОК ЛИТЕРАТУРЫ

1. Джошуа Блох. Java. Эффективное программирование. – М.: Вильямс, 2008. – 352 с.

2. Крейг Уоллс, Джейсон Широска. Spring в действии. – М.: ДМК Пресс,

2017. – 624 с.

3. Лафоре Р. Объектно-ориентированное программирование в Java. – М.:

Вильямс, 2003. – 550 с.

4. Мартин Фаулер. Архитектура корпоративных программных приложений. – М.: Вильямс, 2003. – 576 с.

5. Мартин Роберт. Чистый код: создание, анализ и рефакторинг. – СПб.:

Символ-Плюс, 2016. – 464 с.

# ПРИЛОЖЕНИЯ

Полный код по ссылке на GitHub: https://github.com/Katerina0088/Gorodovoy

Часть кода программы:



@RestController  
@RequestMapping("/employee")  
public class EmployeeController {  
  
 @Autowired  
 private EmployeeService employeeService;  
 @Autowired  
 private UserService userService;  
 @Autowired  
 private TaskService taskService;  
  
 @GetMapping("/list")  
 public ResponseEntity<List<Employee>> getAllEmployees() {  
 try {  
 List<Employee> employeesList = employeeService.getAll();  
 return ResponseEntity.*ok*(employeesList);  
 } catch (Exception e) {  
 return null;  
 }  
 }  
  
 @PostMapping("/add")  
 public ResponseEntity<String> addEmployee( @RequestBody employeePrivilegeReq employeePrivilegeReq) {  
 try {  
 Employee employee = employeePrivilegeReq.getEmployee();  
  
 employeeService.deleteByUserId(employee.getUserId());  
 employeeService.add(employee);  
 userService.saveUserPrivilegeRole( employee.getUserId(), employeePrivilegeReq.getPrivilegeRole());  
 return ResponseEntity.*ok*("Сотрудник добавлен");  
 } catch (Exception e) {  
 return ResponseEntity.*status*(HttpStatus.*BAD\_REQUEST*)  
 .body("Ошибка при добавлении сотрудника: " + e.getMessage());  
 }  
 }  
  
 @DeleteMapping("/deleted")  
 public ResponseEntity<String> deleteEmployee( @RequestBody Long userId) {  
 try{  
 Long emploueeId = employeeService.getEmployeeByUserId(userId);  
 TaskFilter taskFilter = new TaskFilter();  
 taskFilter.setEmployeeId(emploueeId);  
 List<Task>tasks = taskService.getTasksByFilters(taskFilter);  
  
 for (Task task : tasks) {  
 task.setEmployeeId(null);  
  
 taskService.updateTask(task);  
  
 }  
  
  
 employeeService.deleteByUserId(userId);  
 userService.saveUserPrivilegeRole(userId, "ROLE\_USER");  
  
 return ResponseEntity.*status*(HttpStatus.*OK*)  
 .body("Пользователь удален из сотрудников" );  
 }catch (Exception e) {  
 return ResponseEntity.*status*(HttpStatus.*BAD\_REQUEST*)  
 .body("Ошибка при удалении сотрудника: " + e.getMessage());  
 }  
 }  
}



@Controller //обработка переходов по сайту, за это отвечает контроллер  
public class GorodovoyController {  
  
 // подключаем репозиторий для работы с базой по CRUD  
 private final EmployeeRepository employeeRepository;  
  
 // подключаем репозиторий для работы с базой по CRUD  
 private final TaskRepository taskRepository;  
 // подключаем репозиторий для работы с базой по CRUD  
 private final RoleRepository roleRepository;  
 // подключаем репозиторий для работы с базой по CRUD  
 private final TypeOfServiceRepository typeOfServiceRepository;  
 // подключаем репозиторий для работы с базой по CRUD  
 private final UserRepository userRepository;  
  
  
  
 // Подключение конфигурации для наших страниц, чтобы не прописывать путь до них в коде контроллера  
 private final String home;  
  
  
 // Подключение конфигурации для наших страниц, чтобы не прописывать путь до них в коде контроллера  
 private final String administrator;  
 // Подключение конфигурации для наших страниц, чтобы не прописывать путь до них в коде контроллера  
 private final String task;  
 private final UserService userService;  
 private final TaskService taskService;  
 private final StatusRepository statusRepository;  
  
 public GorodovoyController(@Qualifier("administrator") String administrator, EmployeeRepository employeeRepository,  
 TaskRepository taskRepository, RoleRepository roleRepository, TypeOfServiceRepository typeOfServiceRepository,  
 UserRepository userRepository, @Qualifier("home") String home, @Qualifier("tasks") String task, UserService userService,  
 TaskService taskService, StatusRepository statusRepository) {  
 this.administrator = administrator;  
 this.employeeRepository = employeeRepository;  
 this.taskRepository = taskRepository;  
 this.roleRepository = roleRepository;  
 this.typeOfServiceRepository = typeOfServiceRepository;  
 this.userRepository = userRepository;  
 this.home = home;  
 this.task = task;  
 this.userService = userService;  
 this.taskService = taskService;  
 this.statusRepository = statusRepository;  
 }  
  
 // тут каждая функция обрабатывает определенный url ("/") значит начальная страница.  
 // Т.е. при переходе на главную страницу вызывается эта функция  
 @GetMapping({"/","/home"})  
 public String home( Model model) { // Model model - обязательный параметр  
 try {  
 ObjectMapper objectMapper = new ObjectMapper();  
  
 List<Task> tasks = taskRepository.findAllTasks();  
 Integer countTasks = tasks.size();  
 long avgDateTimeTask = taskService.calculateAverageExecutionTime();  
 Integer countReadyTasks = taskService.getCountReadyTasks();  
 int[] countServiceRastBrak = taskService.getCountForMonthService(1L);  
 int[] countServiceBrak = taskService.getCountForMonthService(4L);  
  
 String userName = userService.getCurrentUserName();  
 model.addAttribute("userName", userName );  
 model.addAttribute("title", "Загс. Главная страница."); // что вернем на эту страницу в виде данных  
 model.addAttribute("countTasks", countTasks);  
 model.addAttribute("avgDateTimeTask", avgDateTimeTask);  
 model.addAttribute("countReadyTasks", countReadyTasks);  
 model.addAttribute("countServiceRastBrak", objectMapper.writeValueAsString(countServiceRastBrak));  
 model.addAttribute("countServiceBrak", objectMapper.writeValueAsString(countServiceBrak));  
  
 return home; // тут просто вызывается шаблон, который вернем  
 }  
 catch (Exception e) {  
 model.addAttribute("errorMessage", "Ошибка при загрузке списка сотрудников: " + e.getMessage());  
 return null;  
 }  
 }  
  
 @GetMapping("/administrator")  
 public String administrator(Model model) {  
 try {  
 List<Employee> employees = employeeRepository.findAllEmployees();  
 List<Role> roles = roleRepository.findAllRoles();  
 List<User> users = userRepository.findAll();  
 String userName = userService.getCurrentUserName();  
  
 model.addAttribute("userName", userName );  
 model.addAttribute("employees", employees );  
 model.addAttribute("roles", roles );  
 model.addAttribute("users", users);  
 return administrator;  
 } catch (Exception e) {  
 model.addAttribute("errorMessage", "Ошибка при загрузке страницы administrator: " + e.getMessage());  
 return null;  
 }  
 }  
  
 @GetMapping("/tasks")  
 public String tasks(Model model) {  
 try {  
 User currentUser = userService.getCurrentUser();  
  
 List<Task> tasks;  
 TaskFilter taskFilter = new TaskFilter();  
 taskFilter.setUserId(currentUser.getId());  
 if("ROLE\_USER".equals(currentUser.getRole()))  
 tasks = taskService.getTasksByFilters(taskFilter);  
 else  
 tasks = taskRepository.findAllTasks();  
  
 List<Employee> employees = employeeRepository.findAllEmployees();  
 List<User> users = userRepository.findAll();  
 List<TypeOfService> services = typeOfServiceRepository.findAll();  
 List<Status> statuses = statusRepository.findAll();  
 String userName = userService.getCurrentUserName();  
 model.addAttribute("currentUser", currentUser);  
 model.addAttribute("employees", employees);  
 model.addAttribute("users", users);  
 model.addAttribute("tasks", tasks);  
 model.addAttribute("typeOfServices", services);  
 model.addAttribute("statuses", statuses);  
 model.addAttribute("userName", userName );  
  
 return task;  
 } catch (Exception e) {  
 model.addAttribute("errorMessage", "Ошибка при загрузке страницы task: " + e.getMessage());  
 return null;  
 }  
 }  
  
}



function saveTask(e) {  
 const userId = $("#userList")[0]?.dataset.userId ? $("#userList")[0].dataset.userId: null,//$(".userList")[0].dataset.userId,  
 serviceId = $("#typeOfServiceList")[0].dataset.serviceId,  
 statusId = $("#statusList")[0]?.dataset.statusId ? $("#statusList")[0].dataset.statusId: null,  
 employeeUserId= $("#userEmployeeList")[0]?.dataset.userEmployeeId ? $("#userEmployeeList")[0].dataset.userEmployeeId : null,  
 task = {  
 userId: userId ? parseInt(userId) : null,  
 serviceId: serviceId ? parseInt(serviceId) : null,  
 statusId: statusId ? parseInt(statusId) : null,  
 employeeId: employeeUserId ? parseInt(employeeUserId) : null  
 };  
  
  
  
 $.ajax({  
 url: '/tasks/add',  
 method: 'POST',  
 contentType: 'application/json',  
 data: *JSON*.stringify(task),  
 success: function(response) {  
 reloadTasks();  
 hideAddTaskModal()  
 showSuccessAlert();  
 },  
 error: function(error) {  
 showDangerAlert("Ошибка сохранения задачи");  
 *console*.log(`Ошибка сохранения задачи \n Error: ${error.message}`);  
 }  
 });  
}

function updateTask(e) {  
 const userId = $("#userList")[0]?.dataset.userId ? $("#userList")[0].dataset.userId: null,//$(".userList")[0].dataset.userId,  
 serviceId = $("#typeOfServiceList")[0].dataset.serviceId,  
 statusId = $("#statusList")[0]?.dataset.statusId ? $("#statusList")[0].dataset.statusId: null,  
 employeeUserId= $("#userEmployeeList")[0]?.dataset.userEmployeeId ? $("#userEmployeeList")[0].dataset.userEmployeeId : null,  
 taskId = $('.modal-button-save')[0]?.dataset.id,  
 task = {  
 userId: userId ? parseInt(userId) : null,  
 serviceId: serviceId ? parseInt(serviceId) : null,  
 statusId: statusId ? parseInt(statusId) : null,  
 employeeId: employeeUserId ? parseInt(employeeUserId) : null,  
 id: taskId  
 };  
 $('.modal-button-save').removeAttr('data-id');  
  
  
 $.ajax({  
 url: '/tasks/update',  
 method: 'POST',  
 contentType: 'application/json',  
 data: *JSON*.stringify(task),  
 success: function(response) {  
 reloadTasks();  
 hideAddTaskModal()  
 showSuccessAlert();  
 },  
 error: function(error) {  
 showDangerAlert("Ошибка обновления задачи");  
 *console*.log(`Ошибка обновления задачи \n Error: ${error.message}`);  
 }  
 });  
}